Collaborative Project

ASPIRE

Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications

FP7 Contract: ICT-*215417*-CP

WP4 – RFID Middleware programmability

Public report - Deliverable

ASPIRE Programmable Engine (APE) (Interim Version)

Due date of deliverable:	M24
Actual Submission date:	M24

Deliverable ID:	**WP4/D4.2a**
Deliverable Title:	Programmable RFID Solutions Specification (Interim Version)
Responsible partner:	AIT
Contributors:	Nikos Kefalakis - AIT John Soldatos - AIT Yongming Luo - AIT Mathieu David - AAU Sofyan M. Yousuf - OSI Didier Donsez - UJF Kiev Gama - UJF
Estimated Indicative Person Months:	18

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

Revision: 0.1
Dissemination Level: PU

Document Information

Document Name: **ASPIRE Programmable Engine (ASE) (Interim Version)**
Document ID: **WP4/D4.2a**
Revision: **1.2**
Revision Date: **14 January 2010**
Author: **AIT**
Security: **PU**

Approvals

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

Reviewers

Name	Organization	Date	Comments	Visa
Sofyan M. Yousuf	OSI	18/12/2009	Suggested corrections	

PMs Spend per Partner

Organization	PMs Spend	Comments
AIT		AIT is the main contributor of the work described within Sections 3, 5, 6, 8, 10 and Author of sections 1, 4.2, 5, 6, 8, 9, 10, 11 and Appendixes of the deliverable. AIT has also implemented and contributed this work to the AspireRFID project (http://wiki.aspire.ow2.org/)
AAU		Added Section 2
UJF		UJF is the author of section 4.3

OSI		OSI is the author of Section 12, part of Section 2 and reviewed the document

Document history

Revision	Date	Modification	Authors
0.1	02 Nov 09	TOC	Nikos Kefalakis
0.2	10 Nov 09	TOC Finalization and Chapters Assign.	Nikos Kefalakis
0.3	27 Nov 09	Added Section 4, Section 5, Section 8	Nikos Kefalakis
0.4	09 Dec 09	Added Section 2	Mathieu David
0.5	10 Dec 09	Augmented Section 2	Sofyan M. Yousuf
0.6	11 Dec 09	Added Section 10.1	Yongming Luo
0.7	15 Dec 09	Added Section 9, Section 10, Section 11	Nikos Kefalakis
0.8	17 Dec 09	Added Section 6	Nikos Kefalakis
0.9	17 Dec 09	Added Section 1	John Soldatos
1.0	18 Dec 09	Added Section 4.3	Didier Donsez, Kiev Gama
1.1	18 Dec 09	Added Section 12, Reviewed the Doc.	Sofyan M. Yousuf
1.2	18 Dec 09	Added Section 3, 4.1, Replaced 4.3.1 and 4.3.2, Final Corrections.	Nikos Kefalakis

Content

Section 1 Executive Summary

Among the main objectives of ASPIRE is to research, specify and implement domain specific languages (notably XML based) for specifying programmable / configurable RFID solutions, along with related run-time software that will enable their implementation over the ASPIRE middleware infrastructure. ASPIRE deliverable D4.4 (delivered as an initial version in September 2009) focuses on the specification of APDL (AspireRFID Process Description Language) a domain specific language for describing/configuring RFID solutions. The present deliverable is devoted to the description of a run-time middleware infrastructure that is able to resolve APDL to the number of configuration files, which are required for the deployment of an APDL solution over the ASPIRE middleware infrastructure. This run-time middleware is conveniently called ASPIRE Programmable Engine (APE) or (in short) PE (Programmable Engine).

The Programmable Engine bridges APDL with the underlying ASPIRE middleware infrastructure, through "hiding" the lower-level details of the ASPIRE middleware from the APDL developer. Thanks to the APDL and its respective PE, RFID developers are capable of assembling and configuring RFID solutions in a high-level language (using appropriate tools), in a way that is totally transparent to the low-level middleware libraries (such as those enabling filtering, collection and business event generation). Note that the PE is closely affiliated to the APDL (tight coupling relationship), given that the PE is bound to interwork with specific feature and functionalities of the APDL and vice versa. In addition to the affiliations between APDL and PE, there is also a direct relationship between the PE and the ASPIRE middleware architecture, given that the PE operates over the ASPIRE open source middleware infrastructure (as the later is provided in the scope of the AspireRFID OSS project (see: http://wiki.aspire.ow2.org/). Hence, the present deliverable illustrates the interfaces of the PE to the middleware building blocks of the ASPIRE architecture.

The PE comprises two dual types of functionalities: (a) Encode functionalities enabling the encoding and subsequent mapping of an APDL compliant instance to the ASPIRE middleware configuration files comprising the RFID solution, (b) Decode functionalities enabling the decoding of an RFID solution to an APDL file for subsequent processing and related changes in the solution. Both functionalities are presented in the deliverable, along with relevant APIs (Application Programming Interfaces).

ASPIRE envisages the use of tools for editing the APDL language and the associated PE configuration parameters. This deliverable presents a proof of concept of the related tools, with particular emphasis on the Business Process Workflow Management Editor (BPWME), which enables the graphical modeling of APDL compliant RFID based business processes. Along with the tools, the deliverable provides concrete examples and use cases associated with the operation of the PE.

The deliverable includes also a preliminary evaluation of the suggested approach to programmable RFID development and deployment. In particular, it is shown that developing an RFID solution using a PE based approach, results in a significant reduction to the number of steps required from an inception of an RFID solution to its deployment. Specifically, the deliverable manifests the differences in complexity and steps required for two different configuration methods, one involving the use of the PE and the other the conventional integration of RFID solutions. However, it is also concluded that the PE approach (though general) has certain limitations, since it is not directly applicable to diverse domains other than logistics and supply chain management. Overall, the present deliverable emphasizes on the first specification and implementation of the PE. A later and final version of this deliverable is expected to enhance the functionalities offered by the PE, in accordance with the evolution of the APDL (as part of the related ASPIRE deliverable D4.4).

Section 2 Introduction

RFID technology has advanced significantly over the past few decades. Rapid developments of low cost microelectronics and radio frequency transceivers have considerably reduced size and costs of high frequency and ultra-high frequency RFID transceivers allowing longer reading ranges and faster reading rates than before. The technology is now viable to newer novel applications with higher mobility and large number of tagged items. However, unlike conventional scenarios, these new applications require a more robust and complex middleware platform in order to cover issues at different layers of the communication architecture, from different business contexts. This complexity has left several open research issues in RFID middleware design that still pose a high entry cost for RFID technology adopters, mainly SMEs.

The research carried out in ASPIRE will provide a radical change in the current RFID deployment paradigm through innovative, programmable, royalty-free, lightweight and privacy friendly middleware. ASPIRE solutions will be open source and royalty free, which will bring an important reduction of the Total Cost of Ownership, and at the same time programmable and lightweight in order to be backwards compatible with current IT SME infrastructure. Additionally, ASPIRE will be designed as privacy friendly which means that future privacy features related to RFID can be easily adopted by the platform. Finally, ASPIRE will act as a main vehicle for realizing the proposed swift in the current RFID deployment paradigm. Portions (i.e. specific libraries) of the ASPIRE middleware will be hosted and run on low-cost RFID-enabled microelectronic systems, in order to further lower the TCO in mobility scenarios (i.e. mobile warehouses, trucks). Hence, the ASPIRE middleware platform will be combined with innovative European developments in the area of ubiquitous RFID-based sensing (e.g., physical quantities sensing (temperature, humidity, pressure, acceleration), mobile, low-cost); towards enabling novel business cases that ensure improved business results.

This new middleware paradigm will be particular beneficial to European SMEs, which are experience significant cost barriers to RFID deployment. 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.

A great deal of ASPIRE research will be devoted towards development of the ASPIRE middleware infrastructure with programmability. The aim is to allow development and reuse of RFID solutions through minimal coding effort. The core of the ASPIRE programmability will therefore be an engine capable of mapping high level (business semantics) to low-level middleware abstractions and information flows between them. This engine will orchestrate tags, readers, filters and events into RFID solutions.

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.

Hence this deliverable presents the specifications of the ASPIRE Programmable Engine (APE). This module is an interface between the user and the ASPIRE middleware to facilitate the deployment of a specific scenario. From a well defined business scenario, the user can express the different actions in a business process language, the AspireRFID Process Description Language (APDL), that the Programmable Engine (PE) will convert to a language understandable by the ASPIRE middleware.

The Programmable Engine is a run-time middleware module which will take as input an APDL XML file and be able to:

- "Encode"/"program" the AspireRFID middleware.
- "Decode" the AspireRFID middleware (by retrieving all the information from the AspireRFID middleware and create an APDL XML file by request).

As a first step, we will create a "static" ApireRFID plug-in client to encode/decode the AspireRFID middleware through the Programmable Engine (PE).

At a second step we will investigate the creation of a "dynamic" client which will be able to interact in real time with the Business Process Workflow Management Editor (BPWME) plug-in.

This deliverable will briefly present the AspireRFID Process Description Language in Section 3, and the relation between the Programmable Engine and the other components in Section 4. The specifications of the Programmable Engine will be described in the following sections: the PE interfaces in Section 5 and the encode API implementation in Section 6. How the PE Changed the AspireRFID Cofiguration Process will be presented in Section 8 and an example of the PE encoding in Section 9 will follow. A brief introduction of the Business Process Management Workflow Editor (BPMWE) will be presented in Section 10. The current implementation status and future steps will be detailed in Section 11. Ultimately, Section 12 will conclude this deliverable.

Section 3 AspireRFID Process Description Language

The ASPIRE Programmable Meta-Language is a language created from ASPIRE with the intention to be able to fully describe an Open Loop RFID Business Process and ultimately be used from the Programmable Engine to configure an AspireRFID middleware running instance to serve the described Business Processes. APDL is a combination of a set of specifications which are the following:

- Logical Readers Specs
- ECSpecs
- Master Data Document
- Middleware Management/Configuration Data (Modules Endpoints)

All the above are augmented with design data for the visualization of the RFID solution to the BPWME (Business Process Workflow Management Editor). To achieve that APDL has adopted XPDL V1.0 specifications [27] from where it has used many of it's concepts and definitions.

Let us briefly remember the AspireRFID Process Description Language (APDL) specification structure. The APDL has finite dendritic structure as shown in the Figure 1 below and the "parent" object that is able to contain the description of a complete open loop supply chain management scenario is the Open Loop Composite Business Process (<OLCBProc/>). "OLCBProc" is consisted of a list of objects called Close Loop Composite Business Process (<CLCBProc/>) that are capable of describing a complete close loop supply chain scenario and the object of Transitions (<Transitions/>) which carries the Close Loop Composite Business processes context-related semantics description of Transitions between them which is based on the XPDL V1.0 specifications [27].

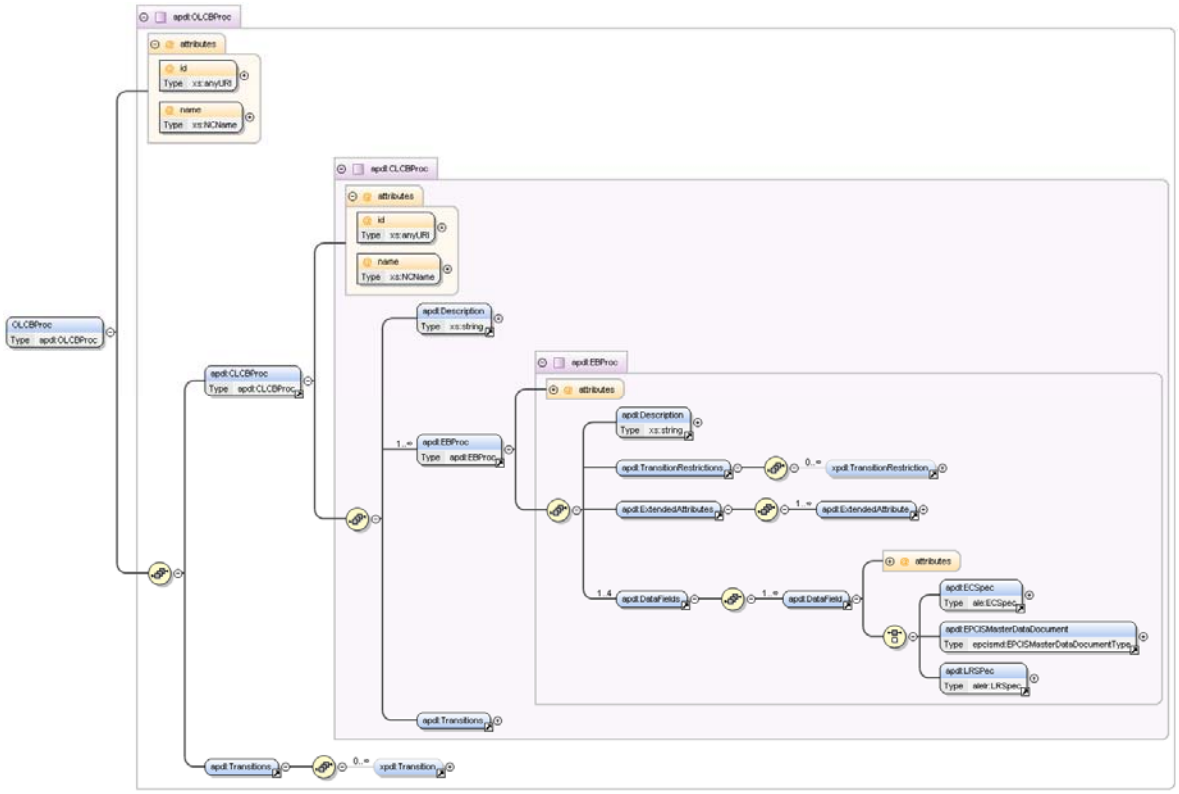

Figure 1 APDL's Schema graphical representation

Each of the "CLCBProc" objects are consisted of a list of **Elementary Business Process** (<EBProc/>) Objects that describe the elementary Business Transactions by providing:

- The various AspireRFID Middleware basic configuration variables,
- The required variables for the workflow graphical representation (x/y coordinates),
- And the required Data fields (<DataField/>) which includes:
    - The transactions required ECSpec,
    - The transactions required LRSpec
    - And the transactions required Master Data.

And the object of **Transitions** (<Transitions/>) which carries the Elementary Business Processes context-related semantics description of Transitions between them which is based on the XPDL V1.0 specifications [27].

More details about APDL can be found at Deliverable D4.4a [24] and an updated Schema of the language can be found at Appendix I.

Section 4 Role within the AspireRFID Architecture

The AspireRFID Programmable Engine (PE) module as we can see in Figure 2 below resides between the AspireRFID IDE environment and the rest of the AspireRFID architecture. More specifically it will be used as median for the AspireRFID Business Process Workflow Management Editor (BPWME) plug-in to "encode" the produced APDL xml file from it to the ASPIRE middleware. Vice versa it will be able to "decode" from the ASPIRE middleware an already encoded Business Process configuration and send it back to the BPWME. Moreover the Programmable Engine (PE) is providing a standalone client which will be able to encode to and decode from the AspireRFID middleware APDL xml files.

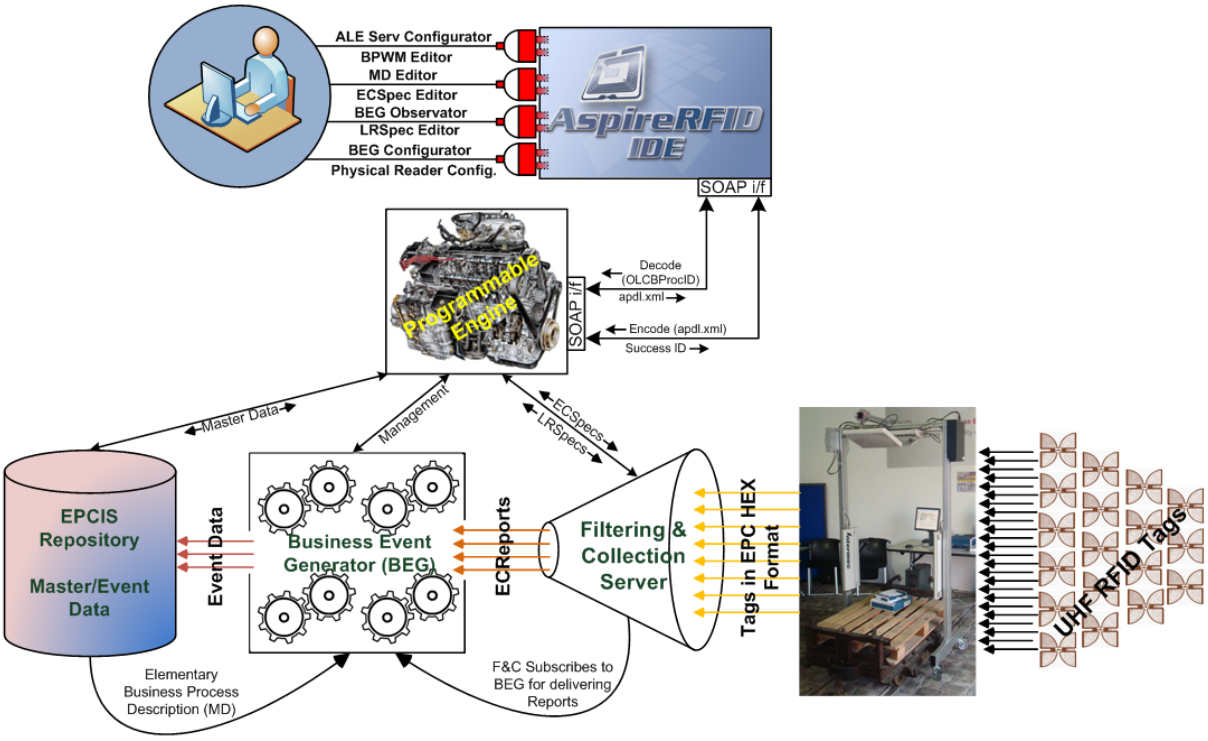

Figure 2 Programmable Engine role in the AspireRFID Architecture

The PE is fully based on Service Oriented Architecture (SOA) and it reveals two interfaces, called "encode" and "decode" that are analyzed at Section 5, which are using the SOAP protocol for exchanging messages. The main Objects that are exchanged from the PE interface are APDL xml files that are used as input for the encoding implementation and as output for the decoding.

For configuring the three basic AspireRFID modules, which are the Filtering and Collection (F&C), the Business Event Generator (BEG) and the Information Service repository (EPCIS), PE takes advantage of their specific and already defined interfaces that are also using SOAP protocol for exchanging messages. In the next paragraphs we are going to describe briefly their interfaces and the PE's communication type with the AspireRFID "underlying" modules. Moreover in the

following Sections we are going to analyze the PE's Interface and how this is implemented from ASPIRE.

At this point it worth's to mention that AspireRFID architecture uses Fosstrak's [1] EPCIS and F&C (ALE) implementations that ASPIRE has enhanced and tailored to meet its needs.

4.1 Filtering and Collection

The role of the filtering and collection module (F&C) within the ASPIRE architecture is mainly to carry out processing to reduce the volume of captured RFID data and to transform raw tag reads into streams of events that are suitable for the application logic of Business Event Generator module. The main specifications that define the F&C module functionality are the ECSpecs and the LRSpecs. For an application, which in our case is the Programmable engine, to be able to configure these specifications the ALE and the ALE-LR interfaces use is required. This use co

4.1.1 ALE Client

Through a client that implements the ALE interface [2] and with the use of SOAP protocol the Programmable Engine may define and manage event cycle specifications (ECSpecs). The methods that ALE Interface exposes and are used from the Programmable Engine to configure the F&C server are:

- The "**define**(specName : String, spec : ECSpec) : void" which Creates a new ECSpec having the name specName, according to spec.
- The "**getECSpecNames**() : List<String>" which Returns an unordered list of the names of all ECSpecs that are visible to the caller.
- The "**undefine**(specName : String) : void" which Removes the ECSpec named specName that was previously created by the define method.
- The "**subscribe**(specName : String, notificationURI : String) : void" which Adds a subscriber having the specified notificationURI to the set of current subscribers of the ECSpec named specName.
- And the "**unsubscribe**(specName : String, notificationURI : String) : void" which Removes a subscriber having the specified notificationURI from the set of current subscribers of the ECSpec named specName.

4.1.2 ALE-LR Client

The Logical Reader API [2] provides a standardized way for an ALE client to define a new logical reader name as an alias for one or more other logical reader names. The API also provides 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. Through a client that implements the ALE-LR interface and with the use of SOAP protocol the Programmable Engine may define Logical Reader specifications (LRSpecs). The methods that ALE-LR Interface exposes and are used from the Programmable Engine to configure the F&C server are:

- The "**getLogicalReaderNames**() : List<String>" which Returns an unordered list of the names of all logical readers that are visible to the caller. This list SHALL include both composite readers and base readers.
- The "**define**(name : String, spec : LRSpec) : void" which Creates a new logical reader named name according to spec.
- And the "**update**(name : String, spec : LRSpec) : void" which Changes the definition of the logical reader named name to match the specification in the spec parameter.

4.2 Business Event Generator

The role of the BEG is to automate the mapping between reports stemming from F&C and IS events. The Business event generation (BEG) module associates business-context information (Master Data) with event data. The data is stored in the Information Services module repository as Event Data and are mapping associated events with a company's master data.

4.2.1 Functionality and relation with the Programmable engine

In order for BEG to create aforementioned Event Data it needs the EPCIS's offered services URLs (Capture/Query) and most importantly the appropriate information from the EPCIS repository (Master Data). These necessary data for the proper population of the EPCIS events are retrieved from the EPCIS repository, and more specifically the data defined at the BusinessTransaction's Attributes vocabulary [23], by using EPCIS's query interface. So for the Programmable Engine to be able to perform the required management over the BEG as shown in Figure 2 above it should be able to retrieve the EPCIS's running instance Query End-Point from a given APDL's EBProc and use it to get the VocabularyElementType [8] for a specific Elementary Business Processes (EBProc) ID. Furthermore the PE should be able to retrieve the EPCIS Client Capture End-Point from a given APDL's EBProc so as to complete all the required information to be able to use the BEG client service and more specifically the "startBegForEvent" as shown in Table 4 below.

4.3 EPC Information Services

The EPCIS is a component responsible for receiving application-agnostic RFID data from the filtering and collection layer, translating that data into business events, and optionally storing them into an EPCIS repository.

The EPCIS provides standard interfaces that allow EPC-related data to be captured and queried through a predefined set of operations. The ASPIRE EPC EPCIS must provide the two corresponding interfaces between the filtering & collection middleware and upstream layers (i.e. business event generation modules or host applications), illustrated in the upper layers of on figure. In particular: the Capture API and the Query API defined in the EPC Global' EPCIS specification [8]

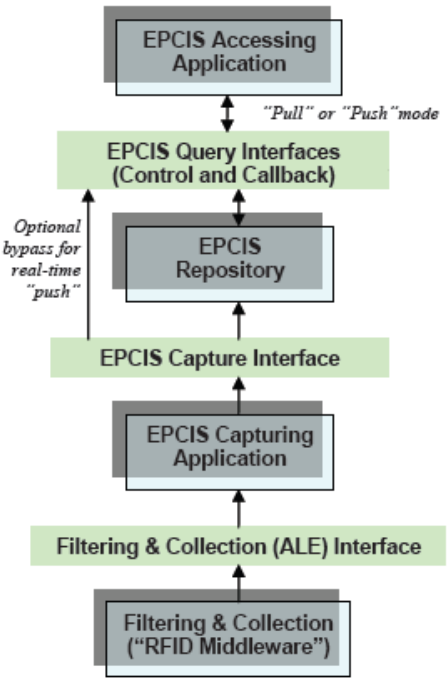

Figure 3 Layers and interfaces concerning the EPCIS [8]

4.3.1 Capture Client

Because the Programmable Engine is not related with the Event Data creation the Capture Client uses only the Master Data capture Interface provided by ASPIRE [see D3.4a Section 10.2.1]. This Interface is used to Store the required Master Data to the EPCIS's vocabularies that are eventually used from the BEG engine for the Event Data "production".

4.3.2 Query Client

At runtime the Programmable Engine requires information from the EPCIS's Stored Master Data so as to Correctly Update/Save new ones thru the Capture client. To achieve that the PE is using the EPCIS's SimpleMasterDataQuery Interface [8]. EPCIS provides the SimpleMasterDataQuery Interface which provides predefined queries that Programmable Engine may invoke using the poll methods of the EPCIS Query Control Interface.

Section 5 PE Interfaces

This section defines normatively the PE Encode and Decode API. The External Interface is defined in the following two sections (5.1 and 5.2) and the implementation is described in Section 6.

The programmable engine exposes two Interfaces. The first one gives the ability to the Clients to "encode" the AspireRFID middleware with the use of an APDL document. And the second one gives the ability to the client to "decode" and retrieve an already configured APDL document from the AspireRFID middleware by giving the ID of the Parent Business Process.

5.1 Encode API

As mentioned before the First Interface exposed from a Programmable Engine implementation is the Encode interface. Encode API is consisted from only one method which requires as input an APDL XML document, which contains the hole RFID Business Process description, and returns an integer code which denotes if the execution of this specific command is successful or not. With this method the PE undertakes the task of configuring a running instance of AspireRFID middleware with the Business Processes described to the given OLCBProc Object.

Method	Argument/Result	Type	Description
encode	openLoopCBProc	OLCBProc	This method configures the AspireRFID middleware to serve the described Business Processes from the given APDL XML document. If the encode is successful the reply ID will be "400" if not the reply ID will be "425".
	[result]	Integer	

Table 1 PE's Encode Interface methods

The primary Datatypes associated with the PE Encode API are the "*Integer*" type, which denotes the success id of the "encode" execution, and the "*OLCBProc*" type which contains the APDL object and the Business Process Description.

5.2 Decode API

The second Interface exposed from a Programmable Engine implementation is the Decode Interface. The Decode API is consisted from only one method "decode" which requires as input a String of an OLCBProc ID which was primarily been used to configure an AspireRFID middleware running instance. This service will be used from clients that want to alter an already encoded Business Process to the AspireRFID middleware by retrieving it ("decode") make the required changes and then reconfiguring back the middleware ("encode").

Method	Argument/Result	Type	Description
decode	openLoopCBProcID	String	This method returns an OLCBProc

	[result]	OLCBProc	Object which is retrieved from an AspireRFID middleware running instance by a prior configured (encoded) OLCBProc by giving that object's ID.

Table 2 PE's Decode Interface methods

The primary Datatypes associated with the PE are the "*String*" type, which is the ID of a prior configured OLCBProc, and the OLCBProc type which is constricted and returned from the decode service.

Section 6 PE Encode API Implementation

In the previous section we have described the Programmable Engine's Interfaces so in this section we are going to describe how ASPIRE is implementing PE's API. The PE's implementation requires the use of various existing API's such as:

- ALE Reading API,
- ALE Logical Reader API,
- BEG API
- EPCIS Predefined Queries (SimpleMasterDataQuery and SimpleMasterDataCapture)

So we are also going to describe how these Interfaces are applied from the PE to configure the AspireRFID middleware.

6.1 Encode Service

The first PE API Implementation that we are going to analyze is the Encode Service. Figure 4 below depicts the various steps PE implementation follows to "encode" an APDL xml document into a running instance of AspireRFID middleware.

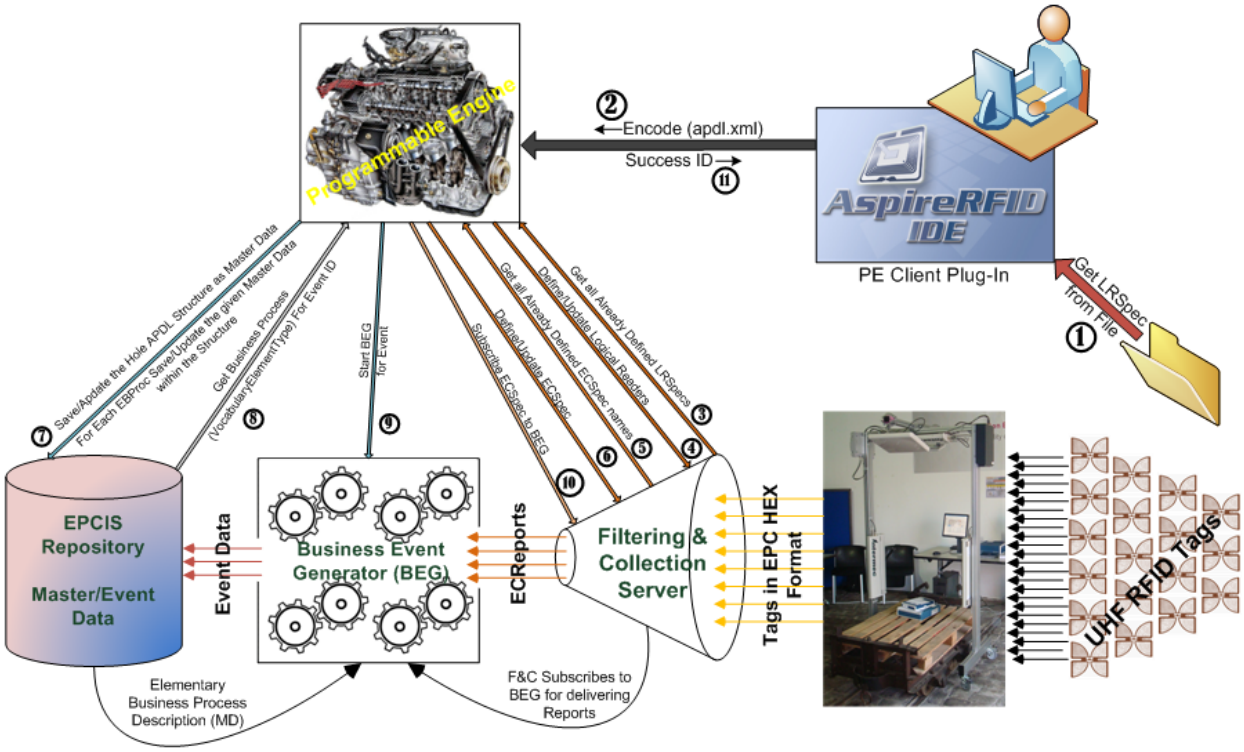

Figure 4 Programmable Engine's Encode Steps

Firstly the PE Client should retrieve from a file (Step 1) map it to an OLCBProc Java Object with the help of JAXB (Java Architecture for XML Binding) and deliver to the PE server interface (Step 2) an APDL XML document encapsulated inside a

SOAP message. At Appendix III the PE's encode SOAP Interface can be found. For the Web Services Implementation Apache CXF framework was used and more specifically with Servlet Transport [26].

6.1.1 APDL Analysis and System Configuration

As soon as an OLCBProc Object arrives to the PE it is analyzed into their CLCBProcs and into their EBProcs. Now for each EBProc and by taking in consideration their parent Objects (Attributes and IDs) the required specification files are build to configure an AspireRFID middleware running instance. Ulterior objective of this analysis is to fill an Object which we conveniently call "ProcessedEBProc" and is described in Table 3 below which will eventually include all the required information for configuring the AspireRFID middleware for a single Elementary Business Process.

Attribute Name	Type	Description
id	String	The ID of the EBProc
name	String	The name of the EBProc
ecSpec	ECSpec	The extracted ECSpec file
lrSpecs	Hashtable<String,LRSpec>	A Hashtable with key value the Logical Reader name and it's LRSpec as value.
epcisMasterDataDocument	EPCISMasterDataDocumentType	All the Master Data that are required to be stored for a specific EBProc at the EPCIS repository.
ecSpecSubscriptionURI	String	The URI where the Defined ECSpec should be subscribed. Actually this URI is where BEG is accepting reports for this EBProc.
definedECSpecName	String	The name of the ECSpec that will be Defined
aleClientEndPoint	String	The URI where the ALE Reading API is revealed
aleLrClientEndPoint	String	The URI where the ALE Logical Reader

		API is revealed
epcisClientCaptureEndPoint	String	The URI where the EPCIS Capture Interface is revealed.
epcisClientQueryEndPoint	String	The URI where the EPCIS Query Interface is revealed.

Table 3 ProcessedEBProc Object

6.1.1.1 ALE-LR Setup

For each EBProc the APDL language gives the ability to the User to describe many Logical Reader specifications (LRSpecs) that are going to be used from the ECSpec. That is why they are saved at a Hashtable Java Object, when the received APDL document is analyzed for every EBProc it describes, in an "LRSpec name"/ "LRSpec" pair manner.

So the third Step as shown in Figure 4 above is to get all the Already Defined LRSpec names from the Running Instance of the AspireRFID middleware, which is going to be configured, so as at the next step to "Define" only Logical Readers that have not been priory been defined yet and "Update" Logical Readers that have been (Step 4).

For the ALE-LR Setup three methods from the ALELR interface [2] are used:

- The "**getLogicalReaderNames**() : List<String>" which Returns an unordered list of the names of all logical readers that are visible to the caller. This list SHALL include both composite readers and base readers.
- The "**define**(name : String, spec : LRSpec) : void" which Creates a new logical reader named name according to spec.
- And the "**update**(name : String, spec : LRSpec) : void" which Changes the definition of the logical reader named name to match the specification in the spec parameter.

6.1.1.2 ALE Setup

The ECSpec required for the AspireRFID configuration is directly taken from each EBProc "DataField" the only tuning done is to concatenate to Every ECReport name the EBProc's ID, with the "@" symbol between them, which will be later be used from the BEG so as to distinguish the received ECReports from the various EBProc's configurations.

In a same logic at Step 5 (Figure 4) the PE implementation gets all the defined ECSpec names, which have been prior defined to the AspireRFID running instance, so as to "Define" ECSpecs that do not prior exist and "Undefine/Define" the existing ones so as to get updated (Step 6). After Configuring BEG and id ready to receive ECReports the next Step (Step 10) would be to Subscribe the

"Defined" ECSpec, from the previous step (Step6), to the BEG Running instance ("ecSpecSubscriptionURI" Table 3).

For the ALE Setup five methods from the ALE interface [2] are used:

- The "**define**(specName : String, spec : ECSpec) : void" which Creates a new ECSpec having the name specName, according to spec.
- The "**getECSpecNames**() : List<String>" which Returns an unordered list of the names of all ECSpecs that are visible to the caller.
- The "**undefine**(specName : String) : void" which Removes the ECSpec named specName that was previously created by the define method.
- The "**subscribe**(specName : String, notificationURI : String) : void" which Adds a subscriber having the specified notificationURI to the set of current subscribers of the ECSpec named specName.
- And the "**unsubscribe**(specName : String, notificationURI : String) : void" which Removes a subscriber having the specified notificationURI from the set of current subscribers of the ECSpec named specName.

6.1.1.3 EPC Information Service Setup

The next thing that ASPIRE's PE takes care of is the configuration of the Master Data. For Each EBProc the Disposition, Transaction Type, Read Point and Business Step are "Saved" to the EPCIS Repository, if they do not exist, from the provided "EPCISMasterDataDocument" [8] with the help of the EPCIS Capture Interface. The EPCIS Capture End Point is provided from the EBProc's "ExtededAttributes" [24] with name "EpcisClientCaptureEndPoint". For the Business Transaction EPCIS vocabulary type the hole OLCBProc Structure is considered and a given EBProc is saved as the Child of it's CLCBProc and in its turn as a child of it's OLCBProc. For the last task the various different Object ID's are used to build the aforementioned structure. This concludes the Step 7 of the PE's configuration process.

6.1.1.4 BEG Setup

Continuing, the Programmable Engine retrieves the EPCIS Query End-Point provided from the EBProc's "ExtededAttributes" [24] with name "EpcisClientQueryEndPoint" and use it to get the VocabularyElementType [8] (Step 8) for a specific Elementary Business Processes (EBProc) ID which "conveniently happens" to be the same as the BusinessTransaction's ID that BEG is going to be configured to serve. After that the PE retrieves the EPCIS Client Capture End-Point from the EBProc's "ExtededAttributes" [24] with name "ECSpecSubscriptionURI" and use the "startBegForEvent" BEG client Service, which requires as input the "VocabularyElementType", the "repositoryCaptureURL" and the "begListeningPort" as shown in Table 4 below which attributes have already been retrieved from the previous steps, to configure the BEG's functionality for the given EBProc (Step 9).

Service Name	Input	Output	Info
getEpcListForEvent	String eventID	EventStatus*	Returns what is currently happening for a specific

			transaction
stopBegForEven	String eventID	boolean	Stop serving a specific Event (described at the Master Data)
getStartedEvents	---	List<String>	Get all the Event IDs that are currently been served from the BEG
startBegForEvent	VocabularyElementType[8] VocElem, String repositoryCaptureURL, String begListeningPort	boolean	Start a specific Event that is available at the EPCIS's Master Data
getEventList	String repositoryQueryURL	List <VocabularyElementType>	Get all the Available Events (ready to be served) from the EPCIS's repository Master Data

Table 4 BEG server Web Service Interface

* EventStatus is consisted of the following objects:
  - A String which denotes the Transactions ID named "transactionID"
  - And a list of Strings (ArrayList<String>) which stores all the read tags that are connected with the abovementioned Transaction named "epcList"

After Subscribing the ECSpec to the BEG running instance the final step (Step 11) is to send back to the Programmable Engine's client the Encode execution status. Where if it was completed successfully it returns an Integer number that equals "**400**" if not it returns "**425**".

Section 7 PE Decode API Implementation

7.1 Decode Service

(To be implemented and documented in Version 2 of this deliverable)

Section 8 How PE Changed the AspireRFID Configuration Process

In this section we are going to compare the two AspireRFID configuration methods, the conventional and the PE client method. We are going to compare them in regard of complexity and time required for a user (e.g. RFID integrator) to configure the AspireRFID middleware. In both cases we assume that all the configuration files are already predefined as comparing the generation of them is out of the scope of the Programmable Engine's features and capabilities.

8.1 Configuring in the Conventional Way

To achieve the configuration of the hole AspireRFID middleware for even a relatively simple scenario, like the one that is described in paragraph 9.1 below where we define only an EBProc, it would require to follow a few steps and to use a bunch of different AspireRFID "Configurators" (e.g. ECSpec Configurator, LRSpec Configurator and BEG configurator). Figure 5 below illustrates the different steps that an RFID integrator should follow to configure the AspireRFID middleware with the use of the aforementioned tools.

More specifically for defining an Elementary Business Process (as shown in Figure 5 below) we should:

- Use the LRSpec Configurator plug-in where the needed LRSpec xml file should be retrieved (Step 1) from the folder that was stored and then "Define" it to the ALE module paired up with the Logical Reader name (Step 2).
- The next step, with the help of the ECSpec configurator plug-in, would be to "Define" the required ECSpec file which should be retrieved from the folder that is stored (Step 3) and then be "Defined" paired up with the ECSpec name to the ALE module (Step 4).
- The next module that should be configured is the Business Event Generator and this is done with the use of the BEG configurator plug-in. With this plug-in firstly we retrieve all the available, already predefined, Business Events from the EPCIS repository (Step 5) and as soon as we choose the one that interests us, and set up a Port for the BEG to receive reports for the specific Business Event, we activate BEG to "serve" the Event (Step 6).
- And for the last Step by using again the ECSpec Configurator in "Subscribe" mode this time the already predefined ECSpec should be Subscribed (Step 7) to the Port that the BEG was prior (at Step 6) configured to receive Reports.

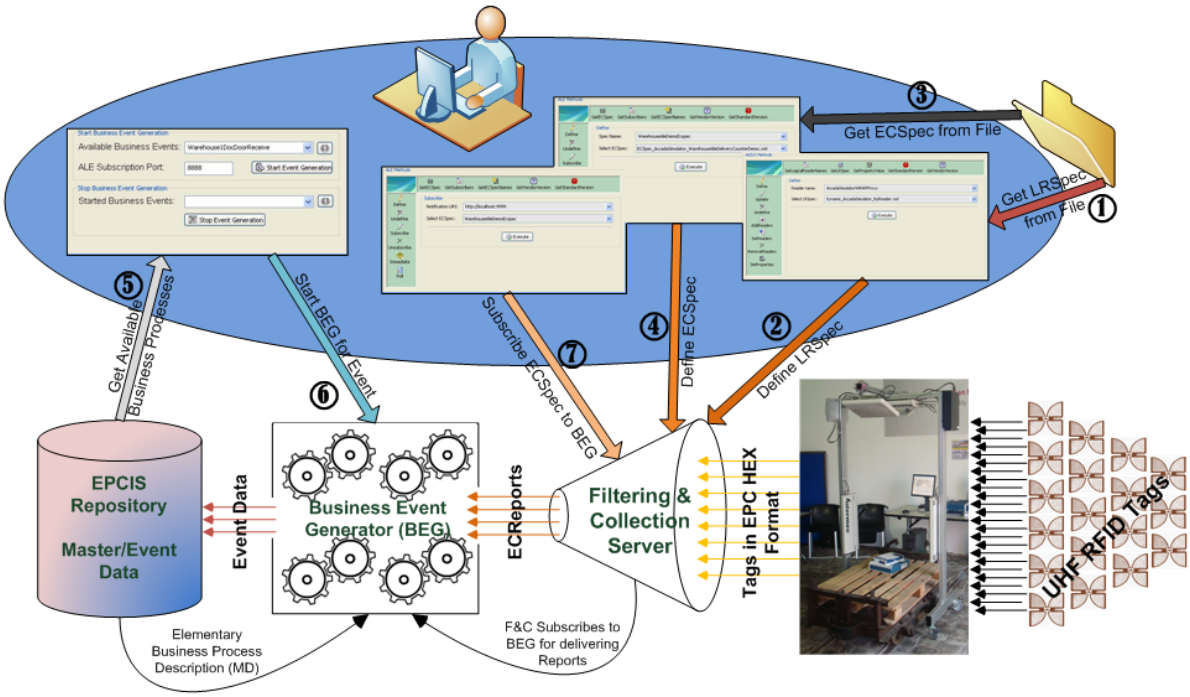

Figure 5 Required AspireRFID Configuring Steps without Programmable Engine

8.2 Configuring in the Programmable Engine Way

From the previous section we observe that, to configure the AspireRFID middleware with the conventional way, for just one Elementary Business Process **Seven Steps** are required from the User. In this section we will describe what is required from the User to configure the AspireRFID middleware with the use of Programmable Engine's plug-in (client) again for only one EBProc. It worth's to mention that even if we had to configure the AspireRFID middleware for "N" EBProc's (even for a complete Open Loop supply chain scenario) the steps that the user would have to follow would be the same as the ones described below and would have to follow them only one time.

So assuming that the APDL XML file has already been build for configuring the AspireRFID middleware with the PE's User Client plug-in the first step would be to retrieve the "apdl.xml" file from the folder that is stored (Step 1). And the second and final Step would be to use the "Encode" service of the PE's thru the PE Client (Step 2).

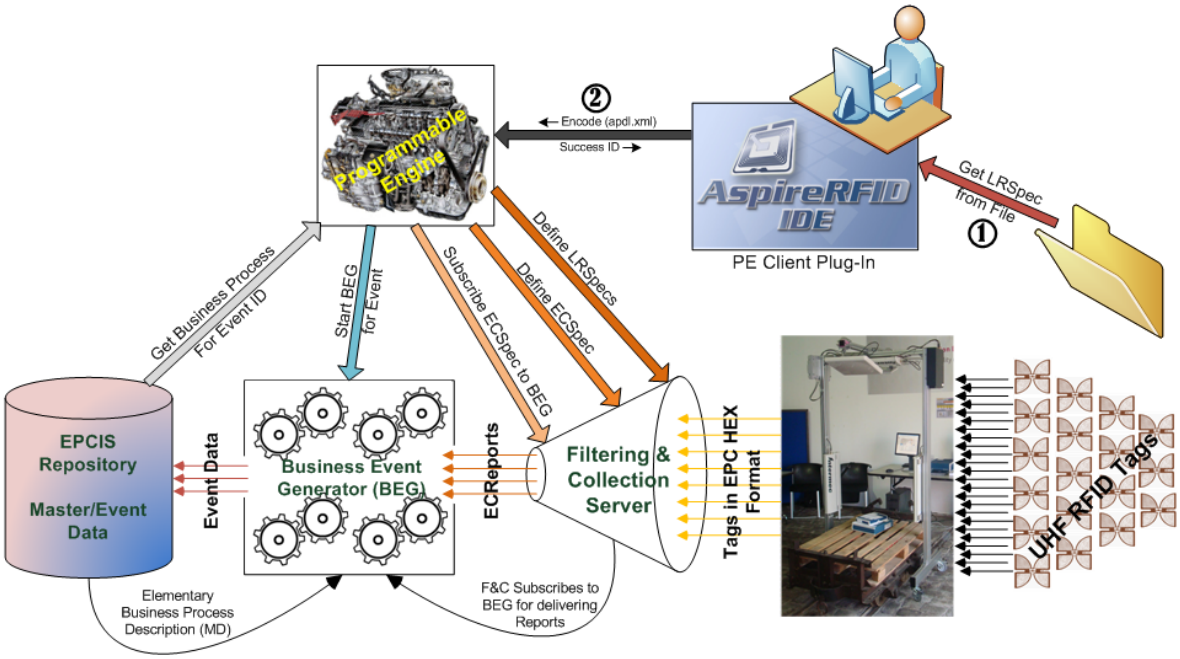

Figure 6 Required AspireRFID Configuring Steps with Programmable Engine

Summing up from the above we easily observe the differences in complexity and steps required for the two different Configuration methods which are:

- For the conventional way: **7 x "N" Steps**
  - Where "N" the EBProc's required to describe the a hole Supply Chain scenario.
- And for the PE's way only **2 Steps** are required independently of how complex the scenario is.

Section 9 PE's Examples

9.1 Encoding Example

In this Section we will use the "Receiving" Example provided at deliverable D4.3b (Programmable Filters – FML Specification) and at D4.4a (Programmable RFID Solutions Specification). At the D4.3b's example we described how the different modules should be configured separately, with the help of the different specification files required, to serve the "Receiving" process of a specific warehouse. At the D4.4a's example we describe how an APDL (AspireRFID Process Description Language) specification file should be defined for a "Receiving" EBProc so as to be able to configure the whole AspireRFID middleware to serve a warehouse receiving process. So in this example we will analyze how an APDL XML file is used from the PE to configure it. So let's start by the problem description.

9.1.1 Describing the Problem

A Company Named "ACME" which is a Personal Computer Assembler collaborates with a Microchip Manufacturer that provides it with the required CPUs. ACME at regular basis places orders to the Microchip Manufacturer for specific CPUs. ACME owns a Central building with three Warehouses. The first warehouse named Warehouse1 has 2 Sections named Section1 and Section2. Section1 has an entrance point where the delivered goods arrive.

ACME needs a way to automatically receive goods at Warehouse1 Section1 and inform its WMS for the new product availability and the correct completeness of each transaction.

9.1.2 Solution Requirements

An RFID Portal should be placed to ACME's Warehouse1 Section1 entrance point which will be called ReadPoint1. The RFID portal will be equipped with one Reader WarehouseRfidReader1. The received goods should get equipped with preprogrammed RFID tags from their "Manufacturer". The received goods should be accompanied with a preprogrammed RFID enabled delivery document. And finally the APDL XML file that was build at the Deliverable D4.4a (Section 8.4) [24] should be used to configure an AspireRFID middleware (Figure 2 above) instance which for your convenience an updated version is available at Appendix II.

9.1.3 Encoding the APDL Document

As shown in Figure 6 and described in Section 8.2 above two steps are required from the Users side to configure the AspireRFID middleware. In this section we will give an example on how the APDL XML file, which was build in D4.4a, is used from the Programmable Engine to configure the AspireRFID middleware.

At the Programmable Engine's side as shown in Figure 4 above after getting the command (Step 1, 2) from the user to encode this specific APDL file the next step is to analyze the received file and distinct the different CLCBProc's and their EBProc's which in this case we have only one from each. The ID for the CLCBProc is "urn:epcglobal:fmcg:bti:acmesupplying" and the ID of the EBProc is "urn:epcglobal:fmcg:bte: acmewarehouse1receive" as shown in Table 5 below.

```
<apdl:OLCBProc id="urn:ow2:aspirerfid:aprod:firstopenloopdescribedprocess"
            name="AcmeSupplyChainManagement">
>
      <!-- AspireRFID Process Description (Language Specification) -->

      <apdl:CLCBProc id="urn:epcglobal:fmcg:bti:acmesupplying"
            name="CompositeBusinessProcessName">
            <!-- RFID Composite Business Process Specification (the ID will be the
            Described Transactions's URI)-->
            <Description>Acme Supply Chain</Description>

            <apdl:EBProc Id="CLCBProcEnd" Name="CLCBProcEnd">
            </apdl:EBProc>

            <apdl:EBProc Id="CLCBProcStart" Name="CLCBProcStart">
            </apdl:EBProc>

            <apdl:EBProc id="urn:epcglobal:fmcg:bte:acmewarehouse1receive"
                  name="AcmeWarehouse3Ship">
            </apdl:EBProc>

            <Transitions>
            </Transitions>
      </apdl:CLCBProc>

</apdl:OLCBProc>
```

Table 5 CLCBProc Object [Encoding APDL Example]

Now for each EBProc and by taking in consideration their parent Objects (Attributes and IDs) the required specification files are build to configure this AspireRFID middleware running instance. So the Programmable Engine extracts one by one all the required specification files from the EBProc to fill an Object which we conveniently call "ProcessedEBProc" and is described in Table 3 above which ultimately will be used to configure the AspireRFID running instance.

```
            <apdl:EBProc id="urn:epcglobal:fmcg:bte:acmewarehouse1receive"
                  name="AcmeWarehouse3Ship">
                  <!-- Elementary RFID Business Process Specification (the ID will be the
                  Described Event's URI)-->
                  <xpdl:Description>Acme Warehouse 3 Receiving ReadPoint5 Gate3
                  </xpdl:Description>
                  <xpdl:TransitionRestrictions>
                        <xpdl:TransitionRestriction>
                              <xpdl:Join Type="AND"/>
                        </xpdl:TransitionRestriction>
```

```
            </xpdl:TransitionRestrictions>
                  <xpdl:ExtendedAttribute Name="ECSpecSubscriptionURI"
                        Value="http://localhost:9999" />
                  <xpdl:ExtendedAttribute Name="AleClientEndPoint"
                     Value="http://localhost:8080/aspireRfidALE/services/ALEService" />
                     <xpdl:ExtendedAttribute Name="AleLrClientEndPoint"
                     Value="http://localhost:8080/aspireRfidALE/services/ALELRService" />
                     <xpdl:ExtendedAttribute Name="EpcisClientCaptureEndPoint"
                     Value="http://localhost:8080/aspireRfidEpcisRepository/capture" />
                     <xpdl:ExtendedAttribute Name="EpcisClientQueryEndPoint"
                     Value=" http://localhost:8080/aspireRfidEpcisRepository/query " />
            </ExtendedAttributes>
            <apdl:DataFields>
            </apdl:DataFields>
      </apdl:EBProc>
```

Table 6 AcmeWarehouse3Ship EBProc

From the part of the EBProc shown in Table 6 above and more specifically the "*ExtendedAttributes*" the PE extracts the following information for the "ProcessedEBProc" object:

- **Id**: urn:epcglobal:fmcg:bte:acmewarehouse1receive
- **Name**: AcmeWarehouse3Ship
- **ecSpecSubscriptionURI**: http://localhost:9999
- **aleClientEndPoint**: http://localhost:8080/aspireRfidALE/services/ALEService
- **aleLrClientEndPoint**: http://localhost:8080/aspireRfidALE/services/ALELRService
- **epcisClientCaptureEndPoint**: http://localhost:8080/aspireRfidEpcisRepository/capture
- **epcisClientQueryEndPoint**: http://localhost:8080/aspireRfidEpcisRepository/query

Which in later steps are used from the PE to configure the AspireRFID running instance.

ALE-LR Setup

For the APDL document we are working on only one Logical Reader is defined which appears in Table 7 below.

```
<apdl:DataField
      type="LRSpec" name=" SmartLabImpinjSpeedwayLogicalReader">
            <LRSPec>
                  <isComposite>false</isComposite>
                  <readers/>
                  <properties>
                        <property>
                              <name>Description</name>
                              <value>
                              This Logical Reader consists of read point 1,2,3
                              </value>
                        </property>
                        <property>
                              <name>ConnectionPointAddress</name>
```

```
                        <value>192.168.212.238</value>
                    </property>
                    <property>
                        <name>ConnectionPointPort</name>
                        <value>5084</value>
                    </property>
                    <property>
                        <name>ReadTimeInterval</name>
                        <value>1000</value>
                    </property>
                    <property>
                        <name>PhysicalReaderSource</name>
                        <value>1,2,3</value>
                    </property>
                    <property>
                        <name>RoSpecID</name>
                        <value>1</value>
                    </property>
                    <property>
                        <name>ReaderType</name>
                        <value>
                    org.ow2.aspirerfid.ale.server.readers.llrp.LLRPAdaptor
                        </value>
                    </property>
                </properties>
            </LRSPec>
        </apdl:DataField>
```

Table 7 LRSpec DataField

As soon as the PE retrieve's the given LRSpec it store it to the **lrSpecs** attribute of the ProcessedEBProc Object in "SmartLabImpinjSpeedwayLogicalReader"/ "LRSpec Dynamic specification Object" pair manner. So the third Step as shown in Figure 4 above is to get all the Already Defined LRSpec names from the Running Instance of the AspireRFID middleware with the getLogicalReaderNames() ALE-LR command and if the "SmartLabImpinjSpeedwayLogicalReader" is not included in the returned list an ALE-LR define("SmartLabImpinjSpeedwayLogicalReader", LRSpec) is executed. If it is included then an ALE-LR update("SmartLabImpinjSpeedwayLogicalReader", LRSpec) is executed (Step 4).

ALE Setup

The ECSpec required for the AspireRFID configuration is given from the ECSpec "DataField" type. The only change to it done before storing it to the ecSpec Attribute (for later use) of the ProcessedEBProc Object is to concatenate to Every ECReport name which in our case is the "bizTransactionIDs" and the "transactionItems" (as we want BEG to produce Object Events), with the "@" symbol between them, the EBProc's ID which is "urn:epcglobal:fmcg:bte:acmewarehouse1receive" for later use of the BEG.

So the ECSpec's ECReport names will become:

- bizTransactionIDs@urn:epcglobal:fmcg:bte:acmewarehouse1receive
- transactionItems@urn:epcglobal:fmcg:bte:acmewarehouse1receive

```
<apdl:DataField
      type="ECSpec" name="RecievingECSpec">
            <ECSpec includeSpecInReports="false">
                  <logicalReaders>
                        <logicalReader>
                              SmartLabImpinjSpeedwayLogicalReader
                        </logicalReader>
                  </logicalReaders>
                  <boundarySpec>
                        <repeatPeriod unit="MS">4500</repeatPeriod>
                        <duration unit="MS">4500</duration>
                        <stableSetInterval unit="MS">0
                        </stableSetInterval>
                  </boundarySpec>
                  <reportSpecs>
                        <reportSpec reportOnlyOnChange="false"
                              reportName="bizTransactionIDs"
                              reportIfEmpty="true">
                            <reportSet set="CURRENT"/>
                              <filterSpec>
                              <includePatterns>
                                    <includePattern>
                                        urn:epc:pat:gid-96:145.12.*
                                    </includePattern>
                              </includePatterns>
                              <excludePatterns/>
                            </filterSpec>
                            <groupSpec/>
                            <output includeTag="true" includeRawHex="true"
                                  includeRawDecimal="true" includeEPC="true"
                                  includeCount="true"/>
                        </reportSpec>
                        <reportSpec reportOnlyOnChange="false"
                            reportName="transactionItems"
                                  reportIfEmpty="true">
                            <reportSet set="ADDITIONS"/>
                            <filterSpec>
                                  <includePatterns>
                                        <includePattern>
                                            urn:epc:pat:gid-96:145.233.*
                                        </includePattern>
                                        <includePattern>
                                            urn:epc:pat:gid-96:145.255.*
                                        </includePattern>
                                        </includePatterns>
                                        <excludePatterns/>
                            </filterSpec>
                            <groupSpec/>
                            <output includeTag="true" includeRawHex="true"
                                  includeRawDecimal="true" includeEPC="true"
                                  includeCount="true"/>
                        </reportSpec>
                  </reportSpecs>
                  <extension/>
            </ECSpec>
      </apdl:DataField>
```

Table 8 ECSpec DataField

After that for Step 5 (Figure 4) the PE implementation gets all the defined ECSpec names, which have been prior defined from the AspireRFID running instance with the ALE's getECSpecNames() command. If the "RecievingECSpec", which is the ECSpec name of our EBProc, is returned then the PE will execute an ALE undefine("RecievingECSpec") command and a define("RecievingECSpec", ECSpec) ALE command, one following the other, so as to achieve the Update of the ECSpec. If the "RecievingECSpec" is not returned then the PE will execute an ALE define("RecievingECSpec", ECSpec) command only.

EPC Information Service Setup

The next thing that ASPIRE's PE takes care of is, the retrieval from the APDL and configuration to the AspireRFID of the EBPRoc's Master Data. For the EBProc the Disposition, Transaction Type, Read Point and Business Step are retrieved and saved one by one, if they do not priory exist, from the given "EPCISMasterDataDocument" [8] shown in **Error! Reference source not found. Error! Reference source not found.** to the EPCIS repository thru the ASPIRE's EPCIS MasterData capture Interface.
So in our case we have for:
Disposition: *urn:epcglobal:fmcg:disp:in_progress*
Transaction Type: *urn:epcglobal:fmcg:btt:receiving*
Read Point: *urn:epcglobal:fmcg:loc:rp:warehouse1docdoor*
And Business Step: *urn:epcglobal:fmcg:bizstep:receiving*

```
<apdl:DataField
      type="EPCISMasterDataDocument" name="RecievingMasterData">
      <epcismd:EPCISMasterDataDocument>
            <EPCISBody>
                  <VocabularyList>
                        <Vocabulary
                              type="urn:epcglobal:epcis:vtype:BusinessTransaction">
                              <VocabularyElementList>
                                    <VocabularyElement
                              id=" urn:epcglobal:fmcg:bte:acmewarehouse1receive">
                                          <attribute
                  id="urn:epcglobal:epcis:mda:event_name"> Warehouse1DocDoorReceive
                                          </attribute>
                                          <attribute
                  id="urn:epcglobal:epcis:mda:event_type"> ObjectEvent
                                          </attribute>
                                          <attribute
                  id="urn:epcglobal:epcis:mda:business_step">
                                          urn:epcglobal:fmcg:bizstep:receiving
                                          </attribute>
                                          <attribute
                  id="urn:epcglobal:epcis:mda:business_location">
                                          urn:epcglobal:fmcg:loc:acme:warehouse1
                                          </attribute>
                                          <attribute
                  id="urn:epcglobal:epcis:mda:disposition">
                                          urn:epcglobal:fmcg:disp:in_progress
                                    </attribute>
                                          <attribute
                  id="urn:epcglobal:epcis:mda:read_point">
```

```
                                        urn:epcglobal:fmcg:loc:rp:warehouse1docdoor
                                              </attribute>
                                              <attribute
                        id="urn:epcglobal:epcis:mda:transaction_type">
                                                    urn:epcglobal:fmcg:btt:receiving
                                              </attribute>
                                              <attribute
                        id="urn:epcglobal:epcis:mda:action">OBSERVE
                                              </attribute>
                                        </VocabularyElement>
                                  </VocabularyElementList>
                            </Vocabulary>
                        </VocabularyList>
                  </EPCISBody>
            </epcismd:EPCISMasterDataDocument>
      </apdl:DataField>
```

Table 9 EPCISMasterDataDocument DataField

For the Business Transaction EPCIS vocabulary type the hole OLCBProc Structure is considered and a given EBProc is saved as the Child of it's CLCBProc and in its turn as a child of it's OLCBProc. For the last task the various different Object ID's are used to build the aforementioned structure which concludes the Step 7 of the PE's configuration process. So the whole "EPCISMasterDataDocument" given in the EBProc description will be saved at the EPCIS repository which will be a clild of the it's OLCBProc ("urn:ow2:aspirerfid:aprod:firstopenloopdescribedprocess") and it's CLCBProc ("urn:epcglobal:fmcg:bti:acmesupplying") Table 5 above.

BEG Setup

Continuing, the Programmable Engine uses the EPCIS Query End-Point, which was retrieved from the EBProc's "ExtededAttributes" above ("http://localhost:8080/aspireRfidEpcisRepository/query"), and use it to get the VocabularyElementType [8] (Step 8) for the "urn:epcglobal:fmcg:bte:acmewarehouse1receive" Elementary Business Processes (EBProc) ID which "conveniently happens" to be the same as the BusinessTransaction's ID that BEG is going to be configured to serve. After that the PE uses the EPCIS Client Capture End-Point ("http://localhost:8080/aspireRfidEpcisRepository/capture") and use the "startBegForEvent" BEG client Service, with inputs the VocabularyElementType that was prior retrieved, the "repositoryCaptureURL" and the "begListeningPort" as shown in Table 4 above (Step 9).

After the aforementioned BEG configuration it is ready to receive ECReports so the next Step (Step 10) is to Subscribe the "Defined" ECSpec, with the ALE subscribe("RecievingECSpec", "http://localhost:9999") from the previous step (Step6), to the BEG Running instance ("ecSpecSubscriptionURI" Table 3).

At this point it worth's to mention that AspireRFID architecture uses Fosstrak's [1] EPCIS and F&C (ALE) implementations that ASPIRE has enhanced and tailored to meet its needs.

9.2 Decoding Example

(To be implemented and documented in Version 2 of this deliverable)

Section 10 Business Process Workflow Management Editor (BPWME) Introduction

ASPIRE Architecture introduces a tool, called Business Process Workflow Management Editor (BPWME) plug-in, which will be part of the AspireRFID IDE that will be capable of producing APDL files and ultimately configure the AspireRFID middleware with the help of the Programmable Engine's Client.

The BPWME will be based on the Eclipse Rich Client Platform (RCP) design, which is what it is used for the AspireRFID IDE design, and more specifically at the:

- Eclipse Graphical Modeling Framework (GMF), described in Section 10.1 below which combines the:
  - Eclipse Modeling Framework (EMF)
  - And the Eclipse Graphical Editing Framework (GEF)

A first flavor of this tool is provided in Figure 7 below where someone can distinguish the main Design tab, the Diagrams outline, the Properties and the Toolbox. At the Design tab a pallet is provided, with APDL's main components, which a User can drag and drop inside the design area. As soon as the User clicks on a component inside the design area its properties appears at the Property tab where they can be changed. If the design gets too big the user can be navigated from the Outline tab where he can choose the area that appears in at the Design tab.

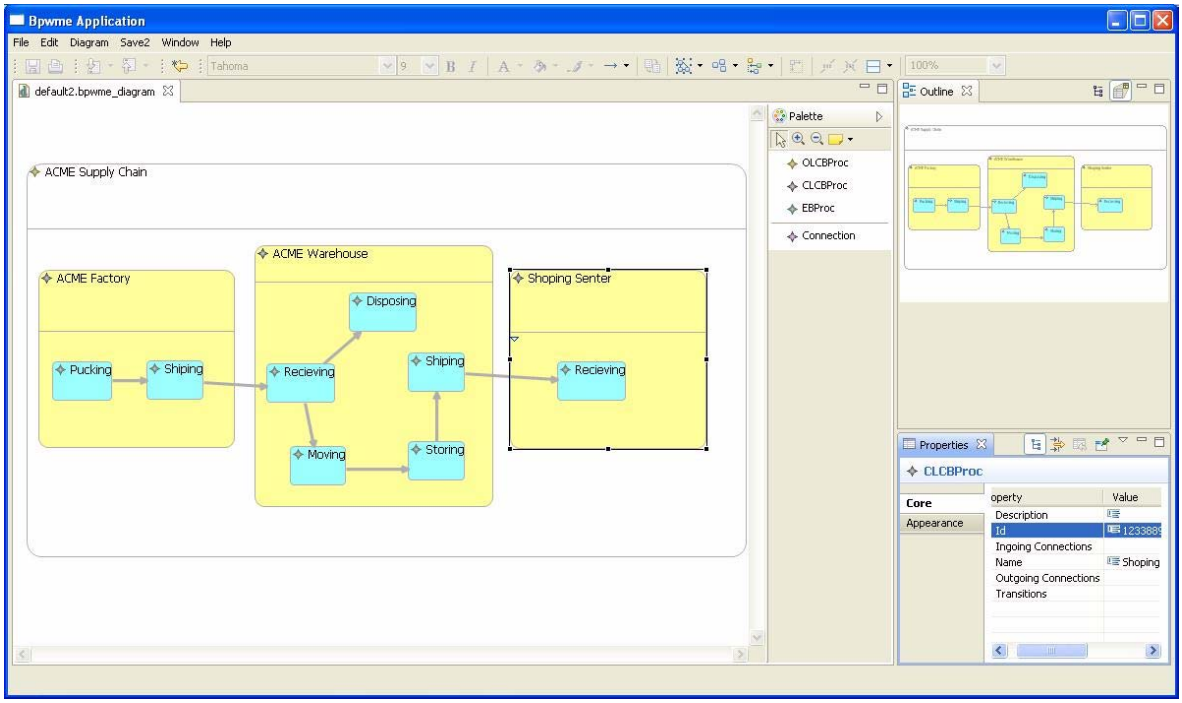

Figure 7 BPWME plug-in

The Programmable Engine's client will be embedded in this tool so as to achieve the direct Encoding of the AspireRFID middleware as soon as an APDL xml file is created. The ability of real time interaction of the BPWME plug-in thru the PE's "encode" and "decode" Interface will be investigated. Moreover the possibility of basing the End-to-End management interface on the BPWME will be investigated as the design and the abilities of the tool matures in time.

The task of Designing and Implementing the BPWME plug-in, as it does not have a deliverable dedicated to it, it will be reported to the Deliverables D4.4b, D3.5, D4.2b and D4.5.

10.1 Graphical Modeling Framework

GMF (Graphical Modeling Framework) is a framework for creating a generic graphical interface in eclipse by combining EMF (Eclipse Modeling Framework) and GEF (Graphical Editing Framework) technology together. The output of a GMF project can be an RCP application or an Eclipse plug-in. The Figure 8 below shows the main components and models used during GMF-based development.

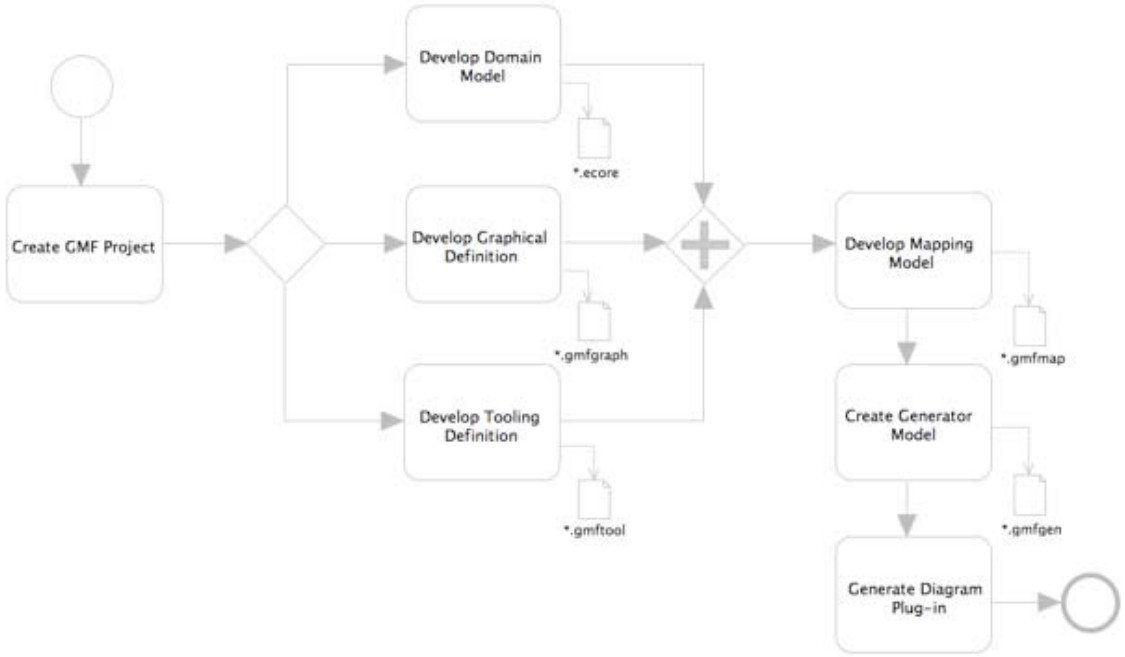

Figure 8 Main components and models used during GMF-based development [25]

To create a GMF project successfully, first we should define the domain model. For the Aspire project, the domain model is given by the APDL Specification [24]. Since we only need to care about the objects we are going to present in the editor, we simplify the APDL Specification as the following model also shown in Figure 9 below. In this model we create three new abstract objects for the editor:

- the WorkflowMap,
- the Node,
- and the Connection.

The WorkflowMap is the root of the whole editor canvas, which includes other nodes and edges. The Node is an abstraction of all the nodes on the editor. The Connection is a direct edge between two Nodes, which creates the relationship between the Nodes.

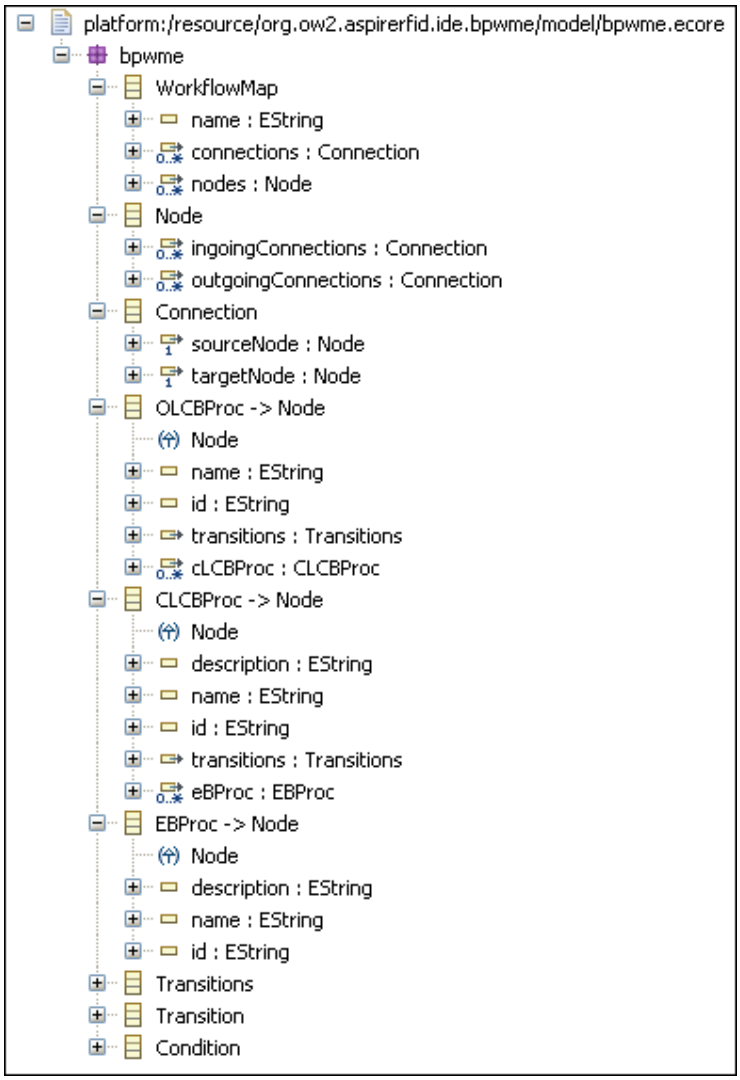

Figure 9 APDL's GMF abstract objects

Guided by the project dashboard, we can then define the tool palette, the figures we want to show in the editor, and the mapping between the domain model and the figures. Then we do the code generation. During each step, there are several choices we can make to adjust the configuration of the project. At last, we may modify the code itself to reflect exactly our own needs.

We mainly have three jobs when modifying the code.

1. Introduce APDL Specification file to the system. Let it work with the existing model file and map file consistently.
2. Introduce other editing policies for editing EBProc process.
3. And finally let the editor work with Aspire RFID IDE seamlessly.

Section 11 Current Implementation Status / Future Steps

11.1 Current Implementation Status

This deliverable describes the Specification and Implementation of the AspireRFID Programmable Engine which is going to be provided in two versions. In the first version the Programmable Engine provides:

- The "Encode" API design which is documented in this deliverable and its implementation which is uploaded to ASPIRE's SVN repository (http://forge.ow2.org/plugins/scmsvn/index.php?group_id=324)
- The "Decode" API design which is documented in this deliverable.
- For configuring the Programmable Engine the time this deliverable is written a Fat Client is used based on Java SWT.
- And a first approach of designing and implementing the Business Process Workflow Management Editor (BPWME) plug-in which is briefly described in Section 10 above.

11.2 Future Steps

In the second version of this deliverable the "Decode" API implementation description will be provided and the actual implementation will be uploaded to the ASPIRE's SVN repository (http://forge.ow2.org/plugins/scmsvn/index.php?group_id=324). The Configuration Client will be embedded to the AspireRFID IDE and will also be part of the BPWME. The "Encode" implementation will also be updated based on the evolution of the PE and the ASPIRE needs. Moreover as far as the BPWME plug-in is concerned a more throw description of the design and implementation will be provided and the source code will also be uploaded to the ASPIRE SVN repository. Also the ability of real time interaction of the BPWME plug-in thru the PE's "encode" and "decode" Interface will be investigated and the possibility of basing the End-to-End management interface on the BPWME plug-in will be investigated also as the design and the abilities of the tool matures in time.

Section 12 Conclusions

The deliverable has outlined the specifications of the ASPIRE programmable engine and demonstrated its features aiming at easing the configuration of ASPIRE solutions. In a nutshell the programmable engine offers users to deploy RFID solutions through working on high level business logic rather than writing significant amounts of low-level programming statements. The deployment is facilitated by the use of a defined business scenario using a business process language (APDL) where the programmable engine encodes them and converts them into a language understandable by the middleware.

The deliverable also outlines both the interfaces; 'Encode' and 'Decode' of the Aspire Programmable Engine. Furthermore, details of how the PE communicates with other ASPIRE RFID modules are also portrayed. While the implementation of the 'Encode' API has been implemented and documented along this deliverable. The implementation and documentation of the 'Decode' API will be conducted and documented in the final version of the deliverable due M36. A short comparison of how the PE eases and speeds up the development of RFID solution is also presented. In conjunction with this a small example of the programmable engine is also presented towards the end of the report.

This deliverable is an interim version and will be augmented in its next version D4.2b due M36.

Section 13 List of Figures

Section 14 List of Tables

Section 15 List of Acronyms

ALE	Application Level Event
APDL	AspireRFID Process Description Language
API	Application Program Interface
ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generator
BPWME	Business Process Workflow Management Editor
BTB	Bluetooth Bridge
CC	Connector Client
CE	Connector Engine
CLCBProc	Close Loop Composite Business Process
DNS	Directory Name Service
EBProc	Elementary Business Process
EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Services
ERP	Enterprise Resource Planning
F&C	Filtering and Collection
GIAI	Global Individual Asset Identifier
GLN	Global Location Number
GPS	Global Positioning System
GRAI	Global Returnable Asset Identifier
GS1	Global Standard 1 (Standardisation group)
GTIN	Global Trade Identification Number
HAL	Hardware Abstraction Layer
HTTP	HiperText Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
IS	Information System or Information Service
ISO	International Standard Organization
IT	Information Technology
J2ME	Java 2 Micro Edition
JAS	Java Application Server
JAXB	Java Architecture for XML Binding
JaxWS	Java web server
JCA	Java Connector Architecture
JCP	Java Community Process
JMS	Java Messaging Service
JMX	Java Management Extensions
JVM	Java Virtual Machine
LGPL	Lesser General Public License
LLRP	Low Level Reader Protocol
ODBC	Object Database Connectivity
OLCBProc	Open Loop Composite Business Process
ONS	Object Name Service
OSI	Open System Interconnection

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

Section 16 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 Mass, 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] WS-I, Basic Profile v1.0, available at: http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html.

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

[21] John Soldatos, Nikos Kefalakis, Nektarios Leontiadis, et. al., "Core ASPIRE Middleware Infrastructure", ASPIRE Project Public Deliverable D3.4a, Jun 2009, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables

[22] Efstathios Mertikas, Nikos Kefalakis and John Soldatos, "Managing Master Data and Business Events in an RFID Network", Submitted to the Pervasive and Mobile Computing Journal (Elsevier), September 2009

[23] John Soldatos, Nikos Kefalakis, Nektarios Leontiadis, et. al., "Programmable Filters – FML Specification", ASPIRE Project Public Deliverable D4.3b, Dec 2009, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables

[24] Nikos Kefalakis, John Soldatos, Sofoklis Efremidis, et. al., "Programmable RFID Solutions Specification", ASPIRE Project Public Deliverable D4.4a, Sept 2009, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables

[25] "Eclipse Graphical Modeling Framework Tutorial", available at: http://wiki.eclipse.org/GMF_Tutorial

[26] "CXF Servlet Transport", available at: http://cxf.apache.org/docs/servlet-transport.html

[27] Workflow Management Coalition Workflow Standard, "Workflow Process Definition Interface -- XML Process Definition Language V1.0", Document Number WFMC-TC-1025, October 25, 2002

Appendix I APDL XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" targetNamespace="urn:ow2:aspirerfid:apdlspec:xsd:1"
        xmlns:ale="urn:epcglobal:ale:xsd:1" xmlns:alelr="urn:epcglobal:alelr:xsd:1"
        xmlns:apdl="urn:ow2:aspirerfid:apdlspec:xsd:1"
        xmlns:epcismd="urn:epcglobal:epcis-masterdata:xsd:1"
        xmlns:xpdl="http://www.wfmc.org/2002/XPDL1.0">

        <xs:import namespace="urn:epcglobal:alelr:xsd:1"
                schemaLocation="resources/epcglobal/EPCglobal-ale-1_1-alelr.xsd"></xs:import>
        <xs:import namespace="urn:epcglobal:ale:xsd:1"
                schemaLocation="resources/epcglobal/EPCglobal-ale-1_1-ale.xsd"></xs:import>
        <xs:import namespace="urn:epcglobal:epcis-masterdata:xsd:1"
                schemaLocation="resources/epcglobal/EPCglobal-epcis-masterdata-1_0.xsd"></xs:import>
        <xs:import namespace="http://www.wfmc.org/2002/XPDL1.0"
                schemaLocation="resources/XPDL.xsd"></xs:import>
        <xs:element name="OLCBProc" type="apdl:OLCBProc" />
        <xs:element name="CLCBProc" type="apdl:CLCBProc" />
        <xs:element name="EBProc" type="apdl:EBProc" />

        <xs:complexType name="OLCBProc">
                <xs:sequence>
                        <xs:element maxOccurs="unbounded" ref="apdl:CLCBProc" />
                        <xs:element ref="xpdl:Transitions" />
                </xs:sequence>
                <xs:attribute name="id" use="required" type="xs:anyURI" />
                <xs:attribute name="name" use="required"
                        type="xs:NCName" />
        </xs:complexType>

        <xs:complexType name="CLCBProc">
                <xs:sequence>
                        <xs:element ref="xpdl:Description" />
                        <xs:element maxOccurs="unbounded" ref="apdl:EBProc" />
                        <xs:element ref="xpdl:Transitions" />
                </xs:sequence>
                <xs:attribute name="id" use="required" type="xs:anyURI" />
                <xs:attribute name="name" use="required"
                        type="xs:NCName" />
        </xs:complexType>

        <xs:complexType name="EBProc">
                <xs:sequence>
                        <xs:element ref="xpdl:Description" />
                        <xs:element ref="xpdl:TransitionRestrictions" />
                        <xs:element ref="xpdl:ExtendedAttributes" />
                        <xs:element ref="apdl:DataFields" />
                </xs:sequence>
                <xs:attribute name="id" type="xs:anyURI" />
                <xs:attribute name="name" type="xs:NCName" />
        </xs:complexType>

        <xs:element name="DataFields">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element minOccurs="3" maxOccurs="unbounded"
                                        ref="apdl:DataField" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="DataField">
                <xs:complexType>
                        <xs:choice>
                                <xs:element ref="ale:ECSpec" />
                                <xs:element ref="epcismd:EPCISMasterDataDocument" />
```

```
                        <xs:element ref="alelr:LRSpec" />
                    </xs:choice>
                    <xs:attribute name="name" use="required"
                        type="xs:NCName" />
                    <xs:attribute name="type" use="required"
                        type="xs:NCName" />
            </xs:complexType>
        </xs:element>

</xs:schema>
```

Appendix II APDL files

Encoding APDL Example

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

<apdl:OLCBProc
        id="urn:ow2:aspirerfid:aprod:firstopenloopdescribedprocess"
        name="AcmeSupplyChainManagement" xmlns:ale="urn:epcglobal:ale:xsd:1"
        xmlns:alelr="urn:epcglobal:alelr:xsd:1" xmlns:apdl="urn:ow2:aspirerfid:apdlspec:xsd:1"
        xmlns:epcglobal="urn:epcglobal:xsd:1" xmlns:epcis="urn:epcglobal:epcis:xsd:1"
        xmlns:epcismd="urn:epcglobal:epcis-masterdata:xsd:1"
        xmlns:p="http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader"
        xmlns:xpdl="http://www.wfmc.org/2002/XPDL1.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="urn:ow2:aspirerfid:apdlspec:xsd:1 ../AspireSpesificationLanguage.xsd ">

        <apdl:CLCBProc id="urn:epcglobal:fmcg:bti:acmesupplying"
                name="CompositeBusinessProcessName">
                <xpdl:Description>Acme Supply Chain</xpdl:Description>

                <apdl:EBProc Id="CLCBProcEnd" Name="CLCBProcEnd">
                        <xpdl:Description />
                        <xpdl:TransitionRestrictions>
                                <xpdl:TransitionRestriction>
                                        <xpdl:Join Type="XOR" />
                                </xpdl:TransitionRestriction>
                        </xpdl:TransitionRestrictions>
                        <xpdl:ExtendedAttributes>
                                <xpdl:ExtendedAttribute Name="XOffset" Value="623" />
                                <xpdl:ExtendedAttribute Name="YOffset" Value="202" />
                        </xpdl:ExtendedAttributes>
                </apdl:EBProc>

                <apdl:EBProc Id="CLCBProcStart" Name="CLCBProcStart">
                        <xpdl:Description />
                        <xpdl:TransitionRestrictions>
                                <xpdl:TransitionRestriction>
                                        <xpdl:Join Type="AND" />
                                </xpdl:TransitionRestriction>
                        </xpdl:TransitionRestrictions>
                        <xpdl:ExtendedAttributes>
                                <xpdl:ExtendedAttribute Name="XOffset" Value="47" />
                                <xpdl:ExtendedAttribute Name="YOffset" Value="196" />
                        </xpdl:ExtendedAttributes>
                </apdl:EBProc>

                <apdl:EBProc
                        id="urn:epcglobal:fmcg:bte:acmewarehouse1receive" name="AcmeWarehouse3Ship">

                        <xpdl:Description>Acme Warehouse 3 Receiving
                                ReadPoint5 Gate3
                        </xpdl:Description>
                        <xpdl:TransitionRestrictions>
                                <xpdl:TransitionRestriction>
                                        <xpdl:Join Type="AND" />
                                </xpdl:TransitionRestriction>
                        </xpdl:TransitionRestrictions>
                        <xpdl:ExtendedAttributes>
                                <xpdl:ExtendedAttribute Name="XOffset"
                                        Value="204" />
                                <xpdl:ExtendedAttribute Name="YOffset"
                                        Value="204" />
                                <xpdl:ExtendedAttribute Name="CellHeight"
                                        Value="30" />
                                <xpdl:ExtendedAttribute Name="CellWidth"
                                        Value="313" />
                                <xpdl:ExtendedAttribute Name="ECSpecSubscriptionURI"
```

```
                        Value="http://localhost:9999" />
                <xpdl:ExtendedAttribute Name="AleClientEndPoint"
                Value="http://localhost:8080/aspireRfidALE/services/ALEService" />
                <xpdl:ExtendedAttribute Name="AleLrClientEndPoint"
                Value="http://localhost:8080/aspireRfidALE/services/ALELRService" />
                <xpdl:ExtendedAttribute Name="EpcisClientCaptureEndPoint"
                Value="http://localhost:8080/aspireRfidEpcisRepository/capture" />
                <xpdl:ExtendedAttribute Name="EpcisClientQueryEndPoint"
                Value="http://localhost:8080/aspireRfidEpcisRepository/query " />

        </xpdl:ExtendedAttributes>
        <apdl:DataFields>
                <apdl:DataField type="EPCISMasterDataDocument"
                        name="RecievingMasterData">
                        <epcismd:EPCISMasterDataDocument>
                                <EPCISBody>
                                        <VocabularyList>
                                                <Vocabulary

                                 type="urn:epcglobal:epcis:vtype:BusinessTransaction">
                                                        <VocabularyElementList>
                                                                <VocabularyElement
                                                   id="urn:epcglobal:fmcg:bte:acmewarehouse1receive">
                                                                        <attribute
                                              id="urn:epcglobal:epcis:mda:event_name">Warehouse1DocDoorReceive
                                                                        </attribute>

                                                     <attribute id="urn:epcglobal:epcis:mda:event_type">
                                                                                ObjectEvent
                                                                        </attribute>
                                                <attribute     id="urn:epcglobal:epcis:mda:business_step">
                                                           urn:epcglobal:fmcg:bizstep:receiving
                                                                        </attribute>
                                                <attribute id="urn:epcglobal:epcis:mda:business_location">
                                                             urn:epcglobal:fmcg:loc:acme:warehouse1
                                                                        </attribute>
                                                <attribute id="urn:epcglobal:epcis:mda:disposition">
                                                                urn:epcglobal:fmcg:disp:in_progress
                                                                        </attribute>
                                                <attribute id="urn:epcglobal:epcis:mda:read_point">
                                                        urn:epcglobal:fmcg:loc:rp:warehouse1docdoor
                                                                        </attribute>
                                                       <attributeid="urn:epcglobal:epcis:mda:transaction_type">
                                                                urn:epcglobal:fmcg:btt:receiving
                                                                        </attribute>
                                                        <attribute id="urn:epcglobal:epcis:mda:action">OBSERVE
                                                                        </attribute>
                                                                </VocabularyElement>
                                                        </VocabularyElementList>
                                                </Vocabulary>
                                        </VocabularyList>
                                </EPCISBody>
                        </epcismd:EPCISMasterDataDocument>
                </apdl:DataField>

                <apdl:DataField type="ECSpec" name="RecievingECSpec">
                        <ale:ECSpec includeSpecInReports="false">
                                <logicalReaders>
                                <logicalReader>SmartLabImpinjSpeedwayLogicalReader
                                        </logicalReader>
                                </logicalReaders>
                                <boundarySpec>
                                        <repeatPeriod unit="MS">4500
                                        </repeatPeriod>
                                        <duration unit="MS">4500</duration>
                                        <stableSetInterval unit="MS">0
                                        </stableSetInterval>
                                </boundarySpec>
                                <reportSpecs>

                                        <reportSpec reportOnlyOnChange="false"
                        reportName="bizTransactionIDs" reportIfEmpty="true">
                                                <reportSet set="CURRENT" />
```

```
                                        <filterSpec>
                                                <includePatterns>
                                <includePattern>urn:epc:pat:gid-96:145.12.*
                                                        </includePattern>
                                                </includePatterns>
                                                <excludePatterns />
                                        </filterSpec>
                                        <groupSpec />
                                <output includeTag="true" includeRawHex="true"
                                        includeRawDecimal="true" includeEPC="true"
                                                includeCount="true" />
                                </reportSpec>
                                <!--
                        For the required ECReportID we will use the EBPSpec id
                                -->
                                <reportSpec reportOnlyOnChange="false"
                        reportName="transactionItems" reportIfEmpty="true">
                                        <reportSet set="ADDITIONS" />
                                        <filterSpec>
                                                <includePatterns>
                                <includePattern>urn:epc:pat:gid-96:145.233.*
                                                        </includePattern>
                                <includePattern>urn:epc:pat:gid-96:145.255.*
                                                        </includePattern>
                                                </includePatterns>
                                                <excludePatterns />
                                        </filterSpec>
                                        <groupSpec />
                                <output includeTag="true" includeRawHex="true"
                                        includeRawDecimal="true" includeEPC="true"
                                                includeCount="true" />
                                </reportSpec>
                        </reportSpecs>
                        <extension />
                </ale:ECSpec>
        </apdl:DataField>

        <apdl:DataField type="LRSpec"
                name="SmartLabImpinjSpeedwayLogicalReader">
                <alelr:LRSpec>
                        <isComposite>false</isComposite>
                        <readers />
                        <properties>
                                <property>
                                        <name>Description</name>
                                <value>This Logical Reader consists of read
                                                point 1,2,3
                                        </value>
                                </property>
                                <property>
                                        <name>ConnectionPointAddress
                                        </name>
                                        <value>192.168.212.238</value>
                                </property>
                                <property>
                                        <name>ConnectionPointPort</name>
                                        <value>5084</value>
                                </property>
                                <property>
                                        <name>ReadTimeInterval</name>
                                        <value>1000</value>
                                </property>
                                <property>
                                        <name>PhysicalReaderSource
                                        </name>
                                        <value>1,2,3</value>
                                </property>
                                <property>
                                        <name>RoSpecID</name>
                                        <value>1</value>
                                </property>
                                <property>
                                        <name>ReaderType</name>
```

```
                                                    <value>
                                              org.ow2.aspirerfid.ale.server.readers.llrp.LLRPAdaptor
                                                    </value>
                                                </property>
                                            </properties>
                                        </alelr:LRSpec>
                                    </apdl:DataField>
                            </apdl:DataFields>

                    </apdl:EBProc>

                    <xpdl:Transitions>
                            <xpdl:Transition Id="Start_Warehouse3RecievingGate3"
                                    Name="Start_Warehouse3RecievingGate3" From="CLCBProcStart"
                                    To="urn:epcglobal:fmcg:bte:acmewarehouse3ship" />
                            <xpdl:Transition Id="Warehouse3RecievingGate3_End"
                                    Name="Warehouse3RecievingGate3_End"
                              From="urn:epcglobal:fmcg:bte:acmewarehouse3ship"
                                    To="CLCBProcEnd" />
                    </xpdl:Transitions>
            </apdl:CLCBProc>

</apdl:OLCBProc>
```

Appendix III Soap Interfaces

PE Encode Soap Interface

```
@WebService(name="ProgrammableEngineEncoderInterface",
targetNamespace="http://encode.programmableengine.aspirerfid.ow2.org/")
public interface ProgrammableEngineEncoderInterface {

	@RequestWrapper (className="org.ow2.aspirerfid.programmableengine.model.OLCBProc")
	@ResponseWrapper(className="java.lang.Integer")
	public Integer encode(@WebParam(name="openLoopCBProc") OLCBProc openLoopCBProc);


}
```

PE Decode Soap Interface

```
@WebService(name="ProgrammableEngineDecoderInterface",
targetNamespace="http://decode.programmableengine.aspirerfid.ow2.org/")
public interface ProgrammableEngineDecoderInterface {

	@RequestWrapper (className="java.lang.String")
	@ResponseWrapper(className="org.ow2.aspirerfid.programmableengine.model.OLCBProc")
	public OLCBProc decode(@WebParam(name="openLoopCBProcID") String openLoopCBProcID);

}
```