ASPIRE FP7 215417

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) (Final Version)

Due date of deliverable:	M39
Actual Submission date:	03.05.11

Deliverable ID:	**WP4/D4.2b**
Deliverable Title:	ASPIRE Programmable Engine (APE) (Final Version)
Responsible partner:	AIT
Contributors:	Nikos Kefalakis - AIT John Soldatos - AIT Mathieu David - AAU Sofyan M. Yousuf – OSI Humberto Morán – OSI Didier Donsez - UJF Kiev Gama – UJF
Estimated Indicative Person Months:	36

Start Date of the Project:	1 January 2008	Duration:	42 Months

Revision:	1.3
Dissemination Level:	PU

Document Information

Document Name:	**ASPIRE Programmable Engine (APE) (Final Version)**
Document ID:	**WP4/D4.2b**
Revision:	**B1.3**
Revision Date:	**3 May 2011**
Author:	**AIT**
Security:	**PU**

Approvals

	Name	Organization	Date	Visa
Coordinator	Neeli Rashmi Prasad	CTIF-AAU	03.05.11	Approved
Technical Coordinator	John Soldatos	AIT	18.04.11	Approved
Quality Manager	Anne Bisgaard Pors	CTIF-AAU	20.04.11	Approved

Reviewers

Name	Organization	Date	Comments	Visa
Humberto Morán	OSI	Jan 2011	First review	Not ready for submission
Humberto Morán	OSI	April 2011	Final review	Ready for submission, with comments

Document history

Revision	Date	Modification	Authors
a_0.1	02 Nov 09	ToC	Nikos Kefalakis
a_0.2	10 Nov 09	ToC Finalization and Chapters Assign.	Nikos Kefalakis
a_0.3	27 Nov 09	Added Section 5, Section 6, Section 8	Nikos Kefalakis
a_0.4	09 Dec 09	Added Section 2	Mathieu David
a_0.5	10 Dec 09	Augmented Section 2	Sofyan M. Yousuf
a_0.6	11 Dec 09	Added Section 10.1	Yongming Luo

a_0.7	15 Dec 09	Added Section 9, Section 10.	Nikos Kefalakis
a_0.8	17 Dec 09	Added Section 7	Nikos Kefalakis
a_0.9	17 Dec 09	Added Section 1	John Soldatos
a_1.0	18 Dec 09	Added Section 5.3	Didier Donsez, Kiev Gama
a_1.1	18 Dec 09	Added Section 11, Reviewed the Doc.	Sofyan M. Yousuf
a_1.2	18 Dec 09	Added Section 3, 5.1, Replaced 5.3.1 and 5.3.2, Final Corrections.	Nikos Kefalakis
b_0.1	20 Jan 11	First Draft of D4.2b	Nikos Kefalakis
b_0.2	25 Jan 11	Review/Commented the Doc.	Humberto Morán
b_0.3	26 Jan 11	Augmented Introduction, conclusions, acronyms	Humberto Morán
b_0.4	10 Feb 11	Corrections following review Comments	Nikos Kefalakis
b_0.5	21 Feb 11	Enhanced Section 3	Nikos Kefalakis
b_0.6	01 Mar 11	Added Section 4, Enhanced/Updated Section 9.1	Nikos Kefalakis
b_0.7	07 Mar 11	Enhanced/Updated Section 5	Nikos Kefalakis
b_0.8	11 Mar 11	Added new Section 6	Nikos Kefalakis
b_0.9	31 Mar 11	Enhanced/Updated Section 7	Nikos Kefalakis
b_1.0	03 Apr 11	Updated Section 2, Corrections for final internal review	Nikos Kefalakis
b_1.1	04 Apr 11	Final review	Humberto Morán
b_1.2	05 Apr 11	Updated Section 11, Final corrections considering review comments	Nikos Kefalakis
B_1.3	03.05.11	Approved	Neeli Prasad

Content

Section 1 Executive Summary

One of the main objectives of ASPIRE is the research, specification and implementation of domain-specific languages (notably XML-based) enabling the specification of programmable / configurable RFID solutions, along with supporting run-time software enabling their implementation as part of the ASPIRE middleware infrastructure.

A background document, ASPIRE's D4.4 (submitted in its initial version in September 2009), focused on the specifications of APDL (AspireRFID Process Description Language); a domain specific language for describing/configuring RFID solutions. Based on such specifications, this deliverable is devoted to the description of a run-time middleware infrastructure able to translate and convert APDL into a number of configuration files, which are in turn used in 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 PE bridges APDL with the underlying ASPIRE middleware infrastructure and allows the "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 using 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). 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. There is also a direct relationship between the PE and the ASPIRE middleware architecture, resulting from the fact that the PE operates over the ASPIRE open source middleware infrastructure (the later provided in the scope of the AspireRFID OSS project – see: http://wiki.aspire.ow2.org/).

In this context, the present deliverable illustrates the interfaces of the PE to the middleware building blocks of the ASPIRE architecture.

The PE comprises six types of functionalities: (a) "Register" functionalities that enable the registering and subsequent mapping of an APDL compliant instance to the ASPIRE middleware configuration files constituting the RFID solution, (b) "Get" functionalities that enable the retrieval of an RFID solution to an APDL file for subsequent editing, (c) "Update" functionalities that enable the user to update a priory registered APDL file (d) "Stop" functionalities that enable the user to stop a specific business process, (e) "Start" functionalities that enable the user to start a priory stop Business process, and (f) "Unregister" functionalities that enable a user to stop and clean the running instance from a specific priory registered specification. This deliverable provides details on all six functionalities and their relevant APIs (Application Programming Interfaces).

ASPIRE also envisages the use of tools for editing APDL and the PE configuration parameters. This deliverable presents a proof of concept of such tools, with particular emphasis on the Business Process Workflow Management Editor (BPWME), which allows the graphical modelling of APDL-compliant RFID based business processes. For improved readability, this deliverable provides concrete examples and use cases associated with the operation of the PE.

To justify the approach, this document includes a preliminary evaluation of the suggested approach to the development and deployment of RFID through programmability. Such evaluation reveals that developing an RFID solution using a PE based approach results in a significant reduction of the number of steps required from inception to deployment. Specifically, the deliverable elaborates on 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. It also reveals some limitations of the general PE approach; particularly that it is not directly applicable to domains other than logistics and supply chain management.

This deliverable, which is the enhanced version of the earlier deliverable D4.2a, describes the functionalities offered by the PE and incorporates the latest developments of the APDL (detailed in ASPIRE's deliverable D4.4).

Section 2 Introduction

RFID has advanced significantly over the past few decades. Rapid developments in low cost microelectronics and radio frequency transceivers have considerably reduced size and costs of high-frequency and ultra-high frequency transceivers, allowing longer reading ranges and faster reading rates than before. Such developments enable novel applications with higher mobility and larger number of tagged items. In turn, these applications require a more robust and complex middleware platform offering adaptable functionality at different layers of the communication architecture, for example to operate in different business contexts. Such complex specifications generate several research challenges in middleware design, such challenges translating into a high entry cost for RFID technology adopters, which in turn hinder the ability of SME's to participate in the RFID revolution.

ASPIRE offers a radical change in the current RFID paradigm through innovative, programmable, royalty-free, lightweight and privacy friendly middleware. ASPIRE solutions are open source and royalty free, therefore bringing an important reduction of the Total Cost of Ownership. They are also programmable and lightweight, therefore bringing backwards compatibility with current IT SME infrastructure. Additionally, ASPIRE has been designed as privacy friendly which means that present and future RFID privacy features can be easily implemented using the platform. Finally, ASPIRE is a vehicle for realizing the proposed switch in the current RFID deployment paradigm. Portions (i.e. specific libraries) of the ASPIRE middleware are hosted and run on low-cost RFID-enabled microelectronic systems, and so further lower the TCO in mobility scenarios (i.e. mobile warehouses, trucks). In the long term, the ASPIRE middleware platform will be combined with other innovative European developments in the area of ubiquitous RFID-based sensing (e.g. ambient sensing [temperature, humidity, pressure, acceleration], mobile, low-cost); therefore enabling novel business cases source of significant productivity gains.

This new middleware paradigm will be particularly beneficial to European SMEs, currently experiencing significant cost barriers to RFID deployment. The ASPIRE open source nature aims to offer immense flexibility and maximum freedom to potential RFID developers and implementers. This versatility includes the freedom of choice of RFID hardware (notably tags and interrogators).

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

The programmability features of ASPIRE aim to maximise the configurability of ASPIRE provided solution. The ASPIRE programmability functionality offers RFID developers and consultants the possibility of deploying RFID solutions through high-level company meta-data (including the business context of its RFID deployments), rather than through low-level programming.

However, the achievement of programmability poses important technical and research challenges. Firstly, it needs to provide an intuitive abstraction for implementation options without losing generality. Secondly, it needs to be "reversible", therefore allowing the re-programmability of existing implementations. Finally, it needs to be "portable", separating the abstraction of the program from the implementation details.

This deliverable presents the specifications of the ASPIRE Programmable Engine (APE). This module is an interface between the user and the ASPIRE middleware that enables 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) converts to a language understandable by the ASPIRE middleware.

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

- **Register** which enables the registering and subsequent mapping of an APDL compliant instance to the ASPIRE middleware configuration files constituting the RFID solution,
- **Update** which enables the user to update a priory registered APDL file,
- **Get** functionalities enabling the retrieval of an RFID solution to an APDL file for subsequent editing,
- **Stop** that enables the user to stop a specific business process,
- **Start** which enables the user to start a priory stop Business process, and
- **Unregister** that enables a user to stop and clean the running instance from a specific priory registered specification.

This deliverable briefly presents the AspireRFID Process Description Language in Section 3, and the methodology for designing the PE at Section 4. The relation between the Programmable Engine and the other components of the ASPIRE middleware is presented in Section 5. The specifications of the Programmable Engine are described in Section 6 where the different PE's APIS are analyzed and in Section 7 the API implementation is presented. How the PE Changed the AspireRFID Configuration Process is presented in Section 8, and an example of the PE registering follows in Section 9. A brief introduction of the Business Process Management Workflow Editor (BPMWE) is presented in Section 10. Finally Section 11 presents the conclusions of this deliverable.

Section 3 AspireRFID Process Description Language (APDL)

The ASPIRE Programmable Meta-Language is a language created within ASPIRE with the intention to be able to fully describe an Open Loop Composite (RFID) Business Process (OLCBProc) and ultimately be used from the Programmable Engine to configure an AspireRFID middleware instance to serve the described Business Processes. Due to the fact that AspireRFID Process Description Language (APDL) [24] [36] is closely bound with the Programmable engine it is crucial to understand it. So in this section we will review the APDL towards describing its logic, main components and structure.

APDL is oriented towards solutions that comply with the EPCglobal Architecture. According to this architecture [17] the modules that compose an end-to-end RFID solution can be logically considered to be layered as depicted in Figure 1.

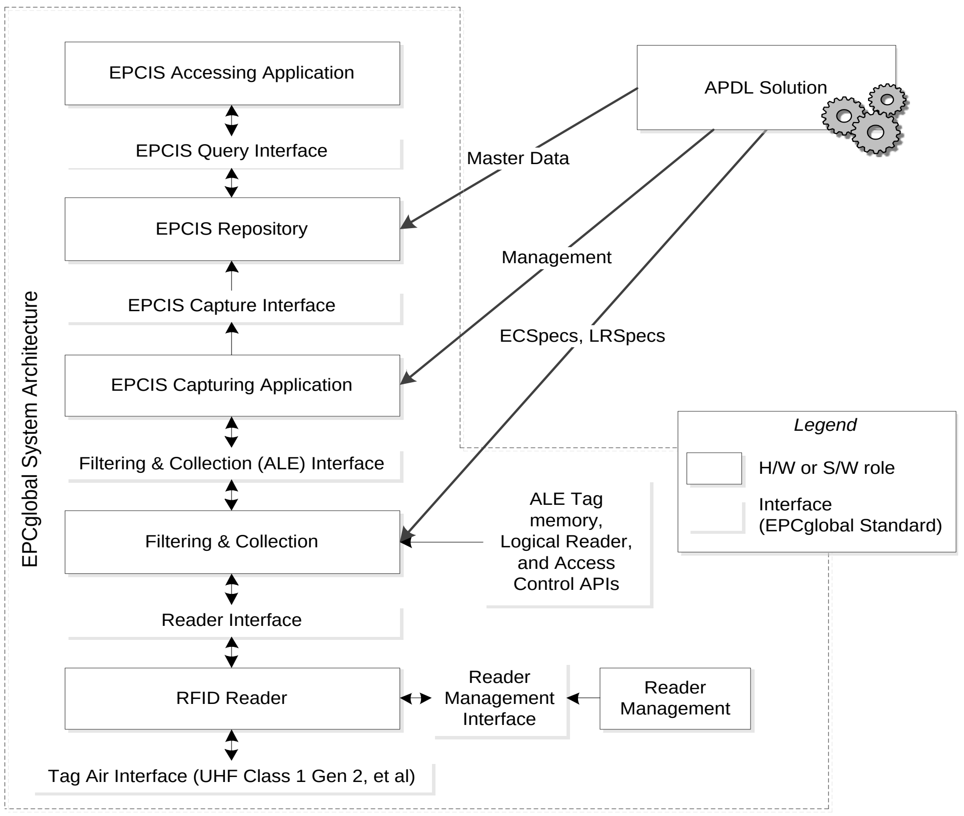

EPCglobal System Architecture
EPCIS Accessing Application
EPCIS Query Interface
EPCIS Repository
EPCIS Capture Interface
EPCIS Capturing Application
Filtering & Collection (ALE) Interface
Filtering & Collection
Reader Interface
RFID Reader
Tag Air Interface (UHF Class 1 Gen 2, et al)
APDL Solution
Master Data
Management
ECSpecs, LRSpecs
ALE Tag memory, Logical Reader, and Access Control APIs
Reader Management Interface
Reader Management
Legend
H/W or S/W role
Interface (EPCglobal Standard)

Figure 1: Middle Middleware configuration using APDL [17]

Hence, according to the EPCglobal architecture, a middleware solution requires the combination and orchestration of various specifications towards:

- Defining the Event Cycle Specifications (ECSpecs) [2],
- Defining the Logical Reader Specifications (LRSpecs) [2],
- And, finally, providing the EPCIS (Electronic Product Code Information Sharing) [8] with the required Master Data (EPCIS Master Data Document) that partially manages how Application Level Events (ALE) [2], will be stored in the EPCIS repository.

3.1 The Required Components/Layers

Each of the above-mentioned specifications is associated with a number of RFID middleware modules and data elements, which collectively comprise an RFID solution.

In particular, at the F&C (Filtering and Collection) [2] module one must configure the ECSpec, which is a complex type that describes an Event Cycle [2] and one or more reports to be produced from it. An ECSpec also includes the Logical Reader list which is going to be used for the denoted Event Cycle. The LRSpecs specification is accordingly used to describe Logical Readers configurations.

Layer Master Data Vocabularies are defined at the EPCIS [8]. The Master Data Vocabularies contain additional data that provides the necessary context for interpreting Event Data [8]. The most important Master Data vocabulary type for describing an RFID Business process is the BusinessTransactionTypeID which is capable of enclosing all the required information for identifying a particular business transaction. In Table 1 the BusinessTransactionID's attributes are shown.

Attribute Name	**Attribute URI**
EventName [2]	urn:epcglobal:epcis:mda:event_name
EventType [2]	urn:epcglobal:epcis:mda:event_type
BusinessStep [2]	urn:epcglobal:epcis:mda:business_step
BusinessLocation [2]	urn:epcglobal:epcis:mda:business_location
Disposition [2]	urn:epcglobal:epcis:mda:disposition
ReadPoint [2]	urn:epcglobal:epcis:mda:read_point
TransactionType [2]	urn:epcglobal:epcis:mda:transaction_type
Action [2]	urn:epcglobal:epcis:mda:action

Table 1: Business Transaction ID Attributes

In order to integrate the Information Sharing layer with the F&C layer (Figure 1), we introduce a capturing application called Business Event Generator (BEG) [22] [23]. BEG lies between the F&C and Information Service (e.g., EPC-IS) modules. The role of the BEG is to automate the mapping between reports stemming from F&C and IS events. The Business event generation module associates Master Data, stored at the EPCIS repository, with RFID tag data which are produced in the form of Event Cycle Reports (ECReports [2]) from the Filtering and collection module. Sources of data include filtered, collected EPC (Electronic Product Code) tag data obtained from various RFID physical sources. The RFID data are captured from BEG module and eventually are stored at the Information Service repository in the form of RFID Events (Object, Quantity, Aggregation, and Transaction Events) as defined in the EPC-IS specification [8].

The BEG module recognizes the occurrence of EPC-related business events, and delivers these as EPCIS data. BEG facilitates the abovementioned middleware modules and data elements to generate and store to the EPCIS repository context aware RFID Event Data. Low level business processes creation requirements are defined, as shown in Figure 2 and Figure 3, to give the ability to combine them together, in order to describe a complete business transaction (e.g. Receiving, Shipping, Pick & Pack, etc). These Low Level business processes that also contain all the above described specifications are characterized as Elementary Business Process (EBProc).

3.2 Defining APDL's Business Process Types

Figure 2 depicts an example of the concept of decomposing a business process into a number of RFID business events. We can see that a "Moving" Business Process could be analyzed in a number of RFID events that we call Elementary Business Processes.

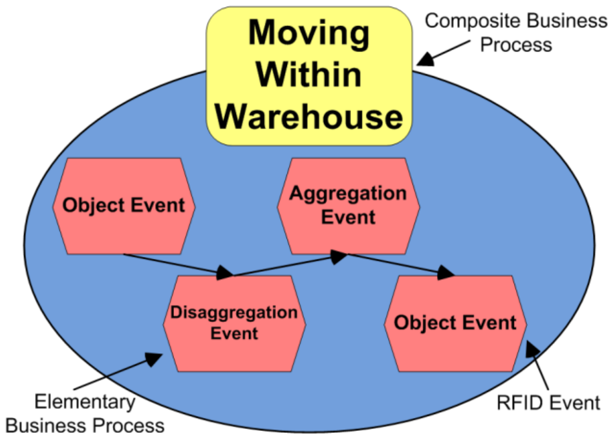


Figure 2: Decomposing an Inter-enterprise Business process

In the APDL Language we have used concepts/definitions, for mapping complex and basic Supply Chain Management (SCM) business processes, which are described below in the context of the following example. Figure 3 illustrates an example of a supply chain of consumer items (in this case bottles) all the way from the moment they are shipped from the factory, going through the warehouse premises, up to the shopping centre.

We call this entire process Open Loop Composite Business Process (OLCBProc). Open-Loop in the context of APDL stands for business processes that are executed throughout the lifecycle of a supply chain. For instance, an Open Loop procedure refers to a supply chain whose objects of interest move from any location in the factory till a retail store shelf regardless to whether these business locations belong to the same company or no.

An OLCBProc can be broken into many Close Loop Composite Business Processes (CLCBProc). A CLCBProc is related with the Business Location that a group of transactions takes place and the company that "owns" these transactions. So at Figure 3 example at the Factory's Business location we define one CLCBProc which contains all the company's transaction for the specific physical location.

A CLCBProc can further be divided into the finest business entity we define called Elementary Business Processes (EBProc). In the example in Figure 3, at the Factory's CLCBProc we define three EBProcs which are Commission of Bottles, Pack Bottles into Case and Shipment of the Case that can be described from an Object Event, an Aggregation Event, and an Object Event respectively. We define an Aggregation Event at packing the bottles because we need to bind the IDs of the bottles, which are the transacted items, with the ID of every case, which is the parent object. A similar decomposition and description of Business Processes from RFID Events is done at the CLCBProc of the Warehouse and the Shopping centre.

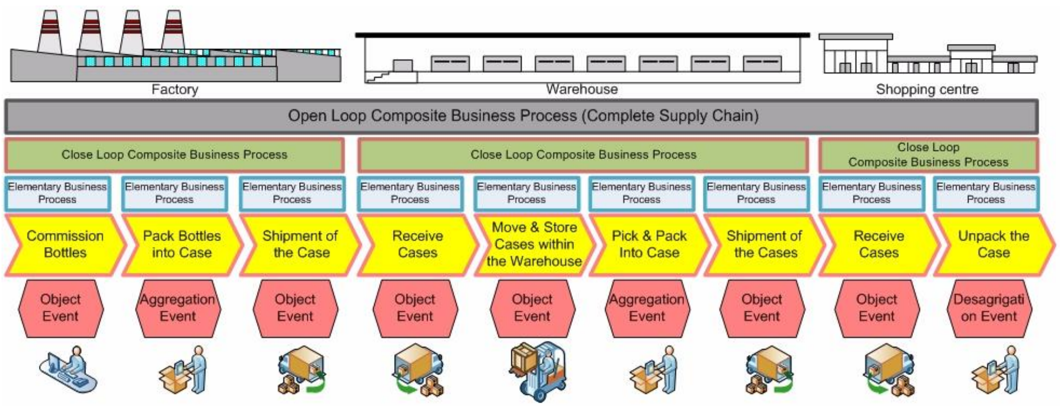


Figure 3: **Complete supply chain scenario example.**

3.3 Generating Business Logic

Summing up, fixed lists of identifiers with standardized meanings for concepts like business step and disposition along with user-created identifiers like read point, business location, business transaction and business transaction type at the EPCIS layer and ECSpecs at the ALE layer must be defined and combined with rules applied by the BEG layer so as RFID Events production can be successfully achieved. All these information elements will be stored and managed as pieces of Master Data within an appropriate database schema.

To create Event Data, some event fields are required and some are optional. Table 2 maps these associations.

R = Required O = Optional	**ObjectEvent**	**Aggregation Event**	**Quantity Event**	**Transaction Event**
Action	R	R	-	R
bizLocation	O	O	O	O
bizStep	O	O	O	O
bizTransactionList	O	O	O	R
childEPCs	-	R	-	-
Disposition	O	O	-	O
epcClass	-	-	R	-
epcList	R	-	-	R
eventTime	R	R	R	R
parented	-	R	-	O
Quantity	-	-	R	-
readPoint	O	O	O	O

Table 2: **Event fields with Event Types mapping (Master Data) [8][35]**

So by taking into consideration the Table 2 to generate an ObjectEvent the information that is required to be produced from the F&C layer is the "epcList" and the "eventTime". Optionally the "bizTransactionList" can also be produced to generate the Event. The rest of the information that is required or is optional is retrieved from the company's Master Data. For that reason we define two ECReports at any defined ECSpec for generating an Object Event, the one would contain the tag Classes that belong to the transaction's items, which is required, and the other that would contain the tag classes that would belong to the transaction ID (e.g. the receiving document's tag Class) which is optional.

Continuing for generating an AggregationEvent the "childEPCs", the "parented" tag List and the "eventTime", which is given by default by every ECReport, are required to be produced from the F&C layer and the "bizTransactionList" is optional. We define at the ECSpec two required ECReports. The one would contain the tag classes that belong to the transaction's Items, which will be used for the required "childEPCs" tag List, (e.g. the tagged items inside a carton box). The second one would contain the tag classes that belong to the Parent Objects, which will be used for the required "parented" EPC list, (e.g. the tagged carton box of the previous example). We define optionally one more ECReport that would contain the tag classes that would belong to the transaction ID (e.g. the order's tagged document that would require the specific Aggregation Event).

Following the same rationale and by taking into consideration the Table 2 we have defined all the required and optional ECReports [2] that need to be produced, which are defined at the ECSpec, from the F&C layer. These reports would then be captured from the BEG Layer and eventually generate the equivalent EPC RFID Events. This Event/Report binding is summarized in Table 3.

ECReport Names	**Object Event**	**Aggregation Event**	**Quantity Event**	**Transaction Event**
bizTransactionIDs	O	O	O	R
transactionItems	R	R	R	R
parentObjects	-	R	-	O
bizTransactionParentIDs	-	-	-	R

Table 3: ECReports name and Event Bindingbeing used at the ECSpec Definition

The ECReport groups shown in Table 3 are explained as follows:

- bizTransactionIDs: Include only the Transaction ID EPC Classes set up to be always reported, by making use of CURRENT at the ECReportSetSpec Section 8.2.6 of [2].
- transactionItems: Include only the Transaction's Items EPC Classes set up to be reported only once, by making use of ADDITIONS at the ECReportSetSpec Section 8.2.6 of [2].
- parentObjects: Include only the Transaction's Parent Objects EPC Classes for an Aggregation Event to be reported only once, by making use of ADDITIONS at the ECReportSetSpec.
- bizTransactionParentIDs: Include only the Transaction's Parent Transaction EPC Classes set up to be always reported, by making use of CURRENT at the ECReportSetSpec.

In the scope of APDL, all the above specifications and management attributes are augmented with design data borrowed from the XPDL V1.0 specification [27] so as to describe the processes workflow and to achieve the visualization of the RFID solution.

3.4 Specification Structure

Let us now briefly review the AspireRFID Process Description Language (APDL) specification structure [36]. An APDL document is based on XML syntax. As far as its vocabulary is concerned, the namespaces shown in Table 4 are used.

Element	**Namespace**
alelr:LRSpec	urn:epcglobal:alelr:xsd:1
ale:ECSpec	urn:epcglobal:ale:xsd:1
epcismd:EPCISMasterDataDocument	urn:epcglobal:epcis-masterdata:xsd:1
xpdl:Transitions	http://www.wfmc.org/2002/XPDL1.0
xpdl:TransitionRestrictions	
xpdl:ExtendedAttributes	
xpdl:Description	

Table 4: Namespaces used in APDL

The APDL has a tree structure, as shown in Figure 4 and Figure 5. The root element which contains the description of a complete supply chain management scenario is the Open Loop Composite Business Process (<apdl:OLCBProc/>).

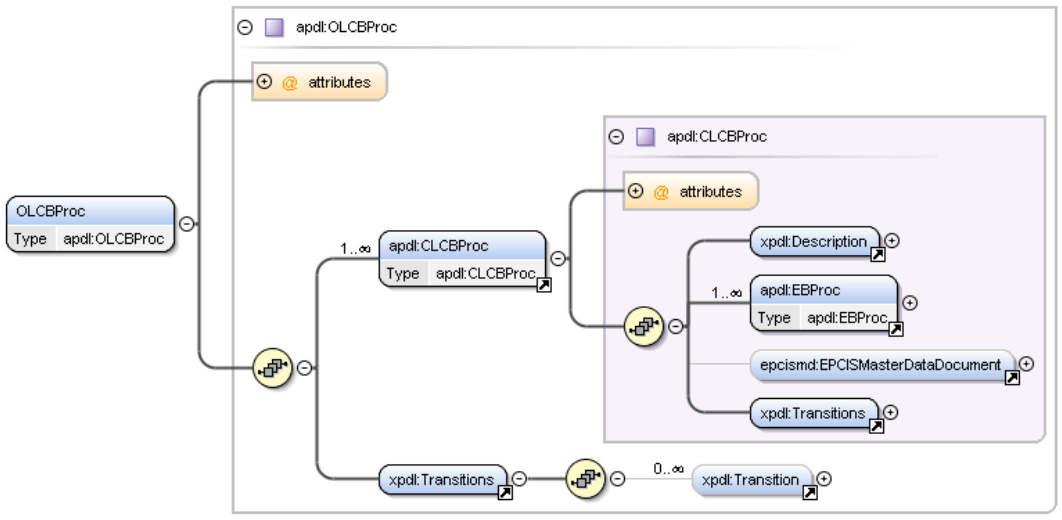


Figure 4: APDL' Schema design (OLCBProc)

"OLCBProc" contains a set of elements called Close Loop Composite Business Process (<apdl:CLCBProc/>) that are capable of describing a complete close loop supply chain scenario and the element of Transitions (<xpdl: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 specification [27].

Each of the "CLCBProc" elements, shown in Figure 4, are consisted of a set Elementary Business Process (<apdl:EBProc/>) elements that describe the elementary Business Transactions, the CLCBProc's Master Data in the form of an EPCIS Master Data Document (<epcismd:EPCISMasterDataDocument/>) [8] and the object of Transitions (<xpdl: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]. The EPCIS Master Data Document element inside the CLCBProc element carries only the information of the Business Location, the available Business Read Points, the traded items Dispositions and the company's available Business Steps.

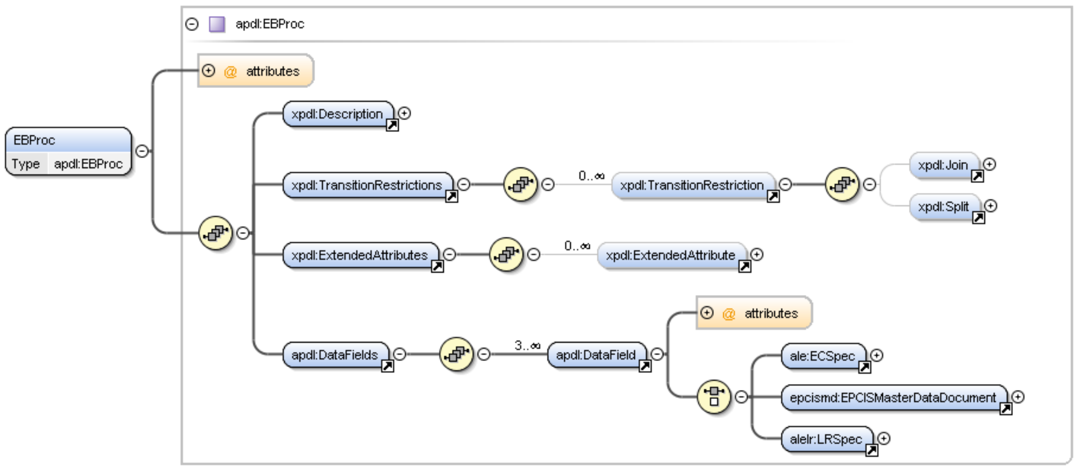


Figure 5: APDL's Schema design (EBProc)

The EBProc elements, shown in Figure 5, are the most important in the APDL specification since they contain the elementary business process description. For generating an RFID Event, which in APDL is interwoven with the "CLCBProc" element, specific configuration information is required from the BEG layer. Most of this information, as discussed in section 3.3, can be provided in the form of EPCglobal's specification files. These files include:

- The LRSpec, which is used for configuring the physical reader that is going to be used from the EBProc,
- The ECSpec, which defines the F&C's Event Cycle timings and ECReports that that will be used (see Table 3), and
- The Business Transaction description (see Table 1) which is stored at the company's Master Data as "BusinessTransactionID" Vocabulary (urn:epcglobal:epcis:vtype:BusinessTransaction), at the EPCIS repository, and APDL is using an EPCIS Master Data Document to store them.

Finally all the above specifications are enhanced by configuration data, which are required from the AspireRFID's PE (programmable Engine). And design data, which are required by the AspireRFID's BPWME.

So the EBProc element more specifically contains:

- A set of DataFields (<apdl:DataFields/>), that include the required
  - ECSpec (<ale:ECSpec/>),
  - LRSpec (<alelr:LRSpec/>) and
  - Master Data (<epcismd:EPCISMasterDataDocument/>) for describing a specific elementary business process transaction.

- A TransitionRestrictions [27] (<xpdl:TransitionRestrictions/>) element, containing a set of TransitionRestriction [27] (<xpdl:TransitionRestriction/>) elements which are used as design data.
- An ExtendedAttributes (<xpdl:ExtendedAttributes/>) element, containing a set of ExtendedAttribute (<xpdl:ExtendedAttribute/>) elements. This element is used in two ways. Firstly in order to store basic graphical representation data (x/y coordinates). In particular, the following key-value pairs are stored: XOffset, YOffset, CellHeight and CellWidth for the EBProc Object graphical representation. Secondly in order to store the basic configuration data. In particular, this set includes the following: (a) the EC Spec Subscription URI, (b) the ALE Client endpoint, (c) the ALE Logical Reader Client endpoint, (d) the EPCIS Capture interface endpoint and (e) the EPCIS query interface endpoint.
- And finally a description (<xpdl:Description/>) element, where optionally a simple description of the process can be stored.

More details about APDL can be found at Deliverable D4.4b [24] and the complete APDL schema definition can be found in Appendix II.

Section 4 Methodology

The development of PE was structured following a combination of two methodologies.

- The one was the *design science methodology* [28], which is characterized by its problem solving nature and it is used when there is a problem that is observed in a specific environment and there is no apparent solution design for solving it [28]. Design science is differentiated from theoretical research in how the concept of truth is valued. In, for example, positivist research the researcher set out to unearth an objective truth using large amount of data that is processed in a systematic way. On the other hand, in pragmatic research, which design science is a descendant of, truth is measured by what works [29], [30].
- The second methodology that was used is *requirement engineering*. The major activities in requirements engineering include elicitation, modelling and analysis, communication, agreement, evolution and Integration [34].

Therefore, the requirements of the PE design were steered by:

- The problem of easily applying/deploying an RFID solution to existing business processes. Some of the well-understood [29] business processes that was used to model the problem include the Receiving, Pick & Pack, Shipping, Palletization/De-palletization, Store to/ Remove from a shelf and moving within logical warehouses [16].
- The feedback collected from the developers and Users of the AspireRFID Open Source community identifying the complexity of deploying and configuring an RFID middleware.
- The feedback that was collected from SME's and the requirements that were identified by using surveys.
- The experience gained by evaluating Open Source EPC compliant RFID middleware (e.g. Fosstrak [1], Rifidi[33]) and commercial ones (e.g. BEA WebLogic RFID Enterprise Server[35]) as far as the ease of development and deployment of RFID oriented Business Processes is concerned.
- The requirements that were collected from the design and development process of the APDL (AspireRFID Process Description Language) [36] and the BPWME (Business Process Workflow Management Editor), where PE plays the role of a "linchpin" between them and the rest of the middleware.
- And finally the feedback acquired and analyzed from AspireRFID middleware pilot deployments (e.g. [32]) and the mapping of the trial processes into the APDL language.

Section 5 Role within the AspireRFID Architecture

The AspireRFID Programmable Engine (PE) module depicted in Figure 6 below resides between the AspireRFID IDE environment and the rest of the AspireRFID architecture. More specifically it is used as intermediary for the AspireRFID Business Process Workflow Management Editor (BPWME) plug-in to "register", "unregister", "update", "start" and "stop" a running instance of the AspireRFID middleware from the produced APDL xml file. It is also able to "get" from the ASPIRE middleware an already registered Business Process configuration and send it back to the BPWME. The Programmable Engine (PE) provides a standalone client able to register, unregister, get, start, stop and update from the AspireRFID middleware APDL xml files.

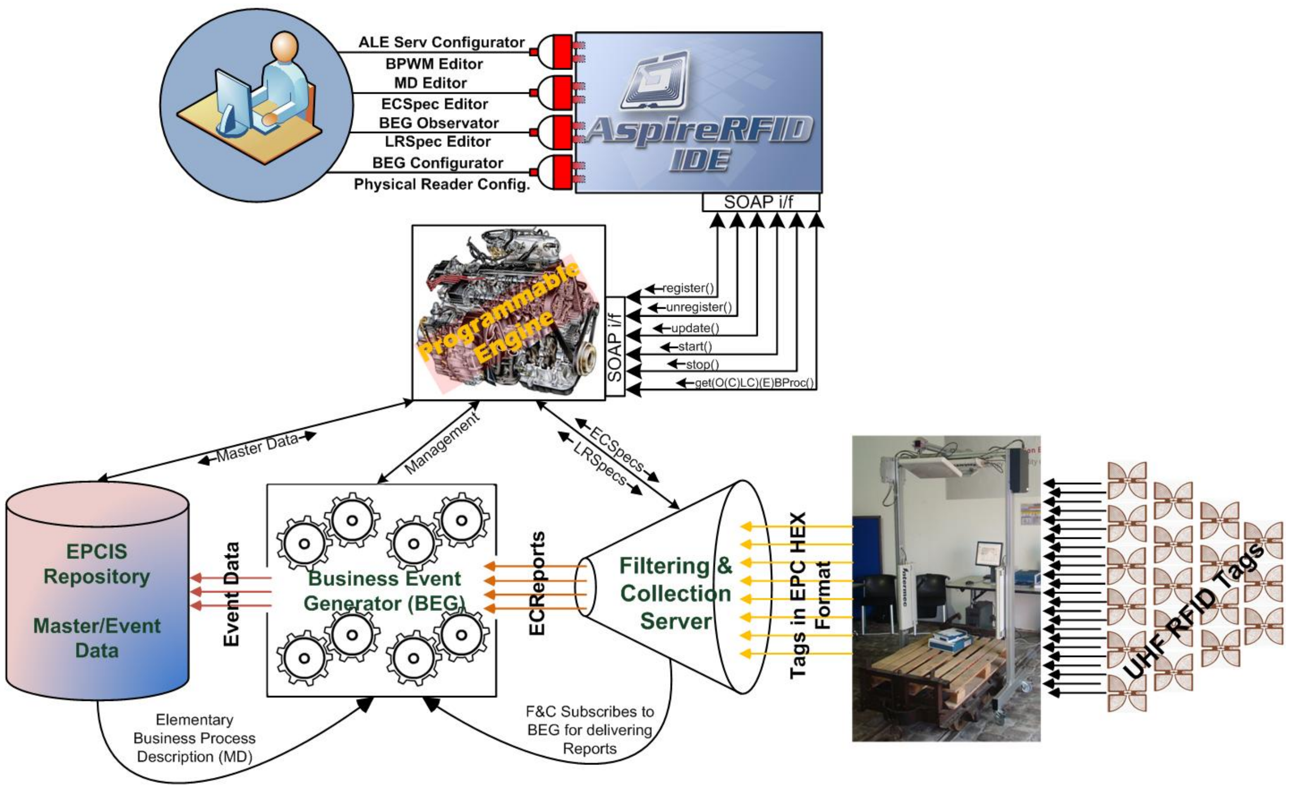


Figure 6: Programmable Engine role in the AspireRFID Architecture

The PE is fully based on Service Oriented Architecture (SOA) and reveals three interfaces, the OLCBProcControl API, the CLCBProcControl API and the EBProcControl API, that use the SOAP protocol for the exchanging of messages. These are fully detailed in Section 6 and the SOAP interfaces can be found in APENDIX III.

For the configuration of the three basic AspireRFID modules, Filtering and Collection (F&C), Business Event Generator (BEG) and Information Service repository (EPCIS); PE uses their specific and already defined SOAP interfaces. In the next paragraphs we are going to briefly describe such API's and the nature of PE's communication with these "underlying" modules. Furthermore, in the following Sections we are going to describe the specific PE's Interface and their implementation within 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. Most of the enhancements conducted to the EPCIS module have been contributed to the Fosstrak project.

The EPCIS module can be downloaded from the Fosstrak site (http://www.fosstrak.org/epcis/download.html).

5.1 Filtering and Collection

The main role of the filtering and collection module (F&C) within the ASPIRE architecture is to reduce the volume of captured RFID data and transform raw tag reads into streams of events suitable for processing by the application logic of Business Event Generator module. The main specifications ruling the F&C module functionality are the ECSpecs and the LRSpecs. For enabling the Programmable engine to configure the Filtering and collection module the use of the ALE and ALE-LR APIs (see Figure 1 above) will be required.

5.1.1 ALE Client

The Filtering & Collection Interface (ALE), see Figure 6 above, provides a standard interface to the Filtering & Collection role that applies to a large collection of use cases in which RFID Tags are inventoried (i.e., where the EPCs carried on the tags are read).

The purpose of the F&C Interface is to provide:

- Means for one or more client applications to request EPC data from one or more Tag sources.
- Means for one or more client applications to request that a set of operations be carried out on Tags accessible to one or more Tag sources. Such operations including writing, locking, and killing.
- Declarative means for client applications to specify what processing to perform on EPC data, including filtering, aggregation, grouping, counting, and differential analysis.
- Means for client applications to request data or operations on demand (synchronous response) or as a standing request (asynchronous response).
- Means for multiple client applications to share data from the same reader or readers, or to share readers' access to Tags for carrying out other operations, without prior coordination between the applications.
- A standardized representation for client requests for EPC data and operations, and a standardized representation for reporting filtered, collected EPC data and the results of completed operations.
- And finally to insulate client applications from knowing how many readers/antennas, and what makes and models of readers are deployed to constitute a single, logical Tag source.

Through a client that implements the ALE API [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 "**getECSpec**(specName : String) : ECSpec" which Returns the ECSpec that was provided when the ECSpec named specName was created by the define method.
- 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.

5.1.2 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 "**getLRSpec**(name : String) : LRSpec" which returns an LRSpec that describes the logical reader named name.
- The "**define**(name : String, spec : LRSpec) : void" which Creates a new logical reader named name according to spec.
- The "**undefine**(name : String) : void" which removes the logical reader named name.
- 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.

5.2 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.

5.2.1 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 6 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 5 below.

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 5: 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"

So the BEG component API, as shown in Table 5 above provides five methods for interaction with the BEG client which are all communicating with the BEG client by exchanging SOAP messages.

- The first method is the ***getEpcListForEvent*** (EventStatus getEpcListForEvent(String eventID)) which is used for returning to the BEG client an EventStatus object which contains the real time list of EPC ids and the transaction ID of a chosen Event (String eventID) from the list of events that the BEG component is already serving. So with the help of this method one can observe at real time the incoming IDs as they are reported to the BEG by the F&C component and are related with a specific transaction Event.
- The second method is the ***stopBegForEvent*** (boolean stopBegForEvent(String eventID)) which is used by the BEG client to stop serving a predefined Event by sending to it its specific EventID.
- The third method is the getStartedEvents (List<String> getStartedEvents()) which returns a list of Event IDs that the BEG component is serving.
- The fourth method is the ***startBegForEvent*** (boolean startBegForEvent(VocabularyElementType vocabularyElementType, String repositoryCaptureURL, String begListeningPort)) which is used to set up the BEG component for start serving a specific Event. More specifically this method takes the already pre described Elementary Business Transaction Event described at the Information Sharing repository's Master Data and uses it for configuring the Business Event Generator to create Business Events from the ECReports received from the port given as variable to the startBegForEvent method. If the method is successful it will return true otherwise it will return false.
- Finally, the fifth method is the ***getEventList*** (List<VocabularyElementType> getEventList(String repositoryQueryURL)) which is used for returning a list of all the available defined Events from a Company's EPCIS Master Data repository.

5.3 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 (see Figure 1 above) 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]

5.3.1 Capture Client

Because the Programmable Engine (PE) is not related with the Event Data generation, the Capture Client uses only the Master Data capture Interface. The specific Application Programming Interface (API) was specified and implemented by ASPIRE which has been contributed to the Fosstrak [1] EPCIS project. 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".

This new interface supports eleven control commands which are briefly described in the followinglist:

1. **alterVocElem** (String vocabularyType, String oldVocabularyElementURI, String newVocabularyElementURI)
   - Which is used to alter a vocabulary's Element URI.
2. **insertVocElem** (String vocabularyType, String vocabularyElementURI)
   - Which is used to insert a vocabulary's Element.
3. **massInsertVocElem** (String vocabularyType, ArrayList<String> vocabularyElementURIs)
   - Which is used to insert many vocabulary's Elements.
4. **deleteVocElem** (String vocabularyType, String vocabularyElementURI)
   - Which is used to delete a vocabulary's Element (When using single delete only the element with its attributes will be deleted).
5. **massDeleteVocElem** (String vocabularyType, ArrayList<String> vocabularyElementURIs)
   - Which is used to delete many vocabulary's Elements (When using mass delete only the listed elements with their attributes will be deleted).
6. **deleteWithDescendantsVocElem** (String vocabularyType, String vocabularyElementURI)
   - Which is used to delete a vocabulary's Element with its direct or indirect descendants (The element with its attributes and with all of its children elements and its children's attributes will be deleted).
7. **massDeleteWithDescendantsVocElem** (String vocabularyType, ArrayList<String> vocabularyElementURIs)
   - Which is used to delete many vocabulary's with their Elements and with their direct or indirect descendants. The elements with its attributes and with all of its children elements and its children's attributes will be deleted.
8. **insertOrAlterVocElemAttr** (String vocabularyType, String vocabularyElementURI, String vocabularyAttributeName, String vocabularyAttributeValue)
   - Which is used to insert or Update a vocabulary Element's Attribute. If the Vocabulary is not inserted yet it will be inserted. The vocabularyElementURI, vocabularyAttributeName pair should be unique so if it already exists it will be changed to the vocabularyAttribute entered or simply rewrite it.
9. **massInsertOrAlterVocElemAttr** (String vocabularyType, String vocabularyElementURI, HashMap<String, String> vocabularyAttributes)
   - Which is used to mass Insert or Update a vocabulary Element's Attributes. If the Vocabulary is not inserted yet it will be inserted. The vocabularyElementURI, vocabularyAttributeName pair (the vocabularyAttributeName is the key of the vocabularyAttributes HashMap) should be unique so if it already exists it will be changed to the vocabularyAttributeValue (which is the value of the vocabularyAttributes HashMap) entered or simply rewrite it.

10. **deleteVocElemAttr** (String vocabularyType, String vocabularyElementURI, String vocabularyAttributeName)
    - Which is used to delete a vocabulary Element's Attribute.
11. **massDeleteVocElemAttr** (String vocabularyType, String vocabularyElementURI, ArrayList<String> vocabularyAttributeNames)
    - Which is used to delete many vocabulary Element's Attributes.

5.3.2 Query Client

At runtime the Programmable Engine requires information from the EPCIS's Stored Master Data so as to Correctly Update/Save new ones through 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 6 PE Interfaces

This section defines normatively the PE's API. The External Interface is defined in the following sections and the implementation is described in Section 7. The programmable engine exposes three Interfaces. They are calcified in three different levels of control which reflects the three different levels of Business Processes that the APDL language provides which are the OLCBProc (Open Loop Composite Business Process), the CLCBProc (Close Loop Composite Business Process) and the EBProc (Elementary Business Process). So these PE Interfaces are named respectively:

- OLCBProcControl API
- CLCBProcControl API
- and EBProcControl API

The reason for creating these three different APIs is to give the agility to the PE to be used in complex as long as simple business processes depending on the given solution needs. Each of these APIs offers six configuration/control methods which are the register, unregister, update, start, stop and get(OLCBProc, CLCBProc, EBProc) and are described in the following three subsections.

6.1 OLCBProcControl API

The OLCBProcControl API is used to control a complete open loop composite business RFID solution. The OLCBProcControl API is consisted from 6 functions that are shown in Table 6 below.

```
register(openLoopCBProc : OLCBProc) : HashMap<String, String>
unregister(openLoopCBProc : OLCBProc) : HashMap<String, String>
update(openLoopCBProc : OLCBProc) : HashMap<String, String>
start(openLoopCBProc : OLCBProc) : HashMap<String, String>
stop(openLoopCBProc : OLCBProc) : HashMap<String, String>
getOLCBProc  (openLoopCBProcID  :  String,  endpoints  :  HashMap<String,
String>) : OLCBProc
```

Table 6: OLCBProcControl API

In Java the Table 6 API shown above, with the exceptions thrown from each method, would be:

- HashMap<String, String> **register**(OLCBProc openLoopCBProc) throws OLCBProcValidationException, NotCompletedExecutionException
- HashMap<String, String> **unregister**(OLCBProc openLoopCBProc) throws NoSuchOLCBProcIdException
- HashMap<String, String> **update**(OLCBProc openLoopCBProc) throws OLCBProcValidationException, NotCompletedExecutionException
- HashMap<String, String> **start**(OLCBProc openLoopCBProc) throws NoSuchOLCBProcIdException
- HashMap<String, String> **stop**(OLCBProc openLoopCBProc) throws NoSuchOLCBProcIdException

- OLCBProc **getOLCBProc** (String openLoopCBProcID, HashMap<String, String> endpoints) throws NoSuchOLCBProcIdException

So a PE implementation SHALL implement the methods of the OLCBProcControl API as specified in Table 7 below.

Method	Argument/ Result	Type	Description
register	openLoopCBProc	OLCBProc	This method initializes and starts the AspireRFID middleware to serve the described Open Loop Business Processes within the given APDL XML document. The method returns an unordered name/value pair list that denotes if the different middleware configuration steps were successful or not.
	[result]	HashMap <String, String>	
unregister	openLoopCBProc	OLCBProc	This method removes all the configurations for a specific Open Loop Composite Business process from the middleware. The method returns an unordered name/value pair list that denotes if the different middleware configurations removal steps were successful or not.
	[result]	HashMap <String, String>	
update	openLoopCBProc	OLCBProc	This method updates and restarts the AspireRFID middleware, for a specific and already registered OLCBProc, from the given APDL XML document. The method returns an unordered name/value pair list that denotes if the different middleware update steps were successful or not.
	[result]	HashMap <String, String>	
start	openLoopCBProc	OLCBProc	This method starts an already registered and stopped OLCBProc by giving as impute the OLCBProc. The method returns an unordered name/value pair list that denotes if the different middleware start steps were successful or not.
	[result]	HashMap <String, String>	
stop	openLoopCBProc	OLCBProc	This method stops an already registered (started) OLCBProc by giving as impute the OLCBProc. The method returns an unordered name/value pair list that denotes if the different middleware start steps were successful or not.
	[result]	HashMap <String, String>	
getOLCBProc	openLoopCBProcID	String	This method returns, an already defined, OLCBProc Object by giving as impute the Objects ID and an unordered name/value pair list of the endpoints of the targeted servers/modules running instances.
	endPoints	HashMap <String, String>	
	[result]	OLCBProc	

Table 7: OLCBProcControl Interface Methods

6.1.1 Error Conditions

Methods of the OLCBProcControl API signal error conditions to the client by means of exceptions. The following exceptions are defined (shown in Table 8 below).

Exception Name	Meaning
OLCBProcValidationException	The specified OLCBProc is invalid
NotCompletedExecutionException	The method was not executed/completed successfully
NoSuchOLCBProcIdException	The given OLCBProc ID does not exist (there is no registered OLCBProc with the given ID).

Table 8: Exceptions in the OLCBProcControl Interface

The exceptions that may be raised by the OLCBProcControl API method are indicated in Table 9 below.

Method	Exceptions
register	OLCBProcValidationException, NotCompletedExecutionException
unregister	NoSuchOLCBProcIdException
update	OLCBProcValidationException, NotCompletedExecutionException
start	NoSuchOLCBProcIdException
stop	NoSuchOLCBProcIdException
getOLCBProc	NoSuchOLCBProcIdException

Table 9: Exceptions Raised by each OLCBProcControl Interface Method

6.2 CLCBProcControl API

The CLCBProcControl API is used to control a complete close loop composite business RFID solution. The CLCBProcControl API is consisted from 6 functions that are shown in Table 10 below.

```
register(closeLoopCBProc : CLCBProc) : HashMap<String, String>
unregister(closeLoopCBProc : CLCBProc) : HashMap<String, String>
update(closeLoopCBProc : CLCBProc) : HashMap<String, String>
start(closeLoopCBProc : CLCBProc) : HashMap<String, String>
stop(closeLoopCBProc : CLCBProc) : HashMap<String, String>
getCLCBProc  (closeLoopCBProcID  :  String,  endpoints  :  HashMap<String,
String>) : CLCBProc
```

Table 10: OLCBProcControl API

In Java the Table 10 API shown above, with the exceptions thrown from each method, would be:

- HashMap<String, String> **register** (CLCBProc closeLoopCBProc) throws CLCBProcValidationException, NotCompletedExecutionException
- HashMap<String, String> **unregister**(CLCBProc closeLoopCBProc) throws NoSuchCLCBProcIdException

- HashMap<String, String> **update**(CLCBProc closeLoopCBProc) throws CLCBProcValidationException, NotCompletedExecutionException
- HashMap<String, String> **start**(CLCBProc closeLoopCBProc) throws NoSuchCLCBProcIdException
- HashMap<String, String> **stop**(CLCBProc closeLoopCBProc) throws NoSuchCLCBProcIdException
- CLCBProc **getCLCBProc**(String closeLoopCBProcID, HashMap<String, String> endpoints) throws NoSuchCLCBProcIdException

So a PE implementation SHALL implement the methods of the CLCBProcControl API as specified in Table 11 below.

Method	Argument/ Result	Type	Description
register	closeLoopCBProc	CLCBProc	This method initializes and starts the AspireRFID middleware to serve the described Close Loop Business Processes within the given APDL XML document. The method returns an unordered name/value pair list that denotes if the different middleware configuration steps were successful or not.
	[result]	HashMap <String, String>	
unregister	closeLoopCBProc	CLCBProc	This method removes all the configurations for a specific Close Loop Composite Business process from the middleware. The method returns an unordered name/value pair list that denotes if the different middleware configurations removal steps were successful or not.
	[result]	HashMap <String, String>	
update	closeLoopCBProc	CLCBProc	This method updates and restarts the AspireRFID middleware, for a specific and already registered CLCBProc, from the given APDL XML document. The method returns an unordered name/value pair list that denotes if the different middleware update steps were successful or not.
	[result]	HashMap <String, String>	
start	closeLoopCBProc	CLCBProc	This method starts an already registered and stopped CLCBProc by giving as impute the CLCBProc. The method returns an unordered name/value pair list that denotes if the different middleware start steps were successful or not.
	[result]	HashMap <String, String>	
stop	closeLoopCBProc	CLCBProc	This method stops an already registered (started) CLCBProc by giving as impute the CLCBProc. The method returns an unordered name/value pair list that denotes if the different middleware start steps were successful or not.
	[result]	HashMap <String, String>	
getCLCBProc	openLoopCBProcID	String	This method returns, an already defined, CLCBProc Object by giving
	endPoints	HashMap	

		<String, String>	as impute the Objects ID and an unordered name/value pair list of the endpoints of the targeted servers/modules running instances.
	[result]	CLCBProc	

Table 11: CLCBProcControl Interface Methods

6.2.1 Error Conditions

Methods of the CLCBProcControl API signal error conditions to the client by means of exceptions. The following exceptions are defined (shown in Table 12 below).

Exception Name	Meaning
CLCBProcValidationException	The specified CLCBProc is invalid
NotCompletedExecutionException	The method was not executed/completed successfully
NoSuchCLCBProcIdException	The given CLCBProc ID does not exist (there is no registered CLCBProc with the given ID).

Table 12: Exceptions in the CLCBProcControl Interface

The exceptions that may be raised by the CLCBProcControl API method are indicated in Table 13 below.

Method	Exceptions
register	CLCBProcValidationException, NotCompletedExecutionException
unregister	NoSuchCLCBProcIdException
update	CLCBProcValidationException, NotCompletedExecutionException
start	NoSuchCLCBProcIdException
stop	NoSuchCLCBProcIdException
getCLCBProc	NoSuchCLCBProcIdException

Table 13: Exceptions Raised by each CLCBProcControl Interface Method

6.3 EBProcControl API

The EBProcControl API is used to control a complete open loop composite business RFID solution. The EBProcControl API is consisted from 6 functions that are shown in Table 14 below.

```
register(elementaryBProc : EBProc) : HashMap<String, String>
unregister(elementaryBProc : EBProc) : HashMap<String, String>
update(elementaryBProc : EBProc) : HashMap<String, String>
start(elementaryBProc : EBProc) : HashMap<String, String>
stop(elementaryBProc : EBProc) : HashMap<String, String>
getEBProc  (elementaryBProcID  :  String,  endpoints  :  HashMap<String,
String>) : EBProc
```

Table 14: EBProcControl API

In Java the Table 14 API shown above, with the exceptions thrown from each method, would be:

- HashMap<String, String> **register**(EBProc elementaryBProc) throws EBProcValidationException, NotCompletedExecutionException
- HashMap<String, String> **unregister**(EBProc elementaryBProc) throws NoSuchEBProcIdException
- HashMap<String, String> **update**(EBProc elementaryBProc) throws EBProcValidationException, NotCompletedExecutionException
- HashMap<String, String> **start**(EBProc elementaryBProc) throws NoSuchEBProcIdException
- HashMap<String, String> **stop**(EBProc elementaryBProc) throws NoSuchEBProcIdException
- EBProc **getEBProc**(String elementaryBProcID, HashMap<String, String> endpoints) throws NoSuchEBProcIdException

So a PE implementation SHALL implement the methods of the EBProcControl API as specified in Table 15 below.

Method	Argument/ Result	Type	Description
register	elementaryBProc	EBProc	This method initializes and starts the AspireRFID middleware to serve the described Open Loop Business Processes within the given APDL XML document. The method returns an unordered name/value pair list that denotes if the different middleware configuration steps were successful or not.
	[result]	HashMap <String, String>	
unregister	elementaryBProc	EBProc	This method removes all the configurations for a specific Open Loop Composite Business process from the middleware. The method returns an unordered name/value pair list that denotes if the different middleware configurations removal steps were successful or not.
	[result]	HashMap <String, String>	
update	elementaryBProc	EBProc	This method updates and restarts the AspireRFID middleware, for a specific and already registered EBProc, from the given APDL XML document. The method returns an unordered name/value pair list that denotes if the different middleware update steps were successful or not.
	[result]	HashMap <String, String>	
start	elementaryBProc	EBProc	This method starts an already registered and stopped EBProc by giving as impute the EBProc. The method returns an unordered name/value pair list that denotes if the different middleware start steps were successful or not.
	[result]	HashMap <String, String>	
stop	elementaryBProc	EBProc	This method stops an already registered (started) EBProc by giving as impute the EBProc. The method returns an unordered name/value pair list that denotes if the different
	[result]	HashMap <String, String>	

			middleware start steps were successful or not.
getEBProc	elementaryBProcID	String	This method returns, an already defined, EBProc Object by giving as impute the Objects ID and an unordered name/value pair list of the endpoints of the targeted servers/modules running instances.
	endPoints	HashMap <String, String>	
	[result]	EBProc	

Table 15: EBProcControl Interface Methods

6.3.1 Error Conditions

Methods of the EBProcControl API signal error conditions to the client by means of exceptions. The following exceptions are defined (shown in Table 16 below).

Exception Name	Meaning
EBProcValidationException	The specified EBProc is invalid
NotCompletedExecutionException	The method was not executed/completed successfully
NoSuchEBProcIdException	The given EBProc ID does not exist (there is no registered EBProc with the given ID).

Table 16: Exceptions in the EBProcControl Interface

The exceptions that may be raised by the EBProcControl API method are indicated in Table 17 below.

Method	Exceptions
register	EBProcValidationException, NotCompletedExecutionException
unregister	NoSuchEBProcIdException
update	EBProcValidationException, NotCompletedExecutionException
start	NoSuchEBProcIdException
stop	NoSuchEBProcIdException
getEBProc	NoSuchEBProcIdException

Table 17: Exceptions Raised by each EBProcControl Interface Method

Section 7 PE API Implementation

In this section an overview of the PE API implementation design will be provided. The PE's implementation requires the use of the various existing API's described in Section 5 such as:

- ALE Reading API,
- ALE Logical Reader API,
- BEG API
- EPCIS Master Data Query and Capture API (SimpleMasterDataQuery and SimpleMasterDataCapture)

PE's most generic API implementation, the OLCBProcControl API, is described. The spessific API is superset comparing to the other two as far as the implementation is concerned so it covers the implementation of the CLCBProcControl and the EBProcControl APIs as well.

OLCBProcControl, as mentioned above, provides six control methods. These methods are:

- register
- unregister
- update
- start
- stop
- getOLCBProc

7.1 register

This method ("register(openLoopCBProc : OLCBProc) : HashMap<String, String>") initializes and starts the AspireRFID middleware to serve the described Open Loop Business Processes within the given APDL XML document. As soon as the method is executed it returns an unordered name/value pair list, as a HashMap Object, that denotes if the different middleware configuration steps were successful or not. Figure 7 below depicts the various steps PE implementation follows to "register" an APDL xml document into a running instance of AspireRFID middleware.

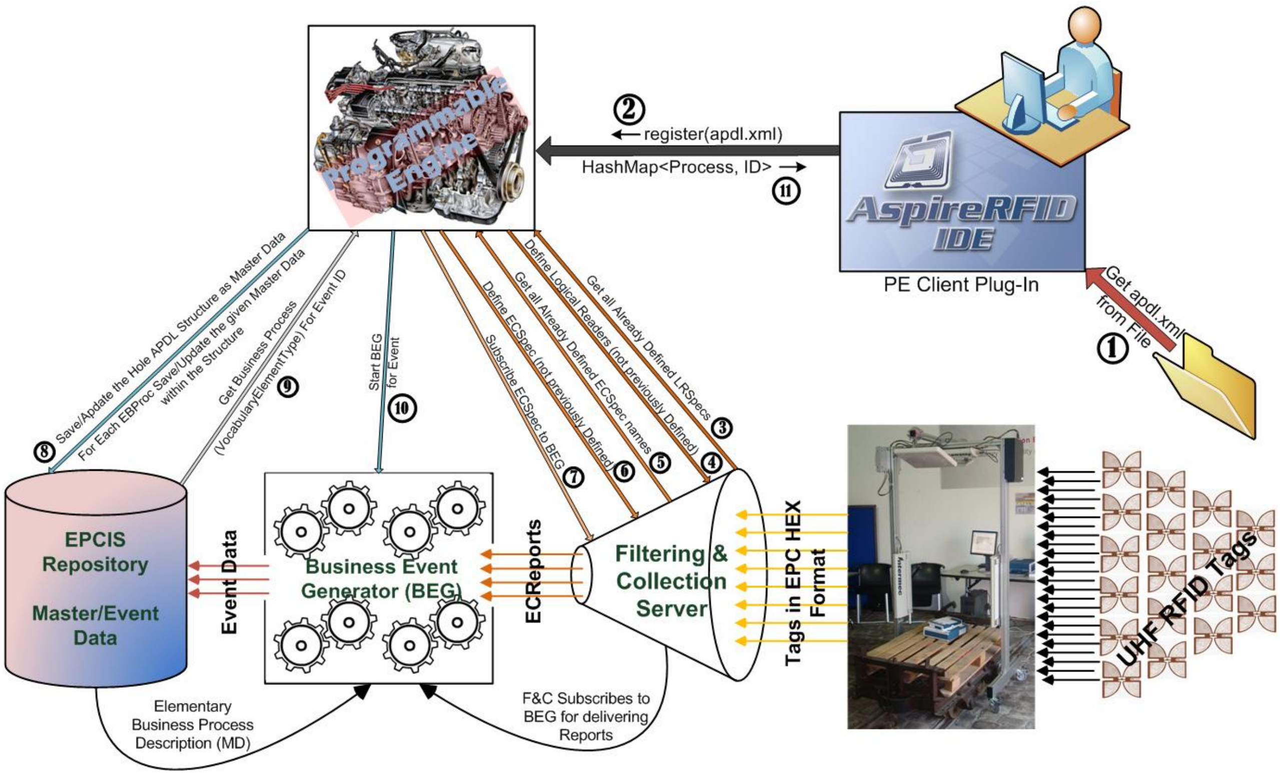

Programmable Engine
② register(apdl.xml)
HashMap<Process, ID> ⑪
AspireRFID IDE
PE Client Plug-In
Get apdl.xml from File ①
Get all Already Defined LRSpecs ③
Define Logical Readers (not previously Defined) ④
Get all Already Defined ECSpec names ⑤
Define ECSpec (not previously Defined) ⑥
Subscribe ECSpec to BEG ⑦
⑧ Save/Apdate the Hole APDL Structure as Master Data
For Each EBProc Save/Update the given Master Data within the Structure
Get Business Process (VocabularyElementType) For Event ID ⑨
Start BEG for Event ⑩
EPCIS Repository
Master/Event Data
Event Data
Business Event Generator (BEG)
ECReports
Filtering & Collection Server
Tags in EPC HEX Format
UHF RFID Tags
Elementary Business Process Description (MD)
F&C Subscribes to BEG for delivering Reports

Figure 7: **Programmable Engine's Register Steps**

Firstly the PE Client should retrieve from the APDL xml file (Step 1) the OLCBProc XML Element and map it to an OLCBProc Java Object with the help of JAXB (Java Architecture for XML Binding). The second step is to deliver to the PE server interface an APDL XML document encapsulated inside a SOAP message. At Appendix III the PE's register SOAP Interface can be found. For the Web Services Implementation Apache CXF framework was used and specifically with use of Servlet Transport implementation [26].

7.1.1 APDL Analysis and System Configuration

When an OLCBProc Object arrives to the PE, its CLCBProcs and its EBProcs, which reside inside every CLCBProc, are analyzed respectively. For each EBProc and by considering its parent's Objects (Attributes and IDs), the PE builds the required specification files in order to configure a running instance of the middleware. Another objective of this analysis is to create an Object "ProcessedEBProc", described in Table 18 below, which holds all the information required for configuring the AspireRFID middleware for each 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 its 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 18: ProcessedEBProc Object

7.1.1.1 ALE-LR Setup

For each EBProc, the APDL language offers the possibility to describe many Logical Reader specifications (LRSpecs) to be used by the ECSpec. These are saved at a Hashtable Java Object, when the APDL document passed as parameter is analyzed for every EBProc it describes.

The third Step as shown in Figure 7 above is to get all the Already-Defined LRSpec names from the Running Instance of the targeted F&C server, so as at the next step to "Define", by using the ALELR API [2], only Logical Readers that have not been priory 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.

7.1.1.2 ALE Setup

The ECSpec required for the middleware configuration are directly taken from each EBProc "DataField". The only change tha is done involves concatenating at Every ECReport name the EBProcs IDusing the "@" symbol. This ID will later be used by the BEG to distinguish the received ECReports from the various EBProcs configurations.

According to the same logic in Step 5 (Figure 7 above), the PE implementation gets all the defined ECSpec names, previously defined to the running instance so as to determine previously inexistent ECSpecs and "Undefine/Define" the existing ones and so to update its own tables accordingly (Step 6). After making BEG and id ready to receive ECReports, the next Step (Step 10) is to Subscribe the determined ECSpec, from the previous step (Step6), to the BEG Running instance ("ecSpecSubscriptionURI" Table 18 above).

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.

7.1.1.3 EPC Information Service Setup

The next point 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 already exist, from the provided "EPCISMasterDataDocument" [8] with the help of the EPCIS

Capture Interface. The EPCIS Capture End Point is provided by the EBProcs "ExtededAttributes" [24] with name "EpcisClientCaptureEndPoint". For the Business Transaction vocabulary type the OLCBProc Structure is considered and every given EBProc is saved as the Child of the CLCBProc it belongs to which in its turn is saved also as a child of the OLCBProc it belongs to. For the last task the various different Elements ID's are used to build the aforementioned structure. Furthermore every parent element (OLCBProc and CLCBProc) host's a Master Data document with information which is related with the children EBProc. So the OLCBProc host generic data (BusinessStep, Disposition, BusinessTransactionType) that apply to the whole supply chain whereas the CLCBProc host user data that are relevant for a specific business location (BusinessLocation and ReadPoint). These data are retrieved and saved to the EPCIS repository by taking special consideration at the TransactionID master Data vocabulary to keep their structure (OLCBProcID>CLCBProcID>EBProcID). This concludes the Step 7 of the PE's configuration process.

7.1.1.4 BEG Setup

Continuing, the Programmable Engine retrieves the EPCIS Query End-Point provided from the EBProcs "ExtededAttributes" [24] with name "EpcisClientQueryEndPoint" and uses it to get the VocabularyElementType [8] (Step 8) for the Elementary Business Processes (EBProc) whose ID is set 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 EBProcs "ExtededAttributes" [24] with name "ECSpecSubscriptionURI" and uses the "startBegForEvent" BEG client Service, (which requires as input the "VocabularyElementType", the "repositoryCaptureURL" and the "begListeningPort" as shown in Figure 7 above) the attributes of which have already been retrieved from the previous steps, to configure the BEG's functionality for the given EBProc (Step 9).

7.1.2 Final Step and Result

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 "register" execution status. The execution status is a name/value pair stored at a HashMap Object. The name is the ID of the executed step with the EBProc ID concatenated to it using the "@" symbol between the two strings. And the value is "succ" (Successful) or notsucc (Not Successful) which denotes if the executed step was successful or not. The step Ids are:

- getLRSpecs
- defineLRSpecs
- getECSpecNames
- defineECSpec
- subscribeECSpec
- saveMasterData
- getTransactionMasterData
- startBegForEvent

7.2 getOLCBProc

This method ("getOLCBProc (openLoopCBProcID : String, endpoints : HashMap<String, String>) : OLCBProc") returns, an already registered, OLCBProc Object by giving as impute the Objects ID and an unordered name/value pair list of the endpoints of the targeted servers/modules running instances. The names of the given name/value pair list are:

- AleClientEndPoint
- AleLrClientEndPoint
- EpcisClientCaptureEndPoint
- EpcisClientQueryEndPoint

Upon requesting this service, see Figure 8 below (step 1), the PE implementation queries at all the given EPCIS repositories (step 2) for the specific OLCBProc ID at the TransactionID vocabulary [8] accompanied with all its children and their attributes. The results are combined and the main structure of the OLCBProc object is build. If there is no much for the specific OLCBProcID a NoSuchOLCBProcIdException will be thrown.

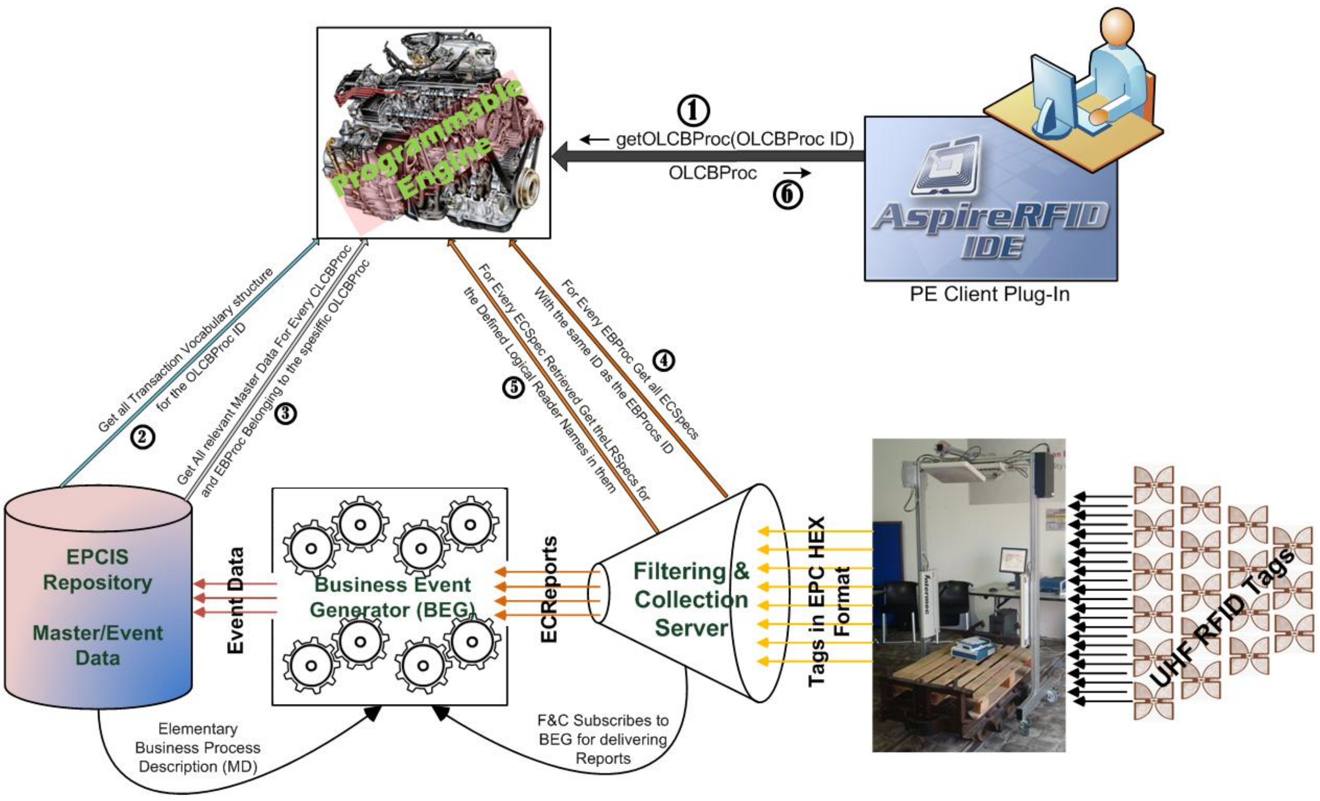

Programmable Engine
① ← getOLCBProc(OLCBProc ID)
OLCBProc → ⑥
AspireRFID IDE
PE Client Plug-In
Get all Transaction Vocabulary structure for the OLCBProc ID ②
Get All relevant Master Data For Every CLCBProc and EBProc Belonging to the spesific OLCBProc ③
For Every EBProc Get all ECSpecs With the same ID as the EBProcs ID ④
For Every ECSpec Retrieved Get theLRSpecs for the Defined Logical Reader Names in them ⑤
EPCIS Repository
Master/Event Data
Event Data
Business Event Generator (BEG)
ECReports
Filtering & Collection Server
Tags in EPC HEX Format
UHF RFID Tags
Elementary Business Process Description (MD)
F&C Subscribes to BEG for delivering Reports

Figure 8: **Programmable Engine's getOLCBProc Steps**

The step 3 (see Figure 8 above) is carried out by following the previous created OLCBProc structure and retrieve from the given EPCIS repositories all the remaining Master Data attributes that are "bind" with the EBProc (e.g. Disposition, Business Location, Business Step, Business Transaction Type and Read Point) and place the accordingly to where they belong. The Disposition, Business Step, Business Transaction Type are placed at the OLCBProc level as they are common for all the EBProcs. And the Business Location and Read Point are places at the CLCBProc level as they are specific for every different business location and not the whole OLCBProc.

The next step for the PE (step 4 see Figure 8 above) is for every, retrieved from the previous steps, EBProc to communicate with all the given Filtering and Collection endpoints and retrieve the ECSpecs with ID that equals with the EBProc ID.

For the ECSpecs retrieval two methods from the ALE API [2] are used:

- The "**getECSpecNames**() : List<String>" which Returns an unordered list of the names of all ECSpecs that are visible to the caller.
- The "**getECSpec**(specName : String) : ECSpec" which Returns the ECSpec that was provided when the ECSpec named specName was created by the define method.

After placing the retrieved ECSpecs to the EBProc they belong to the next step (step 5 see Figure 8 above) is to retrieve the LRSpecs which are listed to the logicalReaders [2] list at every ECSpec.

For the LRSpec retrieval two methods from the ALELR API [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 "**getLRSpec**(name : String) : LRSpec" which returns an LRSpec that describes the logical reader named name.

This concludes the building of the OLCBProc object which, at the final step (step 4 see Figure 8 above) is replied back to the PE client that made the request.

7.3 update

This method ("update(openLoopCBProc : OLCBProc) : HashMap<String, String>") updates and restarts the AspireRFID middleware, for a specific and already registered OLCBProc, from the given APDL XML document. As soon as the method is executed it returns an unordered name/value pair list, as a HashMap Object, that denotes if the different middleware configuration steps were successful or not. Figure 9 below depicts the various steps PE implementation follows to "update" an OLCBProc element at a running instance of AspireRFID middleware.

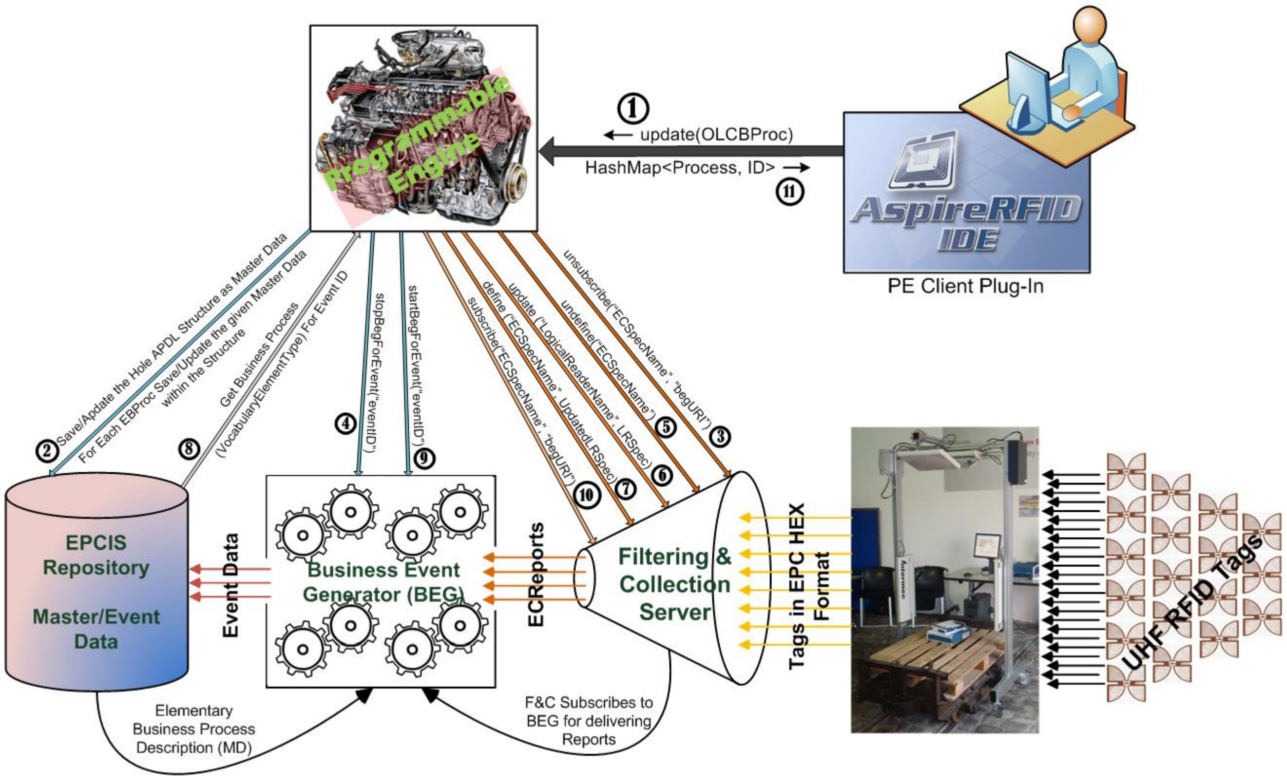

Programmable Engine
① update(OLCBProc)
HashMap<Process, ID> ⑪
AspireRFID IDE
PE Client Plug-In
② Save/Apdate the Hole APDL Structure as Master Data
For Each EBProc Save/Update the given Master Data within the Structure
⑧ Get Business Process (VocabularyElementType) For Event ID
④ stopBegForEvent("eventID")
⑨ startBegForEvent("eventID")
⑩ subscribe("ECSpecName", "begURI")
⑦ define ("ECSpecName", UpdatedLRSpec)
⑥ update ("LogicalReaderName", LRSpec)
⑤ undefine("ECSpecName")
③ unsubscribe("ECSpecName", "begURI")
EPCIS Repository
Master/Event Data
Event Data
Business Event Generator (BEG)
ECReports
Filtering & Collection Server
Tags in EPC HEX Format
UHF RFID Tags
Elementary Business Process Description (MD)
F&C Subscribes to BEG for delivering Reports

Figure 9: **Programmable Engine's update Steps**

As soon as the method is executed, step 1 of Figure 9 above, the EPCISs Master Data is updated (step 2). For Each EBProc the Disposition, Transaction Type, Read Point and Business Step are "updated" to the EPCIS Repository from the provided "EPCISMasterDataDocuments" [8] with the help of the EPCIS Capture Interface. The EPCIS Capture End Point is provided by the EBProcs "ExtededAttributes" [24] with name "EpcisClientCaptureEndPoint".

The third step is (Figure 9 above), for every available EBProc that belongs to the given OLCBProc Object, to unsubscribe the ECSpecs from the Business Event Generator. Step 4 is to command BEG to stop serving the Event by running the "stopBegFromEvent" command. Continuing at step 5 the PE execute the undefined command from the ALE API for the given ECSpec so as later at step 7 to update it by running the define command. Between step 5 and 7 the Programmable Engine runs the update command of the ALELR API so as to update the Filtering and collection instance with the updated LRSpec.

For the ECSpec update four methods from the ALE API [2] are used:

- The "**define**(specName : String, spec : ECSpec) : void" which Creates a new ECSpec having the name specName, according to spec.
- 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.

For the LRSpec update one methods from the ALELR API [2] is used:

- 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.

Continuing, the Programmable Engine retrieves the EPCIS Query End-Point provided from the EBProcs "ExtededAttributes" [24] with name "EpcisClientQueryEndPoint" and uses it to get the VocabularyElementType [8] (Step 8) for the Elementary Business Processes (EBProc) whose ID is set 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 EBProcs "ExtededAttributes" [24] with name "ECSpecSubscriptionURI" and uses the "startBegForEvent" BEG client Service, (which requires as input the "VocabularyElementType", the "repositoryCaptureURL" and the "begListeningPort" as shown in Figure 9 above) the attributes of which have already been retrieved from the previous steps, to configure the BEG's functionality for the given EBProc (Step 9). At step 10 the PE executes the subscribe command ("**subscribe**(specName : String, notificationURI : String) : void") of the ALE API which Adds a subscriber having the BEG URI to the set of current subscribers of the ECSpec named specName.

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 "update" execution status. The execution status is a name/value pair stored at a HashMap Object. The name is the ID of the executed step with the EBProc ID concatenated to it using the "@" symbol between the two strings. And the value is "succ" (Successful) or notsucc (Not Successful) which denotes if the executed step was successful or not. The step Ids are:

- updateLRSpecs
- undefineECSpecs
- unsubscribeECSpecs
- defineECSpec
- subscribeECSpec
- saveMasterData
- getTransactionMasterData
- stopBegForEvent
- startBegForEvent

7.4 stop

This method ("stop(openLoopCBProc : OLCBProc) : HashMap<String, String>") stops an already registered (started) OLCBProc by giving as impute the OLCBProc. As soon as the method is executed it returns an unordered name/value pair list, as a HashMap Object, that denotes if the different middleware configuration steps were successful or not. Figure 10 below depicts the various steps PE implementation follows to "stop" an OLCBProc element at a running instance of AspireRFID middleware.

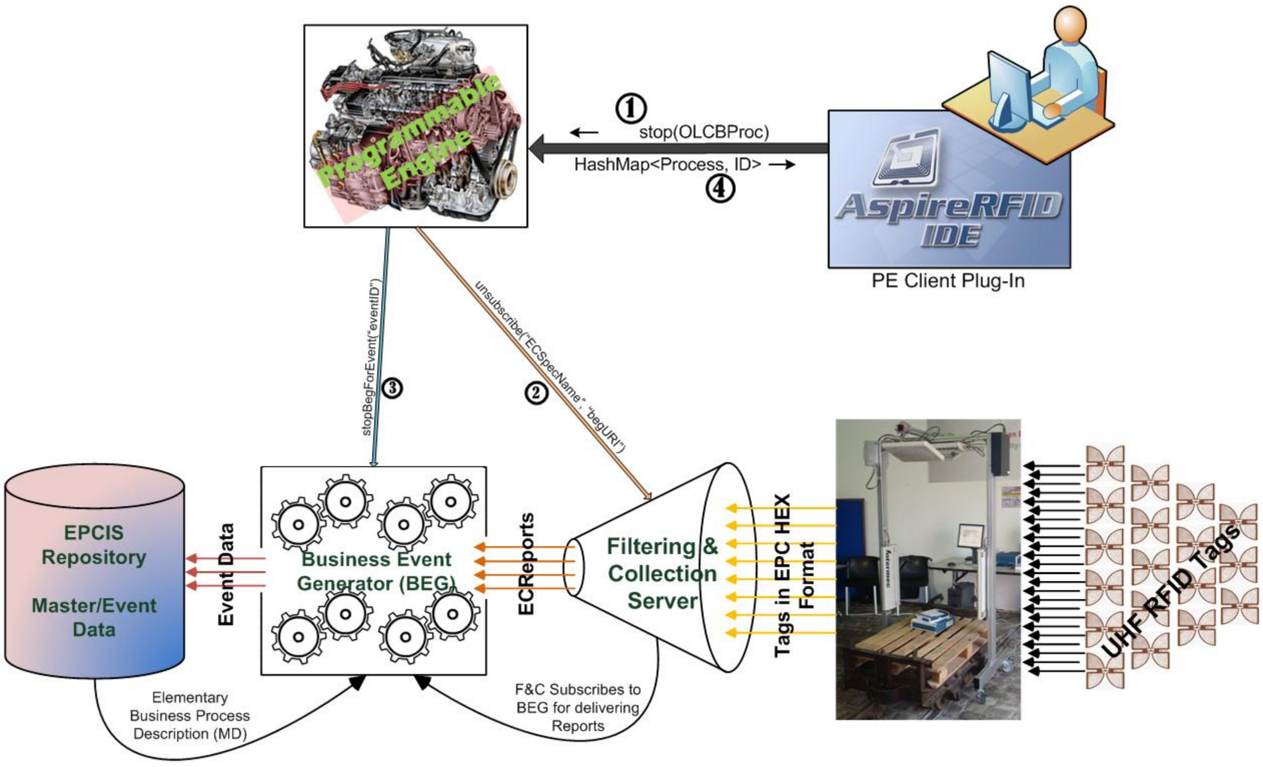

Programmable Engine
① stop(OLCBProc)
HashMap<Process, ID>
④
AspireRFID IDE
PE Client Plug-In
unsubscribe("ECSpecName", "begURI")
②
stopBegForEvent("eventID")
③
EPCIS Repository
Master/Event Data
Event Data
Business Event Generator (BEG)
ECReports
Filtering & Collection Server
Tags in EPC HEX Format
UHF RFID Tags
Elementary Business Process Description (MD)
F&C Subscribes to BEG for delivering Reports

Figure 10: Programmable Engine's stop Steps

As soon as the stop command is executed (step 1 of Figure 10 above) for every EBProc an ALE API "**unsubscribe**(specName : String, notificationURI : String) : void" command is executed (step 2), which Removes the BEG URI that has been specified as the notificationURI from the set of current subscribers of the ECSpec named specName.

The step 3 is to command BEG to stop serving the given Event by running the "**stopBegFromEvent**" command and by using as eventide the EBProcs ID. The final step (Step 4) is to send back to the Programmable Engine's client the "update" execution status. The execution status is a name/value pair stored at a HashMap Object. The name is the ID of the executed step with the EBProc ID concatenated to it using the "@" symbol between the two strings. And the value is "succ" (Successful) or notsucc (Not Successful) which denotes if the executed step was successful or not. The step Ids are:

- unsubscribeECSpecs
- stopBegForEvent

7.5 start

This method ("start(openLoopCBProc : OLCBProc) : HashMap<String, String>") starts an already registered and stopped OLCBProc by giving as impute the OLCBProc. As soon as the method is executed it returns an unordered name/value pair list, as a HashMap Object, that denotes if the different middleware configuration steps were successful or not. Figure 11 below depicts the various steps PE implementation follows to "start" an OLCBProc element at a running instance of AspireRFID middleware.

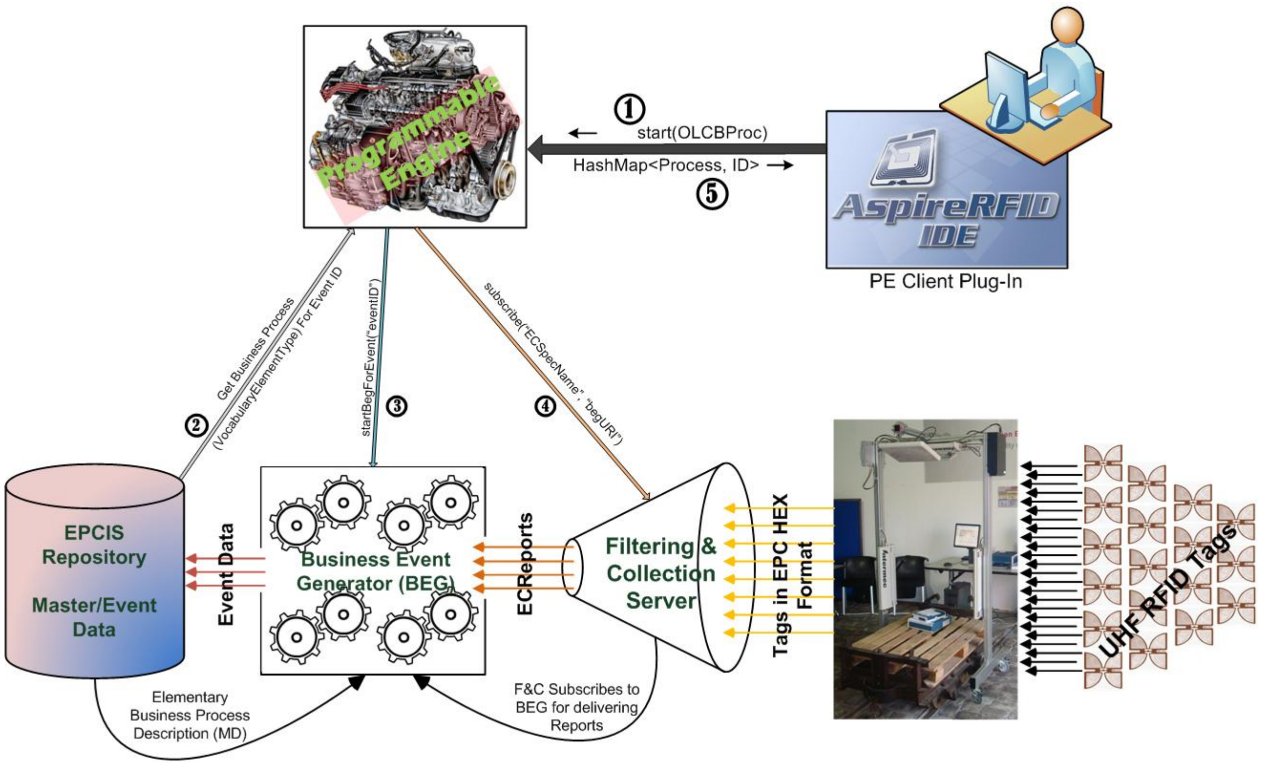

Programmable Engine
① start(OLCBProc)
HashMap<Process, ID>
⑤
AspireRFID IDE
PE Client Plug-In
② Get Business Process (VocabularyElementType) For Event ID
startBegForEvent("eventID") ③
④ subscribe("ECSpecName", "begURI")
EPCIS Repository
Master/Event Data
Event Data
Business Event Generator (BEG)
ECReports
Filtering & Collection Server
Tags in EPC HEX Format
UHF RFID Tags
Elementary Business Process Description (MD)
F&C Subscribes to BEG for delivering Reports

Figure 11: **Programmable Engine's start Steps**

As soon as the start command is executed (step 1 of Figure 11 above) for every EBProc the Programmable Engine retrieves the EPCIS Query End-Point provided from the EBProcs "ExtededAttributes" [24] with name "EpcisClientQueryEndPoint" and uses it to get the VocabularyElementType [8] (Step 2) for the Elementary Business Processes (EBProc) whose ID is set 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 EBProcs "ExtededAttributes" [24] with name "ECSpecSubscriptionURI" and uses the "startBegForEvent" BEG client Service, (which requires as input the "VocabularyElementType", the "repositoryCaptureURL" and the "begListeningPort" as shown in Figure 11 above) the attributes of which have already been retrieved from the previous step, to configure the BEG's functionality for the given EBProc (Step 3). At step 3 the PE executes the subscribe command ("**subscribe**(specName : String, notificationURI : String) : void") of the ALE API which Adds a subscriber having the BEG URI to the set of current subscribers of the ECSpec named specName.

After Subscribing the ECSpec to the BEG running instance the final step (Step 5) is to send back to the Programmable Engine's client the "start" execution status. The execution status is a name/value pair stored at a HashMap Object. The name is the ID of the executed step with the EBProc ID concatenated to it using the "@" symbol between the two strings. And the value is "succ" (Successful) or notsucc (Not Successful) which denotes if the executed step was successful or not. The step Ids are:

- subscribeECSpec
- getTransactionMasterData
- startBegForEvent

7.6 unregister

This method ("unregister(openLoopCBProc : OLCBProc) : HashMap<String, String>") removes all the configurations for a specific Open Loop Composite Business process from the middleware. As soon as the method is executed it returns an unordered name/value pair list, as a HashMap Object, that denotes if the different middleware configuration steps were successful or not. Figure 12 below depicts the various steps PE implementation follows to "update" an OLCBProc element at a running instance of AspireRFID middleware.

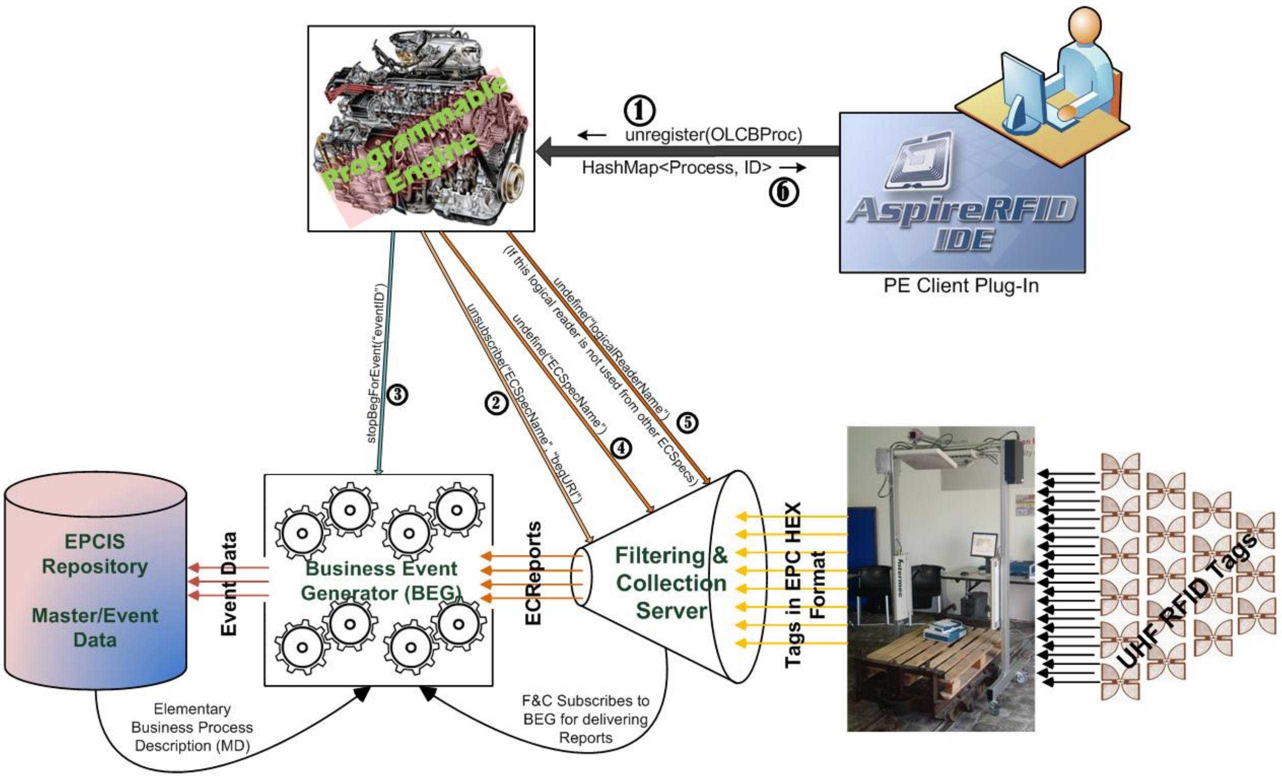

Programmable Engine
①
unregister(OLCBProc)
HashMap<Process, ID>
⑥
AspireRFID IDE
PE Client Plug-In
stopBegForEvent("eventID")
③
unsubscribe("ECSpecName", "begURI")
②
undefine("ECSpecName")
④
undefine("logicalReaderName")
(If this logical reader is not used from other ECSpecs)
⑤
EPCIS Repository
Master/Event Data
Event Data
Business Event Generator (BEG)
ECReports
Filtering & Collection Server
Tags in EPC HEX Format
UHF RFID Tags
Elementary Business Process Description (MD)
F&C Subscribes to BEG for delivering Reports

Figure 12: **Programmable Engine's unregister Steps**

Due to the fact that the EPCIS specification does not allow deleting any Master or Event data from the database no data deletion will be conducted at that level. As soon as the method is executed, step 1 of Figure 12 above, for every EBProc an ALE API "**unsubscribe**(specName : String, notificationURI : String) : void" command is executed (step 2), which Removes the BEG URI that has been specified as the notificationURI from the set of current subscribers of the ECSpec named specName. Step 3 is to command BEG to stop serving the Event by running the "stopBegFromEvent" command with the EBProcs ID. Continuing at step 4 the PE execute the undefined command from the ALE API ("**undefine**(specName : String) : void") with imput the EBProcs ID, which was used as the ECSpec's name, that Removes the ECSpec that was previously created by the define method. At step 5 the "**undefine**(name : String) : void" method is executed which removes the logical reader named name.

The final step (Step 6) is to send back to the Programmable Engine's client the "unregister" execution status. The execution status is a name/value pair stored at a HashMap Object. The name is the ID of the executed step with the EBProc ID concatenated to it using the "@" symbol between the two strings. And the value is "succ" (Successful) or notsucc (Not Successful) which denotes if the executed step was successful or not. The step Ids are:

- undefineECSpecs
- unsubscribeECSpecs
- undefineLRSpec
- stopBegForEvent

Section 8 How PE Changed the AspireRFID Configuration Process

In this section we compare the two AspireRFID configuration methods, the conventional and the PE client method, 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 have been previously defined as their generation is out of the scope of the Programmable Engine's features and capabilities.

8.1 Configuring in the Conventional Way

The configuration of the entire AspireRFID middleware for even a relatively simple scenario, for example like the one that is described in paragraph 9.1 below (where we define only an EBProc); requires a few steps and a number of AspireRFID "Configurators" (e.g. ECSpec Configurator, LRSpec Configurator and BEG configurator). Figure 13 below illustrates the different steps that an RFID integrator should follow to configure the AspireRFID middleware using such tools.

Concretely, to define an Elementary Business Process as shown in Figure 13 below we need to:

- Use the LRSpec Configurator plug-in where the required LRSpec xml file needs to 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 of our interest 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 previously configured for the BEG to receive Reports (at Step 6).

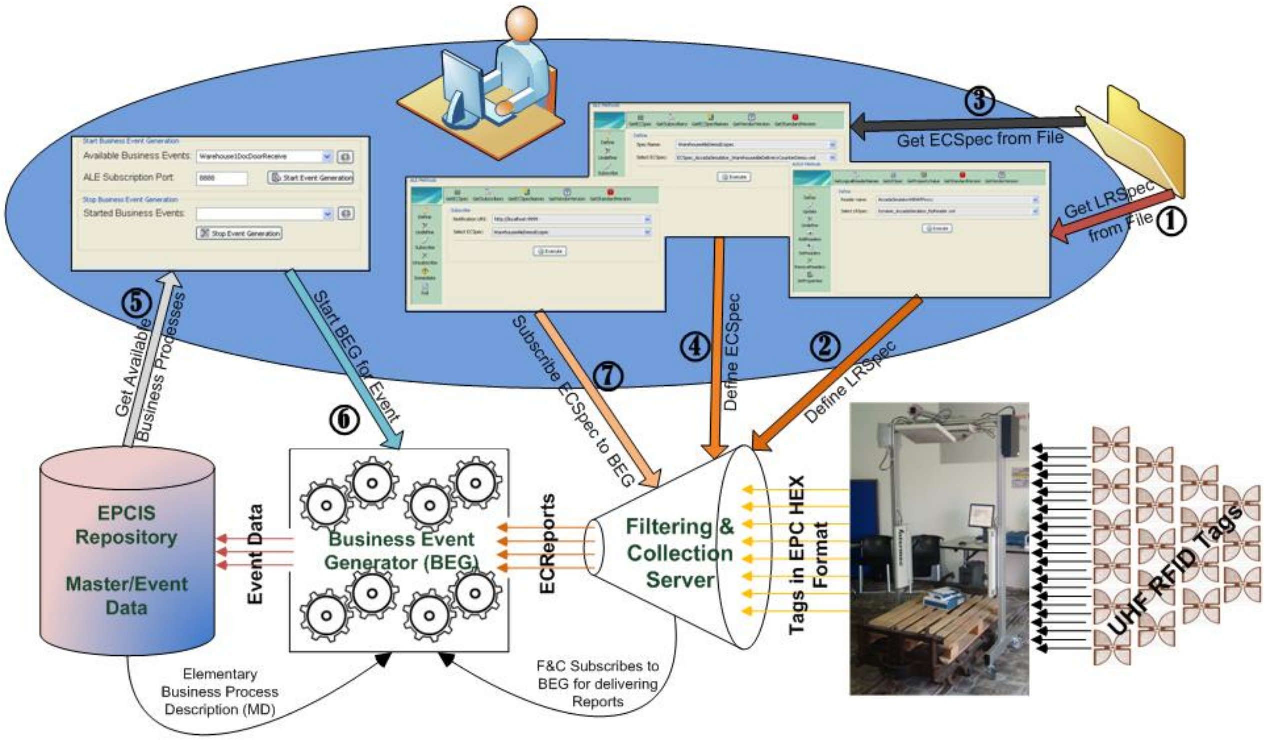


Figure 13: Required AspireRFID Configuring Steps without Programmable Engine

8.2 Configuring in the Programmable Engine Way

From the previous section we observe that, to configure the AspireRFID middleware through the conventional way, **"7" User Steps** where required for just one Elementary Business Process.

In this section we will describe what is required for the User to configure the AspireRFID middleware with the Programmable Engine's plug-in (client), again for only one EBProc. It worth's mentioning that even if we had to configure the AspireRFID middleware for "N" EBProcs (even for a complete Open Loop supply chain scenario) the steps to follow would be the same as the ones described below and would need to be followed only once.

So assuming that the APDL XML file has already been built to configure the AspireRFID middleware with the PE's User Client plug-in, the first step, as shown in Figure 14 below, would be to retrieve the "apdl.xml" file from the folder that is stored (Step 1). The second and final Step would be to use the "Register" service of the PE's through the PE Client (Step 2).

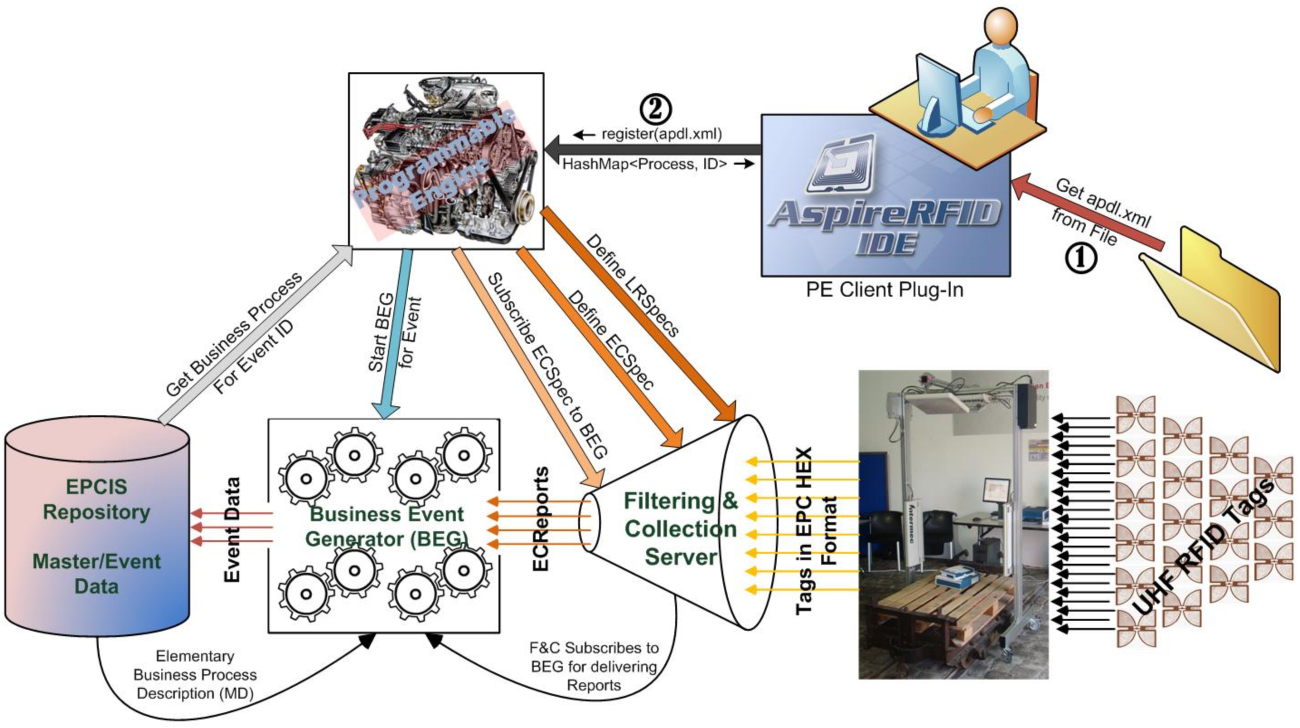


Figure 14: Required AspireRFID Configuring Steps with Programmable Engine

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 EBProcs required to describe the entire 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 Registering an OLCBProc Example

In this Section we will use the "Receiving" Example detailed in deliverable D4.3b (Programmable Filters – FML Specification) and in D4.4a (Programmable RFID Solutions Specification). In 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. In 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 entire AspireRFID middleware to serve a warehouse receiving process. So in this example we will analyze how an APDL XML file is used by the PE to configure it. So let's start with 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 places regular CPU orders to the Microchip Manufacturer. 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 pre-programmed RFID tags from their "Manufacturer". The received goods should be accompanied with a pre-programmed RFID enabled delivery document. The APDL XML file described in Deliverable D4.4a (Section 8.4) [24] should be used to configure an AspireRFID middleware (Figure 6 above) instance which for your convenience an updated version is available at Appendix II.

9.1.3 Registering the APDL Document

As shown in Figure 14 and described in Section 8.2 above, the configuration of the AspireRFID middleware requires two user steps. In this section we will give an example on how the APDL XML file, built in D4.4a, is used from the Programmable Engine to configure the AspireRFID middleware.

As shown in Figure 4 above, after getting the command (Step 1, 2) from the user to register this specific APDL file, the next step on the side of the Programmable Engine's is to analyze the received file and discern the different CLCBProc's and their EBProcs, only one of each in our case. 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 19 below.

```
<apdl:OLCBProc id="urn:epcglobal:fmcg:bti:openloopsupplychain"
  name="AcmeSupplyChainManagement">
  <epcismd:EPCISMasterDataDocument>
    <EPCISBody>
      <VocabularyList>
        <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessStep">
          <VocabularyElementList>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:receiving">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="receiving" />
            </VocabularyElement>
          </VocabularyElementList>
        </Vocabulary>

        <Vocabulary type="urn:epcglobal:epcis:vtype:Disposition">
          <VocabularyElementList>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:in_progress">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="In_progress" />
            </VocabularyElement>
          </VocabularyElementList>
        </Vocabulary>

        <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessTransactionType">
          <VocabularyElementList>
            <VocabularyElement id="urn:epcglobal:fmcg:btt:receiving">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Receiving" />
            </VocabularyElement>
          </VocabularyElementList>
        </Vocabulary>
      </VocabularyList>
    </EPCISBody>
  </epcismd:EPCISMasterDataDocument>

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

    <epcismd:EPCISMasterDataDocument>
      <EPCISBody>
        <VocabularyList>
          <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessLocation">
            <VocabularyElementList>
              <VocabularyElement id="urn:epcglobal:fmcg:loc:greece:pireus:mainacme">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="Acme" />
                <attribute id="urn:epcglobal:epcis:mda:Address" value="Akadimias 3" />
                <attribute id="urn:epcglobal:epcis:mda:City" value="Pireus" />
                <attribute id="urn:epcglobal:epcis:mda:Country" value="Greece" />
              </VocabularyElement>

              <VocabularyElement
id="urn:epcglobal:fmcg:loc:greece:pireus:mainacme,urn:epcglobal:fmcg:loc:acme:warehouse1">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="AcmeWarehouse1" />
                <attribute id="urn:epcglobal:epcis:mda:Read Point"
                  value="urn:epcglobal:fmcg:loc:rp:45632.Warehouse1DocDoor" />
              </VocabularyElement>
            </VocabularyElementList>
          </Vocabulary>

          <Vocabulary type="urn:epcglobal:epcis:vtype:ReadPoint">
            <VocabularyElementList>
              <VocabularyElement
```

```
                id="urn:epcglobal:fmcg:loc:rp:45632.Warehouse1DocDoor">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="Warehouse1DocDoor" />
              </VocabularyElement>
            </VocabularyElementList>
          </Vocabulary>
        </VocabularyList>
      </EPCISBody>
    </epcismd:EPCISMasterDataDocument>

    <apdl:EBProc id="urn:epcglobal:fmcg:bte:acmewarehouse1receive"
      name="Warehouse1DocDoorReceive">
          ………
    </apdl:EBProc>
    <xpdl:Transitions>
      </xpdl:Transition>
    </xpdl:Transitions>
  </apdl:CLCBProc>
</apdl:OLCBProc>
```

Table 19: CLCBProc Object [Register APDL Example]

For each EBProc and by taking in consideration their parents' Objects (Attributes and IDs), the required specification files are built 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 18 above, such object ultimately used to configure the AspireRFID running instance.

```
<apdl:EBProc id="urn:epcglobal:fmcg:bte:acmewarehouse1receive"
  name="Warehouse1DocDoorReceive">
  <xpdl:Description>Acme Warehouse 3 Receiving ReadPoint5 Gate3
  </xpdl:Description>
  <xpdl:TransitionRestrictions>
    </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:ExtendedAttribute
      Name="BegEngineEndpoint"
      Value="http://localhost:8080/aspireRfidBEG/begengine" />
  </xpdl:ExtendedAttributes>
  <apdl:DataFields>
   ………
  </apdl:DataFields>
</apdl:EBProc>
```

Table 20: AcmeWarehouse3Ship EBProc

From the part of the EBProc shown in Table 20 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.

9.1.3.1 ALE-LR Setup

For the APDL document only one Logical Reader is defined, which appears in Table 21 below.

```
<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>4000</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>
```

Table 21: LRSpec DataField

As soon as the PE retrieves the LRSpec from the APDL file it stores it to the **lrSpecs** element of the ProcessedEBProc (see Table 18) Object in a "SmartLabImpinjSpeedwayLogicalReader"/ "LRSpec Dynamic specification Object" pair manner. So the third Step as shown in Figure 7 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 the name is already included then an ALE-LR update ("SmartLabImpinjSpeedwayLogicalReader", LRSpec) is executed (Step 4).

9.1.3.2 ALE Setup

The ECSpec required for the AspireRFID configuration is given from the ECSpec "DataField" type. The change required to be done before storing it to the ecSpec Attribute (for later use) of the ProcessedEBProc Object (see Table 18) 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 and the EBProcs ID which is "urn:epcglobal:fmcg:bte:acmewarehouse1receive" so as to be used at 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">
  <ale:ECSpec includeSpecInReports="false">
    <logicalReaders>
      <logicalReader>SmartLabImpinjSpeedwayLogicalReader
      </logicalReader>
    </logicalReaders>
    <boundarySpec>
      <repeatPeriod unit="MS">5500</repeatPeriod>
      <duration unit="MS">5500</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.*
          </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 />
   </ale:ECSpec>
 </apdl:DataField>
```

Table 22: ECSpec DataField

Continuing with Step 5 (Figure 7) 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 only execute an ALE define("RecievingECSpec", ECSpec) command.

9.1.3.3 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 EBProcs 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 Table 23, to the EPCIS repository through the ASPIRE's EPCIS Master Data 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"
                value="Warehouse1DocDoorReceive" />
              <attribute
                id="urn:epcglobal:epcis:mda:event_type"
                value="ObjectEvent" />
              <attribute
                id="urn:epcglobal:epcis:mda:business_step"
                value="urn:epcglobal:fmcg:bizstep:receiving" />
              <attribute
                id="urn:epcglobal:epcis:mda:business_location"
                value="urn:epcglobal:fmcg:loc:acme:warehouse1" />
              <attribute
                id="urn:epcglobal:epcis:mda:disposition"
                value="urn:epcglobal:fmcg:disp:in_progress" />
```

```
                  <attribute
                    id="urn:epcglobal:epcis:mda:read_point"
                    value="urn:epcglobal:fmcg:loc:45632.Warehouse1DocDoor"/>
                  <attribute
                    id="urn:epcglobal:epcis:mda:transaction_type"
                    value="urn:epcglobal:fmcg:btt:receiving" />
                  <attribute
                    id="urn:epcglobal:epcis:mda:action"
                    value="ADD" />
                </VocabularyElement>
              </VocabularyElementList>
            </Vocabulary>
          </VocabularyList>
        </EPCISBody>
      </epcismd:EPCISMasterDataDocument>
    </apdl:DataField>
```

Table 23: EPCISMasterDataDocument DataField

For the Business Transaction EPCIS vocabulary type the entire OLCBProc Structure is considered and a given EBProc is saved as the Child of its CLCBProc and in its turn as a child of its OLCBProc. For the last task, the different Object ID's are used to build the aforementioned structure (Step 7 of the PE's configuration process). So the entire "EPCISMasterDataDocument" given in the EBProc description will be saved at the EPCIS repository as a child of its OLCBProc ("urn:ow2:aspirerfid:aprod:firstopenloopdescribedprocess") and its CLCBProc ("urn:epcglobal:fmcg:bti:acmesupplying"). See Table 19 above.

9.1.3.4 BEG Setup

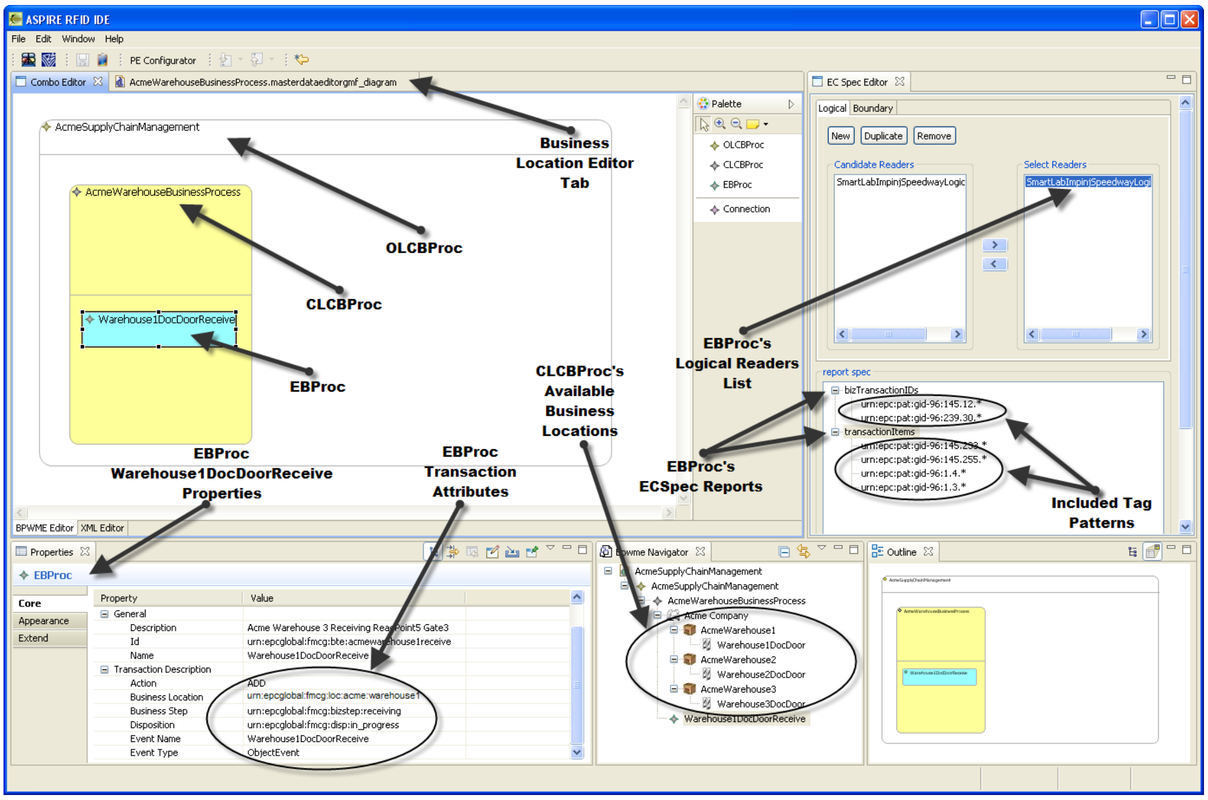
Continuing, the Programmable Engine uses the EPCIS Query End-Point, which was retrieved from the EBProcs "ExtededAttributes" above ("http://localhost:8080/aspireRfidEpcisRepository/query"), and use it to get the VocabularyElementType [8] (Step 8) for the "urn:epcglobal:fmcg:bte:acmewarehouse1receive" Elementary Business Processes (EBProc) ID which matches the BusinessTransaction's ID that BEG will be configured to serve. For the next step the PE uses the EPCIS Client Capture End-Point ("http://localhost:8080/aspireRfidEpcisRepository/capture") and use the "startBegForEvent" BEG client Service, by using as imput the VocabularyElementType that was prior retrieved, the "repositoryCaptureURL" and the "begListeningPort" as shown in Figure 7 above (Step 9).

After the aforementioned configuration BEG 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 18).

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

9.1.3.5 Building the Example's APDL file using the BPWME

Figure 10 depicts the above simple example in the BPWME AspireRFID IDE plug-in.



Candidate Readers: SmartLabImpinjSpeedwayLogic
Select Readers: SmartLabImpinjSpeedwayLogi

report spec
- bizTransactionIDs
  - urn:epc:pat:gid-96:145.12.*
  - urn:epc:pat:gid-96:239.30.*
- transactionItems
  - urn:epc:pat:gid-96:145.233.*
  - urn:epc:pat:gid-96:145.255.*
  - urn:epc:pat:gid-96:1.4.*
  - urn:epc:pat:gid-96:1.3.*

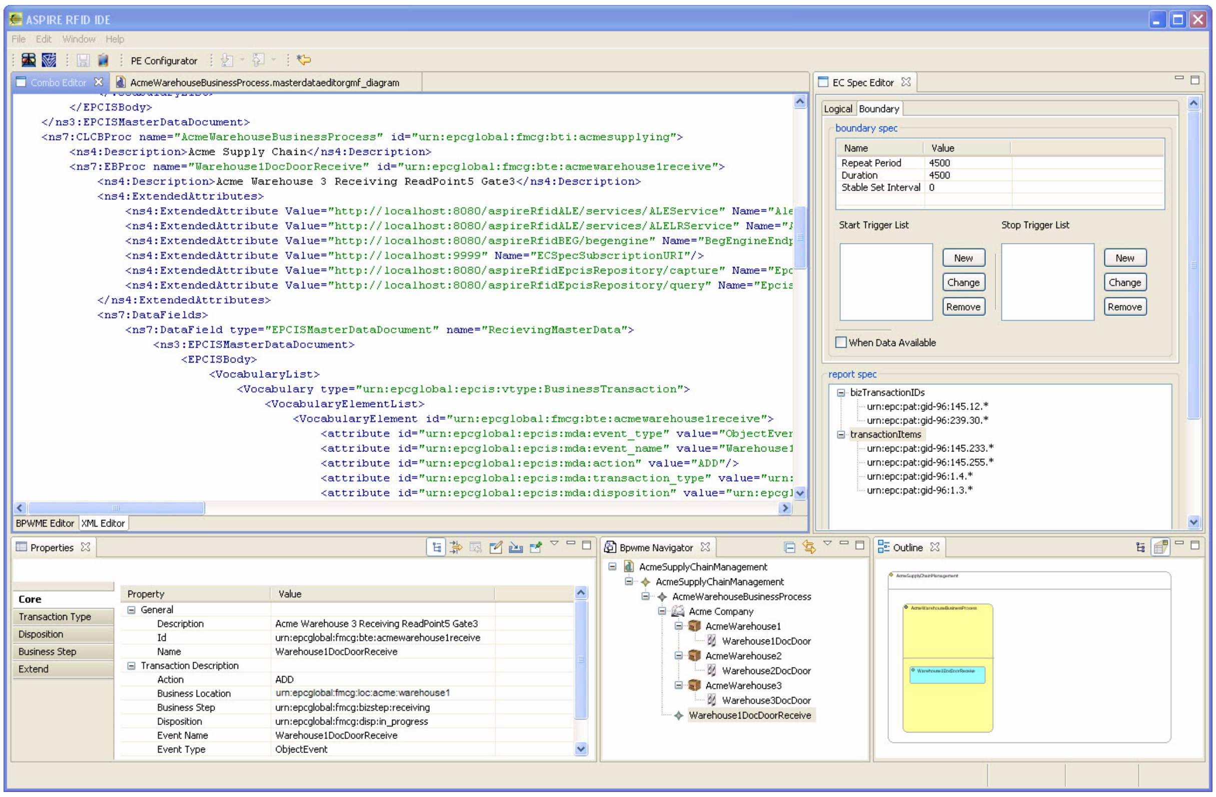
EBProc

Property	Value
General	
Description	Acme Warehouse 3 Receiving ReadPoint5 Gate3
Id	urn:epcglobal:fmcg:bte:acmewarehouse1receive
Name	Warehouse1DocDoorReceive
Transaction Description	
Action	ADD
Business Location	urn:epcglobal:fmcg:loc:acme:warehouse1
Business Step	urn:epcglobal:fmcg:bizstep:receiving
Disposition	urn:epcglobal:fmcg:disp:in_progress
Event Name	Warehouse1DocDoorReceive
Event Type	ObjectEvent

- AcmeSupplyChainManagement
  - AcmeSupplyChainManagement
    - AcmeWarehouseBusinessProcess
      - Acme Company
        - AcmeWarehouse1
          - Warehouse1DocDoor
        - AcmeWarehouse2
          - Warehouse2DocDoor
        - AcmeWarehouse3
          - Warehouse3DocDoor
      - Warehouse1DocDoorReceive

Figure 15: **AcmeWarehouse1Recieve Business Process in BPWME plug-in**

As we can see in Figure 15, when the user wants to generate an Object Event, which is the case in this example, the system automatically provides only the required fields for that specific event. So at the ECSpec the Event's required reports are already in place and is asked form the user only to fill the missing tag patterns. Moreover when the Logical reader is configured it is automatically added to the ECSpec that is used from this Event. We can also see that the available Business Locations, which have prior been set up with the help of Master Data Editor, are bind with the example's CLCBProc. So all the EBProcs in it can use them directly and are available at their Transaction Attributes. In Figure 16 below the automatically generated APDL xml file can be shown at the APDL tab of the main editor window.



```
        </EPCISBody>
    </ns3:EPCISMasterDataDocument>
    <ns7:CLCBProc name="AcmeWarehouseBusinessProcess" id="urn:epcglobal:fmcg:bti:acmesupplying">
        <ns4:Description>Acme Supply Chain</ns4:Description>
        <ns7:EBProc name="Warehouse1DocDoorReceive" id="urn:epcglobal:fmcg:bte:acmewarehouse1receive">
            <ns4:Description>Acme Warehouse 3 Receiving ReadPoint5 Gate3</ns4:Description>
            <ns4:ExtendedAttributes>
                <ns4:ExtendedAttribute Value="http://localhost:8080/aspireRfidALE/services/ALEService" Name="Ale
                <ns4:ExtendedAttribute Value="http://localhost:8080/aspireRfidALE/services/ALELRService" Name="A
                <ns4:ExtendedAttribute Value="http://localhost:8080/aspireRfidBEG/begengine" Name="BegEngineEndp
                <ns4:ExtendedAttribute Value="http://localhost:9999" Name="ECSpecSubscriptionURI"/>
                <ns4:ExtendedAttribute Value="http://localhost:8080/aspireRfidEpcisRepository/capture" Name="Epc
                <ns4:ExtendedAttribute Value="http://localhost:8080/aspireRfidEpcisRepository/query" Name="Epcis
            </ns4:ExtendedAttributes>
            <ns7:DataFields>
                <ns7:DataField type="EPCISMasterDataDocument" name="RecievingMasterData">
                    <ns3: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_type" value="ObjectEver
                                            <attribute id="urn:epcglobal:epcis:mda:event_name" value="Warehouse1
                                            <attribute id="urn:epcglobal:epcis:mda:action" value="ADD"/>
                                            <attribute id="urn:epcglobal:epcis:mda:transaction_type" value="urn:
                                            <attribute id="urn:epcglobal:epcis:mda:disposition" value="urn:epcgl
```

boundary spec

Name	Value
Repeat Period	4500
Duration	4500
Stable Set Interval	0

Start Trigger List

Stop Trigger List

☐ When Data Available

report spec

```
bizTransactionIDs
    urn:epc:pat:gid-96:145.12.*
    urn:epc:pat:gid-96:239.30.*
transactionItems
    urn:epc:pat:gid-96:145.233.*
    urn:epc:pat:gid-96:145.255.*
    urn:epc:pat:gid-96:1.4.*
    urn:epc:pat:gid-96:1.3.*
```

Core
Transaction Type
Disposition
Business Step
Extend

Property	Value
General	
Description	Acme Warehouse 3 Receiving ReadPoint5 Gate3
Id	urn:epcglobal:fmcg:bte:acmewarehouse1receive
Name	Warehouse1DocDoorReceive
Transaction Description	
Action	ADD
Business Location	urn:epcglobal:fmcg:loc:acme:warehouse1
Business Step	urn:epcglobal:fmcg:bizstep:receiving
Disposition	urn:epcglobal:fmcg:disp:in_progress
Event Name	Warehouse1DocDoorReceive
Event Type	ObjectEvent

```
AcmeSupplyChainManagement
    AcmeSupplyChainManagement
        AcmeWarehouseBusinessProcess
            Acme Company
                AcmeWarehouse1
                    Warehouse1DocDoor
                AcmeWarehouse2
                    Warehouse2DocDoor
                AcmeWarehouse3
                    Warehouse3DocDoor
            Warehouse1DocDoorReceive
```

Figure 16: **Produced APDL xml file (APDL tab) of the BPWME**

9.1.3.6 Process Description

ACME gives an order with a specific deliveryID to the Microchip Manufacturer. With the previous action AspireRFID Connector subscribes to the AspireRFID EPCIS Repository to retrieve events concerning the specific deliveryID. As visualized in Figure 3 the order arrives to ACME's premises. ACME's RFID portal (ReadPoint1) reads the deliveryID and all the products that follow with the help of WarehouseRfidReader1. AspireRFID ALE filters out the readings and sends two reports to AspireRFID BEG, one with the deliveryID and one with all the products tags. AspireRFID BEG collects these reports, binds the deliveryID with the products tags and sends this event to the AspireRFID EPCIS Repository. The AspireRFID EPCIS Repository informs the Connector [37] for the incoming event which in his turn sends this information to ACME's WMS. When the WMS confirms that all the requested products were delivered it sends a "transaction finish" message to the AspireRFID Connector which in his turn unsubscribe for the specific deliveryID and sends a "transaction finish" to the RFID Repository.

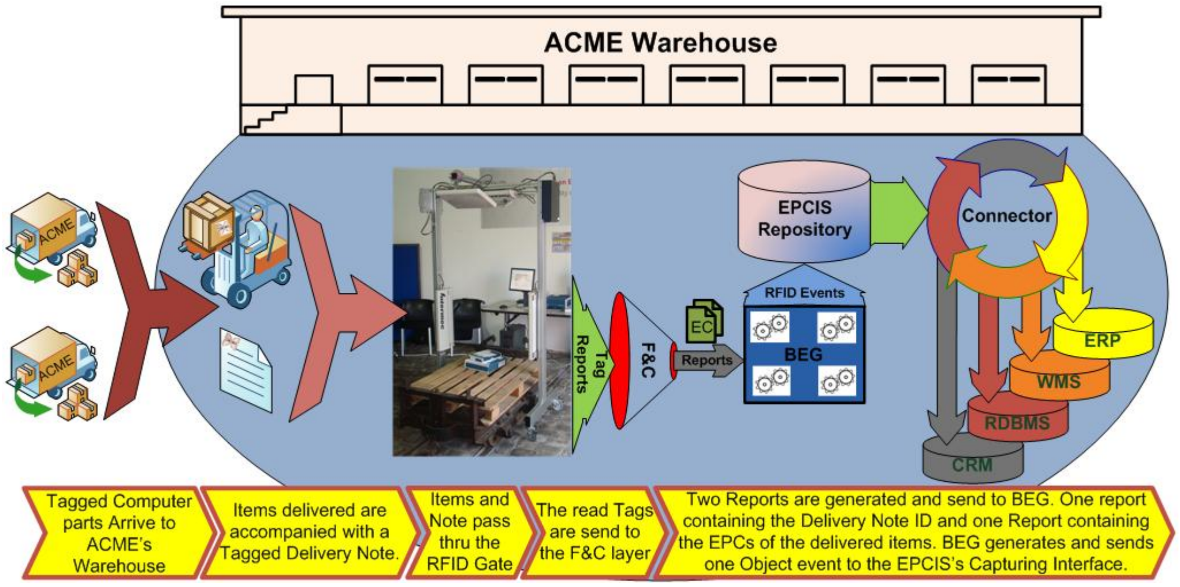
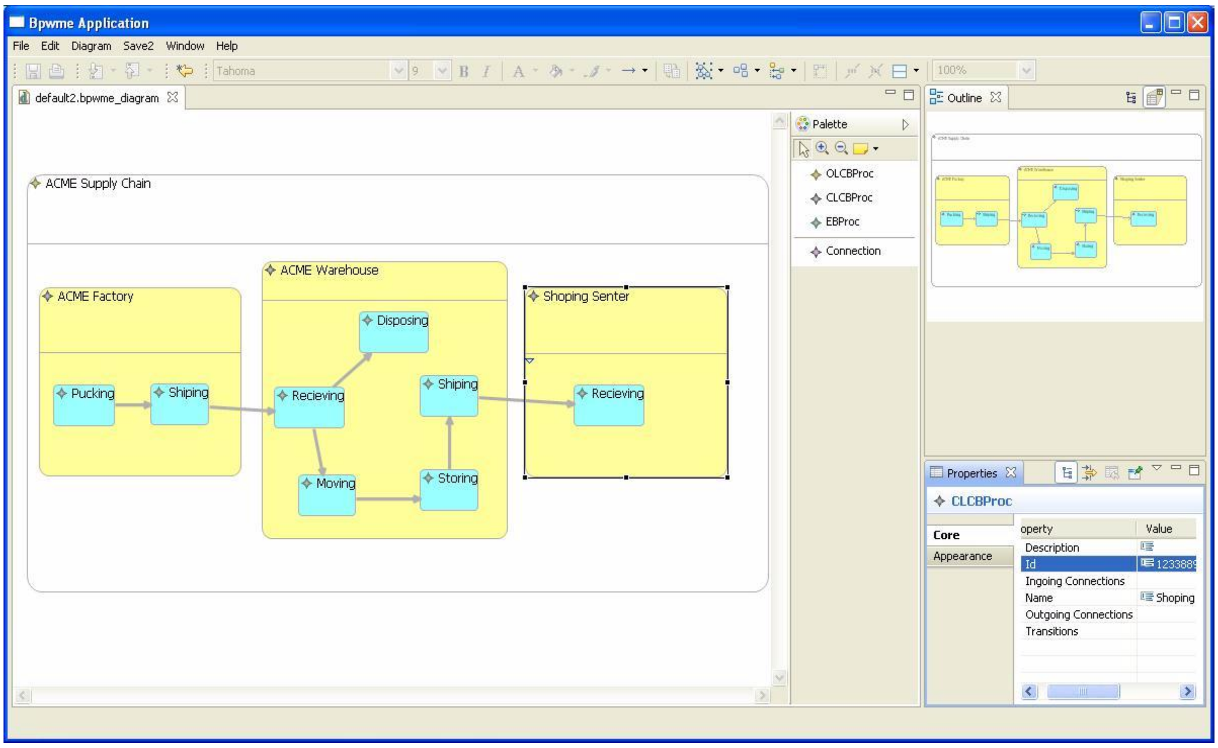

ACME Warehouse
ACME
ACME
EPCIS Repository
Connector
RFID Events
EC
Reports
BEG
Tag Reports
F&C
ERP
WMS
RDBMS
CRM
Tagged Computer parts Arrive to ACME's Warehouse
Items delivered are accompanied with a Tagged Delivery Note.
Items and Note pass thru the RFID Gate
The read Tags are send to the F&C layer
Two Reports are generated and send to BEG. One report containing the Delivery Note ID and one Report containing the EPCs of the delivered items. BEG generates and sends one Object event to the EPCIS's Capturing Interface.

Figure 17: **Acme computer parts Delivery Example**

Section 10 Business Process Workflow Management Editor (BPWME) Introduction

ASPIRE Architecture introduces a tool, called Business Process Workflow Management Editor (BPWME) plug-in (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.

One of the benefits of an RFID Solution Language is that it can boost visual development of RFID solutions, which could obviate the need for tedious low-level programming. In the case of the APDL language, ASPIRE have designed and prototyped an Eclipse plug-in to enable the visual modelling and configuration of RFID enabled processes. This tool is conveniently called Business Process Workflow Management Editor (BPWME) and is illustrated in Figure 18 below.



operty	Value
Description	
Id	1233889
Ingoing Connections	
Name	Shoping
Outgoing Connections	
Transitions	

Figure 18: BPWME plug-in

BPWME provides to the RFID designer the ability to describe a complex RFID solution with the help of a workflow diagram and to be guided to give as input all the required information so as to build the desired EBProcs.

Our experience with BPWME shows that a workflow process design is a more straightforward procedure compared to detailed configurations of distributed software and hardware components by using various configuration interfaces. As such, the use of the workflow editor reduces significantly the time and effort required to configure an RFID solution. Additionally, it provides the ability of registering and storing complete RFID solutions in a single configuration file. This can greatly facilitate reusability across classes of similar RFID solutions, since it allows adapting existing solutions rather than developing from scratch. Furthermore, BPWME reduces the knowledge overhead imposed by the need to use various tools, while at the same time easing debugging and maintenance efforts.

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

In Figure 18 above 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.

The Programmable Engine's client will be embedded in this tool so as to achieve the direct registration, of the AspireRFID middleware as soon as an APDL xml file is created. The ability of real time interaction of the BPWME plug-in through the PE's "olcbproccontrol", "clcbproccontrol" and "ebproccontrol" Interfaces will be investigated.

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

10.1 Graphical Modelling Framework

GMF (Graphical Modelling Framework) is a framework for creating a generic graphical interface in eclipse by combining EMF (Eclipse Modelling 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 19 below shows the main components and models used during GMF-based development.

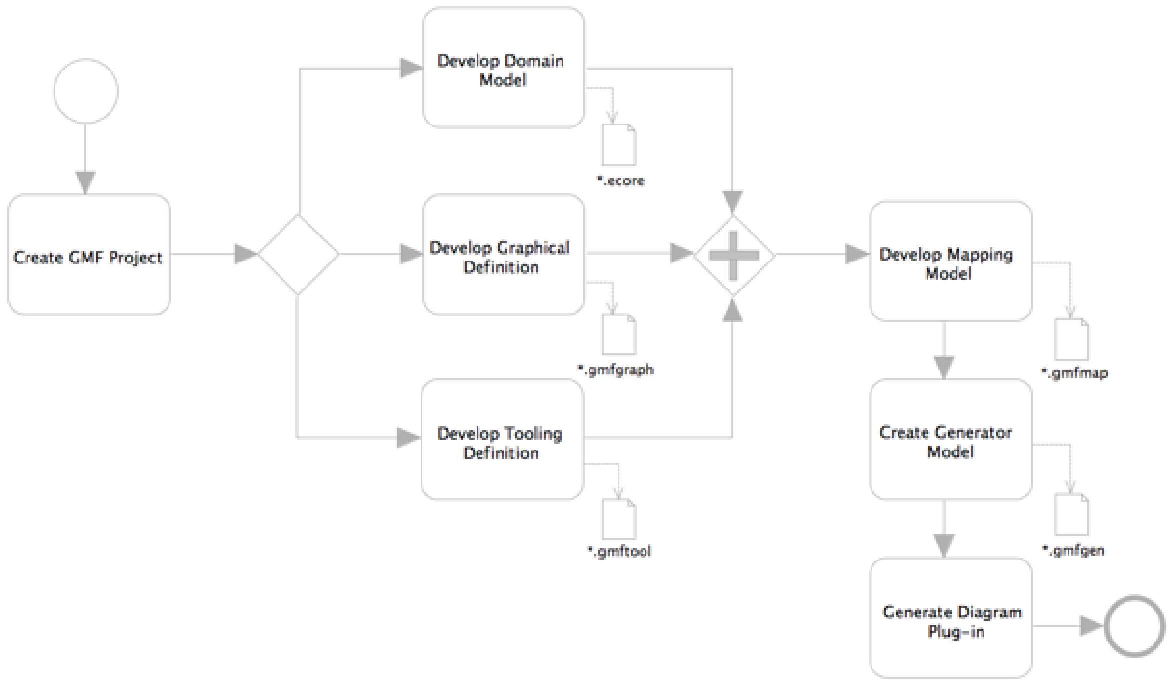
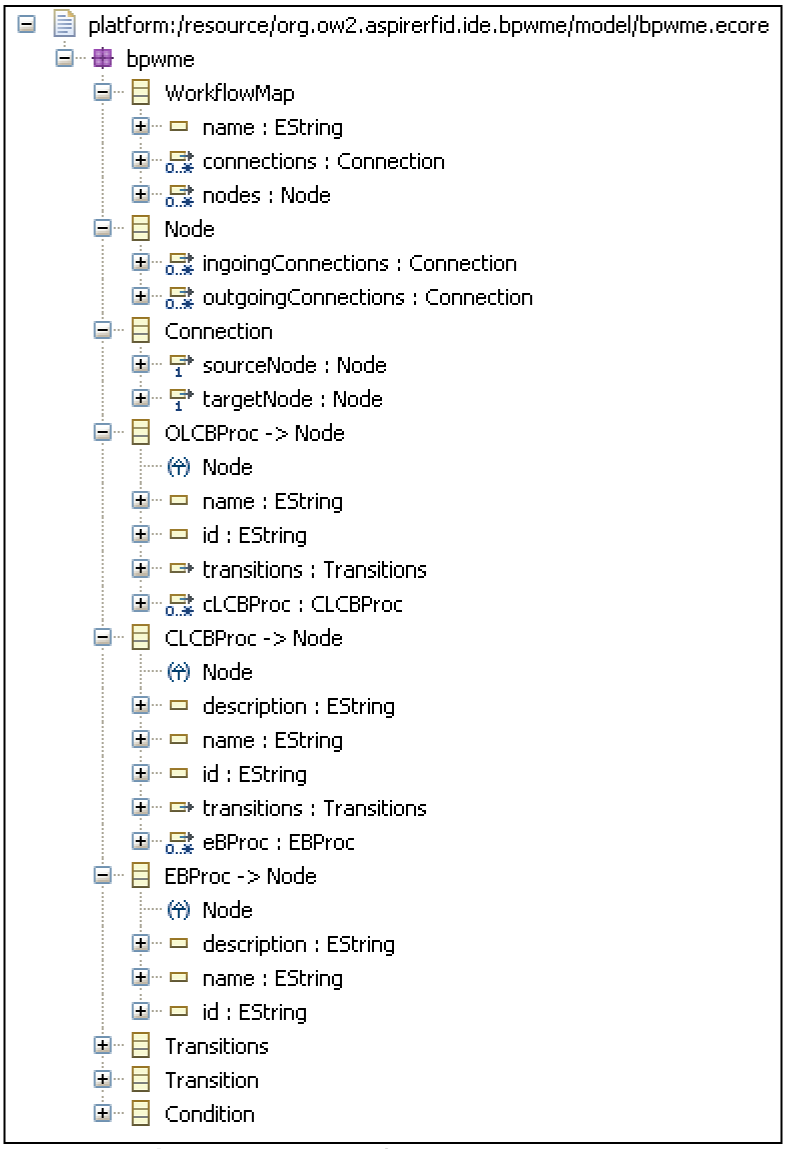


Figure 19: Main components and models used during GMF-based development [25]

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 20 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 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.



```
platform:/resource/org.ow2.aspirerfid.ide.bpwme/model/bpwme.ecore
  bpwme
    WorkflowMap
      name : EString
      connections : Connection
      nodes : Node
    Node
      ingoingConnections : Connection
      outgoingConnections : Connection
    Connection
      sourceNode : Node
      targetNode : Node
    OLCBProc -> Node
      Node
      name : EString
      id : EString
      transitions : Transitions
      cLCBProc : CLCBProc
    CLCBProc -> Node
      Node
      description : EString
      name : EString
      id : EString
      transitions : Transitions
      eBProc : EBProc
    EBProc -> Node
      Node
      description : EString
      name : EString
      id : EString
    Transitions
    Transition
    Condition
```

Figure 20: 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 Conclusions

The deliverable has outlined the specifications of the ASPIRE programmable engine and demonstrated its features, especially those aiming at easing the configuration of ASPIRE-based solutions and maximising its adaptability.

Briefly, the programmable engine allows users to deploy highly configurable complex RFID solutions at a fraction of time comparing with the classic configuration methods. The deployment is enhanced by a business scenario which is defined using a business process language (APDL) that is translated to middleware-understandable code by the programmable engine. The deliverable outlines the interfaces of the Aspire Programmable Engine and it moreover elaborates on how the PE communicates with other ASPIRE RFID modules.

PE in combination with the APDL is a solution that, if properly employed, has the potential to offer benefits, such as TCO minimization, standards compliance, facilitation in the development of complex RFID process-based solutions etc. However, PE is not a solution that is applicable under any circumstances. PE should be used in complex RFID solutions where all the EPC layers are used and not in simple, standalone RFID applications.

PE development involved several challenges, among which are worth mentioning striking the balances between simplicity and flexibility; maintaining compatibility with the existing infrastructure (specifically that of EPCglobal); and providing the adequate level of abstraction to facilitate use.

The advantages of the PE approach are clear: firstly, the code is isolated from the specifics of the implementation. Secondly, implementations happen in an atomic way, therefore avoiding errors and inconsistencies from partial, uncoordinated developments. Thirdly, previous implementations can be 'reused', so enabling a rich source of applications and benefits. Finally, implementations are transferrable to other platforms, applications and industries.

Finally in this deliverable except from the APIs description we have included a brief comparison showing how the PE facilitates the development of RFID solutions based on the ASPIRE middleware and a brief example of the use of the programmable engine.

Section 12 List of Figures

Section 13 List of Tables

Section 14 List of Acronyms

ALE	Application Level Event
APDL	AspireRFID Process Description Language
API	Application Programming 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
GMF	Graphical Modelling Framework
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	Hypertext 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 15 Acknowledgements

Part of this work has been carried out in the scope of Master Students Thesis that has been contributed to the AspireRFID open source software. More specifically the following students have partially worked for the Implementations:

Yongming Luo - AIT	Business Process Workflow Management Editor (BPWME) Introduction Section 10)
Karageorgiou Eleftherios	Business Process Workflow Management Editor (BPWME) Introduction Section 10)

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 D3.4b, June 2010, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables

[22] Nikos Kefalakis, John Soldatos, Efstathios Mertikas and Neeli R. Prasad, "Generating Business Events in an RFID Network", submitted to IEEE International Conference on RFID - Technologies and Applications 2011.

[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, et. al., "Programmable RFID Solutions Specification", ASPIRE Project Public Deliverable D4.4b, June 2010, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables

[25] "Eclipse Graphical Modelling 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

[28] Jan Holmström, Mikko Ketokivi and Ari-Pekka Hameri "Bridging Practice and Theory: a Design Science Approach" Decision Sciences 40 (1), 2009.

[29] Creswell, J., "Research design: Qualitative, quantitative, and mixed methods approaches", Sage Pubns, 2008.

[30] Hevner, A., March, S., Park, J. & Ram, S., "Design Science in Information Systems Research", Management information systems quarterly 28(1), 2004, pp. 75–106.

[31] Jackson, M., "Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices", Addison Wesley, 1995.

[32] Rhea Wessel, "Staff Jeans to Introduce RFID-enabled Customer Services", RFID Journal, Oct 2010.

[33] Rifidi Project, http://www.rifidi.org/

[34] Nuseibeh, B. and Easterbrook, S., "Requirements engineering: a roadmap", Int'l Conf. on Software Engineering, Ireland, 2000.

[35] BEAWebLogic RFID Enterprise Server™, "Understanding the Event, Master Data, and Data Exchange Services", Version 2.0, Revised: October 12, 2006.

[36] N. Kefalakis, J. Soldatos, N. Konstantinou, N. Prasad: APDL: A Reference XML Schema for Process-centered Definition of RFID Solutions, In Int J. of Systems and Software (JSS), Elsevier, 2011 (doi:10.1016/j.jss.2011.02.036).

[37] Nektarios Leontiadis, Nikos Kefalakis, John Soldatos, "Bridging RFID Systems and Enterprise Applications through Virtualized Connectors", International Journal of Automated Identification Technology (IJAIT), Vol. 1, No.2, 2010.

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="EPCglobal-ale-1_1-alelr.xsd"></xs:import>
        <xs:import namespace="urn:epcglobal:ale:xsd:1"
                schemaLocation="EPCglobal-ale-1_1-ale.xsd"></xs:import>
        <xs:import namespace="urn:epcglobal:epcis-masterdata:xsd:1"
                schemaLocation="EPCglobal-epcis-masterdata-1_0.xsd"></xs:import>
        <xs:import namespace="http://www.wfmc.org/2002/XPDL1.0"
                schemaLocation="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 minOccurs="0" maxOccurs="1"
                                ref="epcismd:EPCISMasterDataDocument" />
                        <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 maxOccurs="1" ref="ale:ECSpec" />
                                <xs:element maxOccurs="1"
                                        ref="epcismd:EPCISMasterDataDocument" />
                                <xs:element maxOccurs="1" 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

Register APDL Example

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

<apdl:OLCBProc id="urn:epcglobal:fmcg:bti:openloopsupplychain"
  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
../aspireRfidSpecificationLanguage/AspireSpesificationLanguage.xsd ">

  <epcismd:EPCISMasterDataDocument>
    <EPCISBody>
      <VocabularyList>
        <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessStep">
          <VocabularyElementList>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:receiving">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="receiving" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:picking">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Picking" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:shipping">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="shipping" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:shipment">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Shipment" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:production">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Production" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:accepting">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Accepting" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:inspecting">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Inspecting" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:storing">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Storing" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:packing">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Packing" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:loading">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Loading" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:commissioning">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Commissioning" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:decommissioning">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Decommissioning" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:bizstep:destroying">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Destroying" />
            </VocabularyElement>
          </VocabularyElementList>
        </Vocabulary>

        <Vocabulary type="urn:epcglobal:epcis:vtype:Disposition">
          <VocabularyElementList>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:active">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Active" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:inactive">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Inactive" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:reserved">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Reserved" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:encoded">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Encoded" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:in_transit">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="In_transit" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:non_sellable">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Non_sellable" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:in_progress">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="In_progress" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:disp:sold">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Sold" />
            </VocabularyElement>
          </VocabularyElementList>
        </Vocabulary>

        <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessTransactionType">
          <VocabularyElementList>
            <VocabularyElement id="urn:epcglobal:fmcg:btt:shipping">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Shipping" />
            </VocabularyElement>
            <VocabularyElement id="urn:epcglobal:fmcg:btt:receiving">
              <attribute id="urn:epcglobal:epcis:mda:Name" value="Receiving" />
            </VocabularyElement>
          </VocabularyElementList>
        </Vocabulary>
      </VocabularyList>
    </EPCISBody>
  </epcismd:EPCISMasterDataDocument>


  <!--
    Open Loop Composite Business Process (AspireRFID Process Description
    Language Specification)
  -->

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


    <epcismd:EPCISMasterDataDocument>
      <EPCISBody>
        <VocabularyList>
          <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessLocation">
            <VocabularyElementList>
              <VocabularyElement id="urn:epcglobal:fmcg:loc:greece:pireus:mainacme">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="Acme" />
                <attribute id="urn:epcglobal:epcis:mda:Address" value="Akadimias 3" />
                <attribute id="urn:epcglobal:epcis:mda:City" value="Pireus" />
                <attribute id="urn:epcglobal:epcis:mda:Country" value="Greece" />
              </VocabularyElement>

              <VocabularyElement
id="urn:epcglobal:fmcg:loc:greece:pireus:mainacme,urn:epcglobal:fmcg:loc:acme:warehouse1">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="AcmeWarehouse1" />
                <attribute id="urn:epcglobal:epcis:mda:Read Point"
                  value="urn:epcglobal:fmcg:loc:rp:45632.Warehouse1DocDoor" />
              </VocabularyElement>

              <VocabularyElement
```

```
id="urn:epcglobal:fmcg:loc:greece:pireus:mainacme,urn:epcglobal:fmcg:loc:acme:warehouse2">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="AcmeWarehouse2" />
                <attribute id="urn:epcglobal:epcis:mda:Read Point"
                  value="urn:epcglobal:fmcg:loc:rp:06141.Warehouse2DocDoor" />
              </VocabularyElement>

              <VocabularyElement

id="urn:epcglobal:fmcg:loc:greece:pireus:mainacme,urn:epcglobal:fmcg:loc:acme:warehouse3">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="AcmeWarehouse3" />
                <attribute id="urn:epcglobal:epcis:mda:Read Point"
                  value="urn:epcglobal:fmcg:loc:rp:56712.Warehouse3Docdoor" />
              </VocabularyElement>
            </VocabularyElementList>
          </Vocabulary>

          <Vocabulary type="urn:epcglobal:epcis:vtype:ReadPoint">
            <VocabularyElementList>
              <VocabularyElement
                id="urn:epcglobal:fmcg:loc:rp:45632.Warehouse1DocDoor">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="Warehouse1DocDoor" />
              </VocabularyElement>

              <VocabularyElement
                id="urn:epcglobal:fmcg:loc:rp:06141.Warehouse2DocDoor">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="Warehouse2DocDoor" />
              </VocabularyElement>

              <VocabularyElement
                id="urn:epcglobal:fmcg:loc:rp:56712.Warehouse3Docdoor">
                <attribute id="urn:epcglobal:epcis:mda:Name" value="Warehouse3DocDoor" />
              </VocabularyElement>
            </VocabularyElementList>
          </Vocabulary>

        </VocabularyList>
      </EPCISBody>
    </epcismd:EPCISMasterDataDocument>

    <apdl:EBProc id="urn:epcglobal:fmcg:bte:acmewarehouse1receive"
      name="Warehouse1DocDoorReceive">
      <!--
        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: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:ExtendedAttribute Name="BegEngineEndpoint"
          Value="http://localhost:8080/aspireRfidBEG/begengine" />


        <!-- The DefinedECSpecName can be collected from the EBProc id-->
        <!--
          For the BEG configuration the port can be collected from the
          "ECSpecSubscriptionURI" value and the event to serve from the EBPSpec
          id
        -->
      </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"
                        value="Warehouse1DocDoorReceive" />
                      <!--
                        For the required ECReportID we will use the EBPSpec id
                        and the information for which kind of reports BEG will
                        use the event type will provide them.
                      -->
                      <attribute id="urn:epcglobal:epcis:mda:event_type"
                        value="ObjectEvent" />
                      <attribute id="urn:epcglobal:epcis:mda:business_step"
                        value="urn:epcglobal:fmcg:bizstep:receiving" />
                      <attribute id="urn:epcglobal:epcis:mda:business_location"
                        value="urn:epcglobal:fmcg:loc:acme:warehouse1" />
                      <attribute id="urn:epcglobal:epcis:mda:disposition"
                        value="urn:epcglobal:fmcg:disp:in_progress" />
                      <attribute id="urn:epcglobal:epcis:mda:read_point"
                        value="urn:epcglobal:fmcg:loc:45632.Warehouse1DocDoor" />
                      <attribute id="urn:epcglobal:epcis:mda:transaction_type"
                        value="urn:epcglobal:fmcg:btt:receiving" />
                      <attribute id="urn:epcglobal:epcis:mda:action"
                        value="ADD" />
                    </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">5500</repeatPeriod>
              <duration unit="MS">5500</duration>
              <stableSetInterval unit="MS">0</stableSetInterval>
            </boundarySpec>
            <reportSpecs>
              <!--For the required ECReportID we will use the EBPSpec id
-->
              <reportSpec reportOnlyOnChange="false" reportName="bizTransactionIDs"
                reportIfEmpty="true">
                <reportSet set="CURRENT" />
                <filterSpec>
                  <includePatterns>
                    <includePattern>urn:epc:pat:gid-96:145.12.*</includePattern>
                    <includePattern>urn:epc:pat:gid-96:239.30.*</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>
                    <includePattern>urn:epc:pat:gid-96:1.4.*</includePattern>
                    <includePattern>urn:epc:pat:gid-96:1.3.*</includePattern>
                  </includePatterns>
                  <excludePatterns />
                </filterSpec>
                <groupSpec />
                <output includeTag="true" includeRawHex="true"
                  includeRawDecimal="true" includeEPC="true" includeCount="true" />
              </reportSpec>
            </reportSpecs>
            <extension />
          </ale:ECSpec>
        </apdl:DataField>

        <!--
          We could have many LRSpecs defining many Logical Readers for one
          EBProc
        -->
        <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>4000</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 “olcbproccontrol” Soap Interface

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

  @WebMethod()
  @WebResult(name = "registerStatus")
  public HashMap<String, String> register(@WebParam(name = "openLoopCBProc") OLCBProc openLoopCBProc)
                        throws OLCBProcValidationException, NotCompletedExecutionException;

  @WebMethod()
  @WebResult(name = "unregisterStatus")
  public HashMap<String, String> unregister(@WebParam(name = "openLoopCBProc") OLCBProc
openLoopCBProc)
                        throws NoSuchOLCBProcIdException;

  @WebMethod()
  @WebResult(name = "updateStatus")
  public HashMap<String, String> update(@WebParam(name = "openLoopCBProc") OLCBProc openLoopCBProc)
                        throws OLCBProcValidationException, NotCompletedExecutionException;

  @WebMethod()
  @WebResult(name = "startStatus")
  public HashMap<String, String> start(@WebParam(name = "openLoopCBProc") OLCBProc openLoopCBProc)
                        throws NoSuchOLCBProcIdException;

  @WebMethod()
  @WebResult(name = "stopStatus")
  public HashMap<String, String> stop(@WebParam(name = "openLoopCBProc") OLCBProc openLoopCBProc)
                        throws NoSuchOLCBProcIdException;

  @WebMethod()
  @WebResult(name = "OLCBProc")
  public OLCBProc getOLCBProc(@WebParam(name = "openLoopCBProcID") String openLoopCBProcID,
              @WebParam(name = "endPoints") HashMap<String, String> endPoints)
                        throws NoSuchOLCBProcIdException;

}
```

PE “clcbproccontrol” Soap Interface

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

  @WebMethod()
  @WebResult(name = "registerStatus")
  public HashMap<String, String> register(@WebParam(name = "closeLoopCBProc") CLCBProc
closeLoopCBProc)
                        throws CLCBProcValidationException, NotCompletedExecutionException;

  @WebMethod()
  @WebResult(name = "unregisterStatus")
  public HashMap<String, String> unregister(@WebParam(name = "closeLoopCBProc") CLCBProc
closeLoopCBProc)
                        throws NoSuchCLCBProcIdException;

  @WebMethod()
  @WebResult(name = "updateStatus")
  public HashMap<String, String> update(@WebParam(name = "closeLoopCBProc") CLCBProc closeLoopCBProc)
```

```
                        throws CLCBProcValidationException, NotCompletedExecutionException;

  @WebMethod()
  @WebResult(name = "startStatus")
  public HashMap<String, String> start(@WebParam(name = "closeLoopCBProc") CLCBProc closeLoopCBProc)
                        throws NoSuchCLCBProcIdException;

  @WebMethod()
  @WebResult(name = "stopStatus")
  public HashMap<String, String> stop(@WebParam(name = "closeLoopCBProc") CLCBProc closeLoopCBProc)
                        throws NoSuchCLCBProcIdException;

  @WebMethod()
  @WebResult(name = "CLCBProc")
  public CLCBProc getCLCBProc(@WebParam(name = "closeLoopCBProcID") String closeLoopCBProcID,
               @WebParam(name = "endPoints") HashMap<String, String> endPoints)
                        throws NoSuchCLCBProcIdException;

}
```

PE “ebproccontrol” Soap Interface

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

  @WebMethod()
  @WebResult(name = "registerStatus")
  public HashMap<String, String> register(@WebParam(name = "elementaryBProc") EBProc elementaryBProc)
                        throws EBProcValidationException, NotCompletedExecutionException;

  @WebMethod()
  @WebResult(name = "unregisterStatus")
  public HashMap<String, String> unregister(@WebParam(name = "elementaryBProc") EBProc
elementaryBProc)
                        throws NoSuchEBProcIdException;

  @WebMethod()
  @WebResult(name = "updateStatus")
  public HashMap<String, String> update(@WebParam(name = "elementaryBProc") EBProc elementaryBProc)
                        throws EBProcValidationException, NotCompletedExecutionException;

  @WebMethod()
  @WebResult(name = "startStatus")
  public HashMap<String, String> start(@WebParam(name = "elementaryBProc") EBProc elementaryBProc)
                        throws NoSuchEBProcIdException;

  @WebMethod()
  @WebResult(name = "stopStatus")
  public HashMap<String, String> stop(@WebParam(name = "elementaryBProc") EBProc elementaryBProc)
                        throws NoSuchEBProcIdException;

  @WebMethod()
  @WebResult(name = "EBProc")
  public EBProc getEBProc(@WebParam(name = "elementaryBProcID") String elementaryBProcID,
               @WebParam(name = "endPoints") HashMap<String, String> endPoints)
                        throws NoSuchEBProcIdException;

}
```