APDL: A reference XML Schema for process-centered definition of RFID solutions

Nikos Kefalakis, John Soldatos and Nikolaos Konstantinou
Athens Information Technology, 0,8km Markopoulo Ave., P.O. Box 68, GR-19002 PEANIA, GREECE

E-mail: { nkef@ait.edu.gr, jsol@ait.edu.gr and nkons@ait.edu.gr}

Phone: {+302106682755, +302106682759, +302106682763}

Fax: {+302106682703}

Neeli R. Prasad

Center for TeleInFrastruktur (CTIF), Aalborg University, Fredrik Bajers Vej 7, 202 9220 Aalborg East, Denmark

E-mail: {np@es.aau.dk}

Phone: {+45 9940 9835}

Fax: {+45 9815 1583}

Abstract

Despite the proliferation of RFID systems and applications, there is still no easy way to develop, integrate and deploy non-trivial RFID solutions. Indeed, the latter comprise various middleware modules (e.g., data collection and filtering, generation of business events, integration with enterprise applications), which must be deployed and configured independently. In this paper we introduce APDL (AspireRFID Process Description Language), an XML based specification for describing and configuring RFID solutions. Using APDL one can minimize the steps and effort required to integrate and configure an RFID solution, since it unifies all the configuration parameters and steps comprising an RFID deployment. APDL supports several configuration parameters defined in the scope of the EPCglobal architecture and related standards. However, it extends beyond the EPCglobal architecture, to a wider class of RFID solutions. Furthermore, APDL is amendable by visual tools, which obviates the need to carry out low-level programming tasks in order to deploy an RFID solution. These tools are also presented and evaluated in the paper.

Keywords

RFID Middleware, RFID Business Processes, RFID Business Events, Master Data, RFID for Logistics, RFID in Supply Chain Management

1. Introduction

We are currently witnessing a proliferation of RFID (Radio Frequency Identification) applications, in a wide range of fields including logistics, trade and industry. These applications are the first step towards the realization of long awaited visions such as pervasive computing [1], machine-to-machine communications [2], as well as the Internet-of-Things (IoT) [3]. While RFID technology is based on simple operational principles, the complete design and implementation of any non-trivial RFID solution is still a very arduous and resource consuming task. Indeed, non-trivial RFID applications comprise typically multiple readers and tags, as well as multiple consuming applications in a highly heterogeneous landscape [4]. In this landscape, different tag information streams have to be routed across different business applications, according to sets of complex business rules. Given this complexity, the development of RFID solutions is nowadays facilitated by middleware infrastructures, which undertake to interface to heterogeneous readers, filter the tag streams, generate application specific events, and eventually route these events to the appropriate business applications. But this complexity, due to the need of specialized in RFID systems experts, increases the cost buriers and the TCO (Total Cost of Ownership) in Enterprises and especially to Small and Medium Enterprises (SME) sometimes the RFID technology becomes a “luxury” that they cannot afford.

A variety of both proprietary (e.g., [5], [6]) and standard-based middleware infrastructures for RFID systems have recently emerged (e.g., [4]). The most prominent effort in the area of standardized middleware infrastructure is the architectural framework specified by EPCglobal [7], which specifies several standards [8] and related middleware building blocks for building RFID systems for logistics and warehouse management. The EPCglobal core middleware solution, which is capable to connect an RFID reader up to an enterprise system, is consisted from three layers, which are depicted in Figure 1, and are:

- The Filtering and Collection layer (F&C)[20], which is responsible for collecting tag streams from the physical RFID readers and based on the users configurations produce the equivalent reports,
- The Capturing layer, which is responsible for adding business context to the received tag reports from the F&C layer, and
- The EPCIS (Electronic Product Code Information Sharing) [19] repository, which is responsible for storing the companies Master Data and the RFID Events that are produced from the Capturing Layer.

However, even with such middleware frameworks at hand, RFID solution developers and integrators have to allocate significant effort in the process of configuring and integrating the various middleware building blocks. The practical implication in this fact is that RFID applications developers must still engage with low-level programming tasks in order to successfully build, configure and deploy an RFID solution [9]. Most of this effort relates to the need for configuring the middleware building blocks of the application [10], while at the same time ensuring the interworking and interoperability of these building blocks. Taking as example the EPCglobal framework described above it requires:

- Two (2) configuration files at the F&C layer. One for defining the physical RFID reader and one for defining the filtering/grouping pattern that is going to be used for producing the tag report.
- Moreover the capturing layer needs to be developed based on the business process that is going to serve and finally
- The EPCIS repository should contain the required Company’s Master Data (e.g. read points, item dispositions, business locations, business transaction description etch).

All of these configurations from their nature are pretty generic and requires, from a developer, deep understanding of the EPCglobal’s different specifications and middleware’s capabilities, so as to successfully configure one to serve even a relatively simple Business process (e.g. receiving).

The above-mentioned problems and challenges stem from the lack of a standard way for specifying an integrated RFID solution. Despite the emergence of several middleware standards, there is no easy and accepted way for specifying the middleware elements of an RFID solution. Furthermore, there are no formal proposals for integrating the standard middleware blocks of an RFID solution. Indeed, a single specification for describing an RFID solution could greatly alleviate the task of specifying, configuring and implementing an RFID solution. This is because a single RFID solution specification could obviate the need for configuring and integrating multiple middleware blocks, based on a variety of specifications and XML configuration files, each one pertaining to a specific middleware block.

Motivated by the above challenges, this paper introduces the specification of a meta-language for describing and configuring RFID solutions, and illustrates how it eases the configuration and deployment of non-trivial RFID solutions. The specification is defined as an XML-based domain specific language for describing RFID-based processes and is conveniently called APDL (AspireRFID Process Description Language), since it was developed in the framework of the European project ASPIRE[1]. APDL has been carefully designed to be simple, intuitive and

[1] ASPIRE is funded by the European Commission under contract FP7-215417

comprehensive, which facilitates its use by the majority of RFID developers, integrators and consultants, thus reducing the Total Cost of Ownership (TCO) of an RFID solution. APDL is not a general purpose specification; it is rather a specialized specification, which captures the data and semantics of RFID processes. Nevertheless, APDL adopts several concepts of the general-purpose XPDL [17] (XML Process Definition Language) specification, especially in terms of modeling composite multi-step inter-enterprise business processes. The (re)use of XPDL constructs boosts the scalability, extensibility and technological longevity of the APDL language. Furthermore, APDL is amenable by tools, in order to enable stakeholders to use graphical modeling environment for designing and deploying RFID solutions.

APDL has a clear orientation towards logistics solutions that comply with the EPC global architecture. This is because it leverages several EPCglobal middleware specifications, and combines them towards integrated RFID solutions. However, it has also the flexibility to describe/represent smaller scale solutions that make use of smaller subsets of EPCglobal middleware building blocks. Furthermore, APDL can support non-EPC solutions based on appropriate customization of its underlying middleware components. It is important that APDL based solutions can be designed and deployed using visual tools for modeling, deployment and configuration tasks. These tools are built taking into account the APDL constructs, based on a Model Driven Architecture (MDA) paradigm. Hence, in addition to the APDL capabilities, the paper presents tools and techniques enabling RFID application development and deployment. We also discuss that APDL could potentially evolve to an open XML specification for describing RFID solutions, with a view to standardizing the modeling of any RFID solution.

The structure of the paper is as follows: Following this introductory section, section 2 describes the methodology used to build the APDL schema and its requirements. Section 3 reviews related work in terms of RFID middleware configuration, process description specifications and domain specific languages. Accordingly, section 4 introduces the main characteristics of the APDL language. It also discusses the EPCglobal specifications, which are exploited by APDL. Section 5 describes the tags and constructs of the language at a finer level of granularity, while also listing some illustrating examples about the use of the constructs. Section 6 discusses a set of tools that are associated with the APDL language. Section 7 provides examples of RFID solutions, which are described using APDL, while section 8 concludes the paper.

2. Methodology

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

- The one was the design science methodology [26], 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 [26]. 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 [27], [28].
- The second methodology that was used is requirement engineering. The major activities in requirements engineering include elicitation, modeling and analysis, communication, agreement, evolution and Integration [25].

Therefore, the requirements of the APDL 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 [24].
- The feedback collected from the developers and Users of the AspireRFID [21] 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[4], Rifidi[31]) and commercial ones (e.g. BEA WebLogic RFID Enterprise Server[22]) 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 BPWME (Business Process Workflow Management Editor) and the PE (Programmable Engine), where APDL plays the role of a "linchpin".
- And finally the feedback acquired and analyzed from AspireRFID middleware pilot deployments (e.g. [30]) and the mapping of the trial processes into the APDL language.

By combining all the information gathered from the various inputs described above we came up with the following list of requirements of the APDL language. So essentially, APDL should be:

- As simple as possible,
- Domain-Oriented,
- Able to support RFID processes and Data,
- Capable of carrying all the required data needed to configure an EPC compliant RFID middleware,
- Capable of describing a complete Open Loop RFID enabled Supply Chain Scenario (e.g. Figure 3),
- Able of carrying workflow graphical representation data (e.g., XPDL),
- XML Based,
- Capable of becoming an Open Specification for EPC based RFID Solutions
- Standard and extensible, and finally
- to allow stakeholders to build RFID solutions

3. *Related Work: Domain Specific Language and Language Oriented Programming*

Domain-specific languages (DSL's) became increasingly popular in the latest years, since they have the ability to capture powerful abstractions of well-known application domains. The concept is not new; DSL's have made their appearance since the early computer days for a broad domain of technologies. Well known DSL's include TeX for documents, Microsoft Excel macros for spreadsheets, SQL for relational databases, etc. The key idea is that by appropriately establishing domain specific notations, one can increase productivity of the target programmer audience. The concept of applying a DSL-based solution in order to solve specific problems or even developing a DSL that describes a problem context is generally known as Language Oriented Programming [14].

A specific category of DSL's includes the *process description languages* that are developed in order to model workflows. RFID-based solutions could benefit by such languages, since they involve non-trivial and often complex processes and workflows. A prominent example of a process description language is BPEL (or WS-BPEL, Web Services Business Process Execution Language), which is an OASIS (Organization for the Advancement of Structured Information Standards) standard for specifying interactions with Web Services. BPEL is the successor to older efforts such as WSFL (Web Services Flow Language) by IBM and XLANG by Microsoft. BPEL can be used to effectively tackle distributed processes in systems communicating via Web Services [12]. Also, BPEL4WS, the successor of BPML (Business Process Modeling Language), is a meta-language for the modeling of business processes, supported by the Business Process Management Initiative (BPMI). There is a wide variety of BPEL engines (e.g., jBPM, OpenLink Virtuoso, Microsoft BizTalk etc.).

XPDL (XML Process Definition Language) [17] is a standard formalized by the Workflow Management Coalition (WfMC). XPDL is not an executable language like BPEL but a process design format that models the process definitions while in the same time it preserves graphical information (XY coordinates). With XPDL, a product can write out a process definition with full fidelity, and another product can read it in and reproduce the same diagram that was sent [15]. XPDL is supported by numerous commercial and open-source implementations. Other workflow languages include YAWL (Yet Another Workflow Language) [13] and, WS-CDL (Web Services

Choreography Description Language) [16] and it is the successor of WSCI (Web Service Choreography Interface). All of the above languages are based on XML structure.

In order to model the workflow in human-understandable form, several notation languages and schemes have been developed, including UML (Unified Modeling Language) activity diagrams, Event-Driven Process Chain (EPC's), or Petri Nets. The most prominent approach seems to be BPMN (Business Process Modeling Notation), a notation for business processes supported by the OMG (Object Management Group) and the BPMI. The language provides the ability of understanding business procedures through a graphical notation in a standard manner. BPMN can be used to model BPEL processes [11].

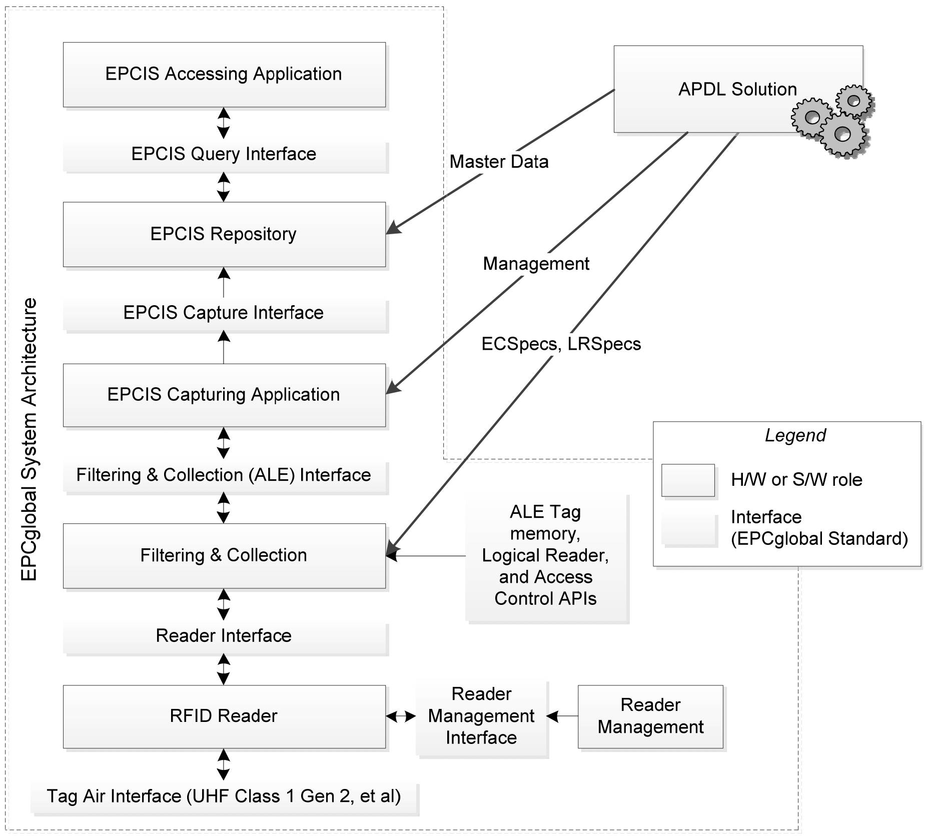
The workflow supporting languages, that were presented, are rather general purpose as they have been designed to model a variety of workflow environments capturing thus most of the well known business processes. RFID solutions can be modeled as complex workflow processes including business processes definitions and numerous configuration specifications targeting all the logical layers of RFID middleware. The languages presented above, are powerful, yet by design not tuned to any specialized domain. Hence, they do not take into account the nature of RFID solutions and fail to capture concepts like RFID-based processes and RFID middleware data.

From the aforementioned languages, XPDL would probably be the most appropriate candidate to use taking in considerations the RFID language requirements mentioned above. But following the same requirements due to the generality and complexity of XPDL for describing an RFID enabled Supply Chain scenario we were pushed toward creating a new hybrid that would be simpler to understand, to describe its structure and specialized on RFID Business Processes and Data (such as APDL).

4. The APDL Business Process Description Framework

APDL is oriented towards solutions that comply with the EPCglobal Architecture. According to this architecture [7] the modules that compose an end-to-end RFID solution can be logically considered to be layered as depicted in Figure 1. The typical information flow through these layers involves:

- Collecting RFID data from the physical readers, through reading the tagged items. At this level middleware implementations insulate higher layers from knowing what readers have been chosen. Moreover, they achieve virtualization of tags, which allows RFID applications to support different tag formats.
- Filtering the RFID sensor streams according to application needs, and accordingly emitting application level events. At this level middleware implementations insulate the higher layers from the physical design choices on how tags are sensed and accumulated, and how the time boundaries of events are triggered.
- Mapping the filtered readings to business semantics as required by the target applications and business processes. At this level middleware implementations insulate enterprise applications from understanding the details of how individual steps in a business process are carried out.



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 [7]

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

- Defining the Event Cycle Specifications (ECSpecs) [20],
- Defining the Logical Reader Specifications (LRSpecs) [20],
- And, finally, providing the EPCIS (Electronic Product Code Information Sharing) [19] with the required Master Data (EPCIS Master Data Document) that partially manages how Application Level Events (ALE) [20], will be stored in the EPCIS repository.

4.1.The Required Components/Layers

The APDL challenge is to create a single specification which will be able to describe a complete RFID Business Process in a coherent way that combines all the above specifications, while also accounting for Middleware configuration data (e.g. modules connection endpoints), and Workflow information. 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) [20] module one must configure the ECSpec, which is a complex type that describes an Event Cycle [20] 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.

At the **EPCIS** [19] layer Master Data Vocabularies are defined. The Master Data Vocabularies contain additional data that provides the necessary context for interpreting Event Data [19] . 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 [19]	urn:epcglobal:epcis:mda:event_name

EventType [19]	urn:epcglobal:epcis:mda:event_type
BusinessStep [19]	urn:epcglobal:epcis:mda:business_step
BusinessLocation [19]	urn:epcglobal:epcis:mda:business_location
Disposition [19]	urn:epcglobal:epcis:mda:disposition
ReadPoint [19]	urn:epcglobal:epcis:mda:read_point
TransactionType [19]	urn:epcglobal:epcis:mda:transaction_type
Action [19]	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). 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 [20]) 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 [19].

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).

4.2. Defining APDL's Business Process

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.

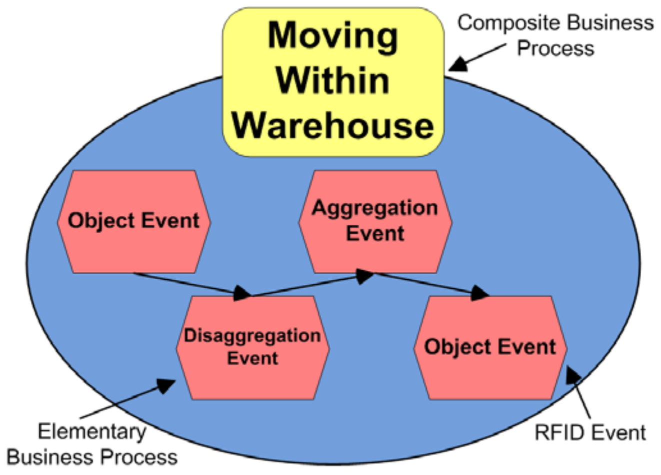


Figure 2. Decomposing an Inter-enterprise Business process.

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 EBProc’s 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.

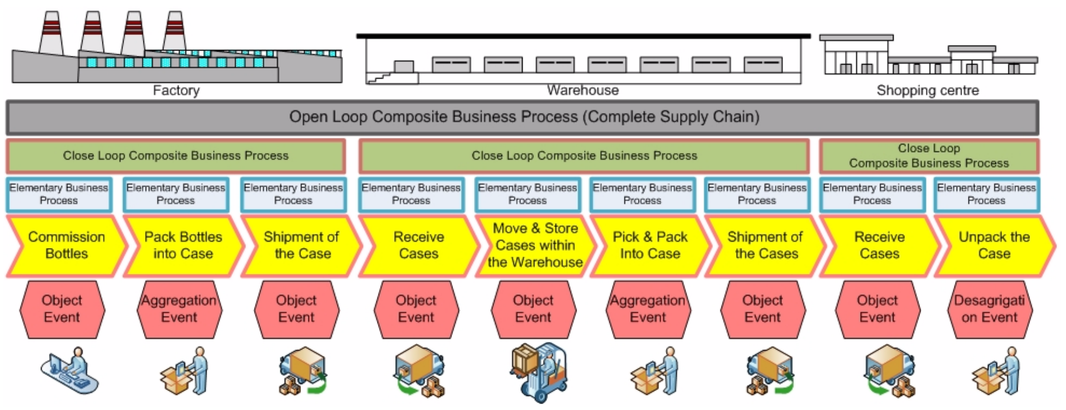


Figure 3. Complete supply chain scenario example.

4.3. 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**	**QuantityEvent**	**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) [22][19]

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 generationg 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 belongs 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 [20] 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 Binding being 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 [20].
- 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 [20].
- 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 [17] so as to describe the processes workflow and to achieve the visualization of the RFID solution.

5. The AspireRFID Business Process Description Language

From an implementation perspective, 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/>`).

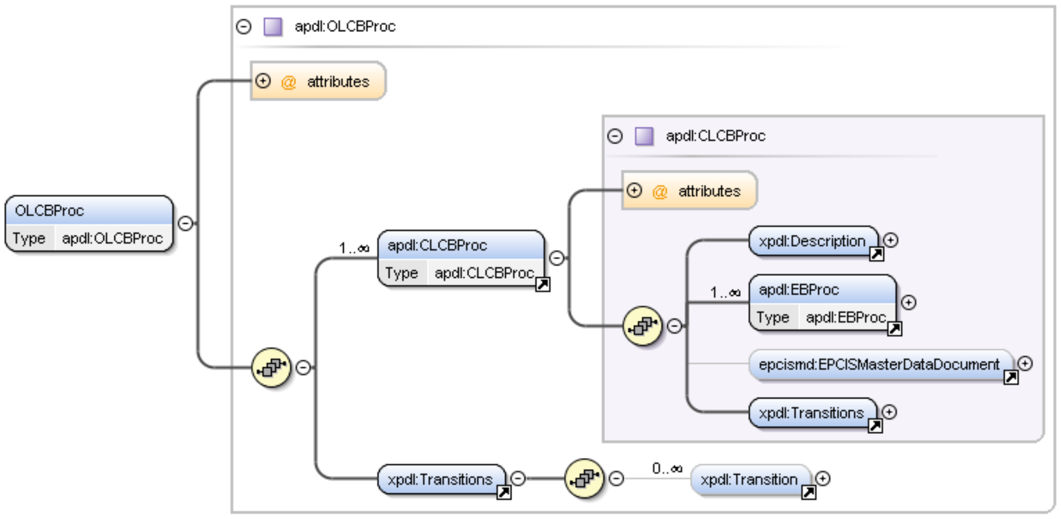


Figure 4 APDL' Schema design (OLCBProc)

"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 [17].

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/>`) [19] 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 [17]. 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.

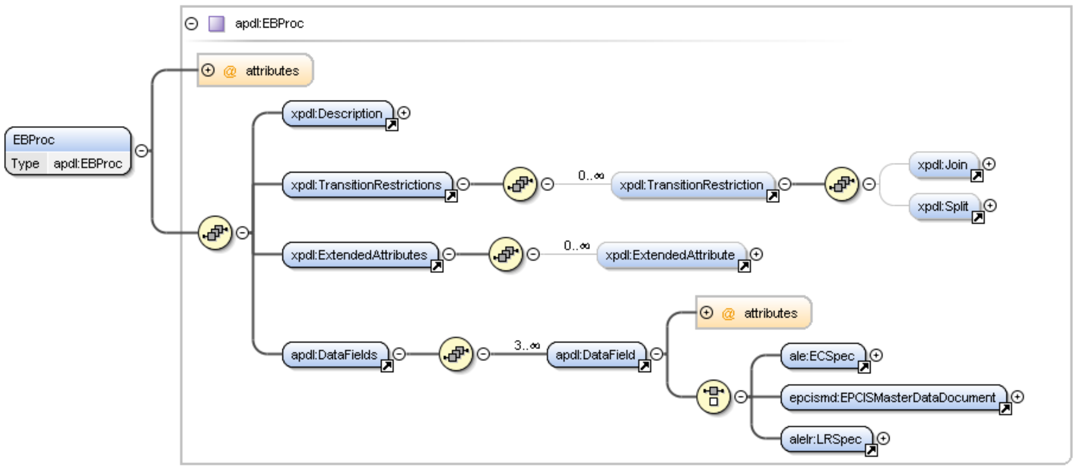


Figure 5 APDL's Schema design (EBProc)

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 4.3, can be provided in the form of EPCglobal's specification files. These files includes:

- 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 [21] PE (programmable Engine). And design data, which are required by the AspireRFID's BPWME.

So the EBProc element more specifically contains:

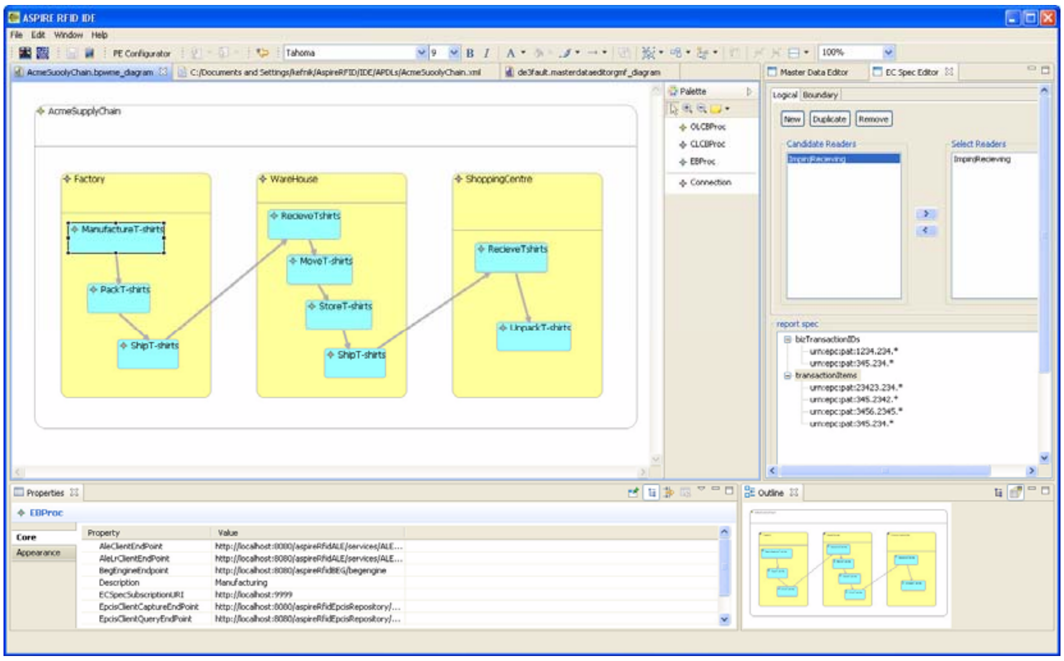
- A set of DataFields (`<apdl:DataFields/>`), that include the required
  - ECSpec (`<ale:ECSpec/>`),
  - LRSpec (`<alelr:LRSpec/>`) and
  - Master Data (`<epcismd:EPCISMasterDataDocument/>`) for describing a specific elementary business process transaction.
- A TransitionRestrictions [17] (`<xpdl:TransitionRestrictions/>`) element, containing a set of TransitionRestriction [17] (`<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.

The complete APDL schema definition can be found in the Appendix.

6. AspireRFID APDL Tools

6.1. Business Process Workflow Management Editor

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 we have designed and prototyped an Eclipse plug-in to enable the visual modeling and configuration of RFID enabled processes. This tool is conveniently called Business Process Workflow Management Editor (BPWME) and is illustrated in Figure 6.



Property	Value
AleClientEndPoint	http://localhost:8080/aspireRfidALE/services/ALE...
AleLrClientEndPoint	http://localhost:8080/aspireRfidALE/services/ALE...
BegEngineEndpoint	http://localhost:8080/aspireRfidBEG/begengine
Description	Manufacturing
ECSpecSubscriptionURI	http://localhost:9999
EpcisClientCaptureEndPoint	http://localhost:8080/aspireRfidEpcisRepository/...
EpcisClientQueryEndPoint	http://localhost:8080/aspireRfidEpcisRepository/...

Figure 6 BPWME Designer Editor snapshot

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 EBProc's.

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 encoding 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.

6.2. AspireRFID Programmable Engine

A middleware layer that facilitates the RFID development and augments the BPWME and APDL's functionality is the Programmable Engine (PE) module.

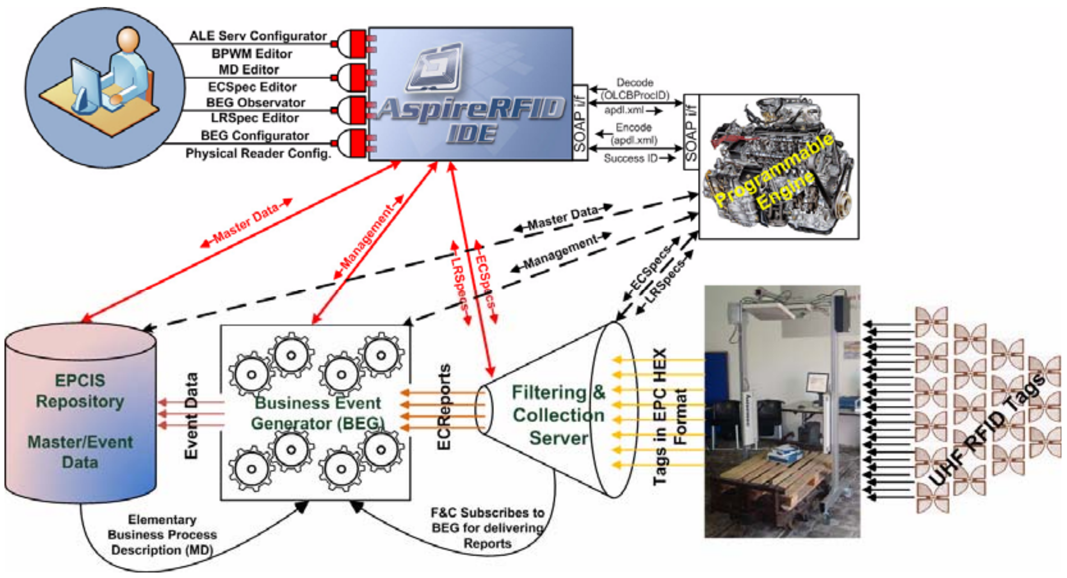


Figure 7 Programmable Engine

6.2.1. *PE Interfaces*

The PE, as shown in Figure 7, bridges APDL with the underlying RFID middleware infrastructure, through "hiding" the lower-level details of the middleware from the APDL developer. PE is a run-time middleware infrastructure that is able to resolve APDL to the number of configuration files, which are required for the deployment of an APDL solution over an RFID middleware infrastructure. Thanks to the APDL and its respective PE, RFID developers are capable of assembling and configuring RFID solutions in a high-level language, by using the BPWME plug-in, in a way that is totally transparent to the low-level middleware libraries (such as those enabling filtering, collection and business event generation).

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

Encode API

The First Interface exposed from a Programmable Engine implementation is the Encode interface. The Encode API, described in Table 5, is consisted from only one method which requires as input an APDL XML document, which contains the hole RFID Business Process description, and returns an integer code which denotes if the execution of this specific command is successful or not. With this method the PE undertakes the task of configuring a running instance of AspireRFID middleware with the Business Processes described to the given OLCBProc Object.

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

Table 5 PE's Encode Interface methods

Decode API

The second Interface exposed from a Programmable Engine implementation is the Decode Interface. The Decode API, described in Table 6, is consisted from only one method "decode" which requires as input a String of an OLCBProc ID which was primarily been used to configure an AspireRFID middleware running instance. This service will be used from clients that want to alter an already encoded Business Process to the AspireRFID middleware by retrieving it ("decode") make the required changes and then reconfiguring back the middleware ("encode").

Method	Argument/Result	Type	Description
decode	openLoopCBProcID	String	This method returns an OLCBProc Object which is retrieved from an AspireRFID middleware running instance by a prior configured (encoded) OLCBProc by giving that object's ID.
	[result]	OLCBProc	

Table 6 PE's Decode Interface methods

6.2.2. How PE Changed the AspireRFID Configuration Process

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

Configuring an RFID Middleware with the Conventional Way
To achieve the configuration of the hole AspireRFID middleware for even a relatively simple scenario, like the one that is described in paragraph 7 below where we define only one EBProc, it would require to follow a few steps and to use a bunch of different AspireRFID "Configurators" (e.g. ECSpec Configurator, LRSpec Configurator and BEG configurator). Figure 8 below illustrates the different steps that an RFID integrator should follow to configure the AspireRFID middleware with the use of the aforementioned tools

More specifically for defining an Elementary Business Process (as shown in Figure 8 below) we should:

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

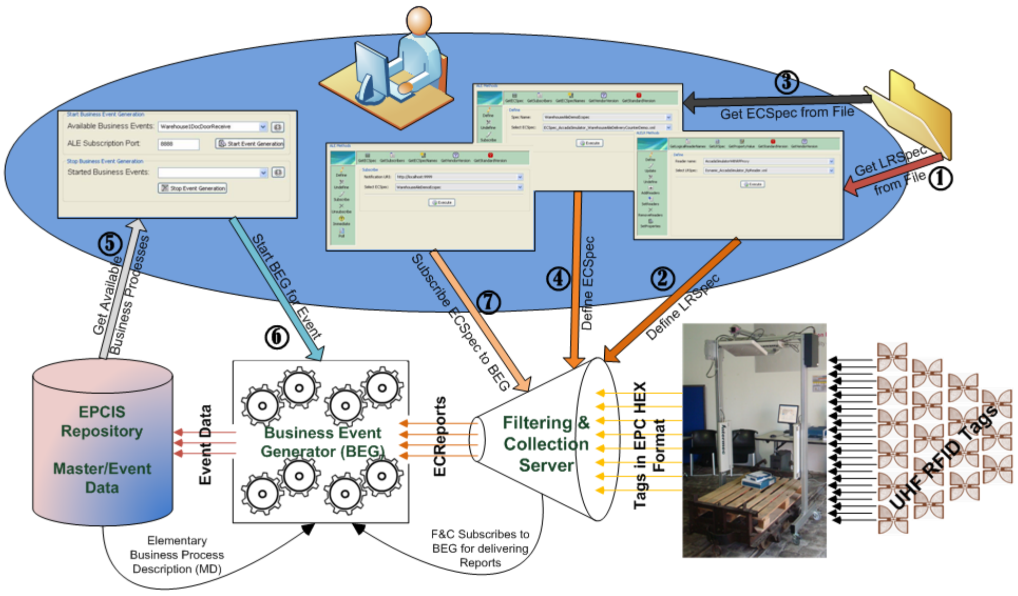


Figure 8 Required AspireRFID Configuring Steps without Programmable Engine

Configuring an RFID Middleware with the Programmable Engine's Way

From the previous section we observe that, to configure the AspireRFID middleware with the conventional way, for just one Elementary Business Process **"7" Steps** where required from the User. In this section we will describe what is required from the User to configure the AspireRFID middleware with the use of Programmable Engine's plug-in (client) again for only one EBProc. It worth's to mention that even if we had to configure the AspireRFID middleware for "N" EBProc's (even for a complete Open Loop supply chain scenario) the steps that the user would have to follow would be the same as the ones described below and would have to follow them only one time.

So assuming that the APDL XML file has already been build for configuring the AspireRFID middleware with the PE's User Client plug-in the first step, as shown in Figure 9, would be to retrieve the "apdl.xml" file from the folder that is stored (Step 1). And the second and final Step would be to use the "Encode" service of the PE's thru the PE Client (Step 2).

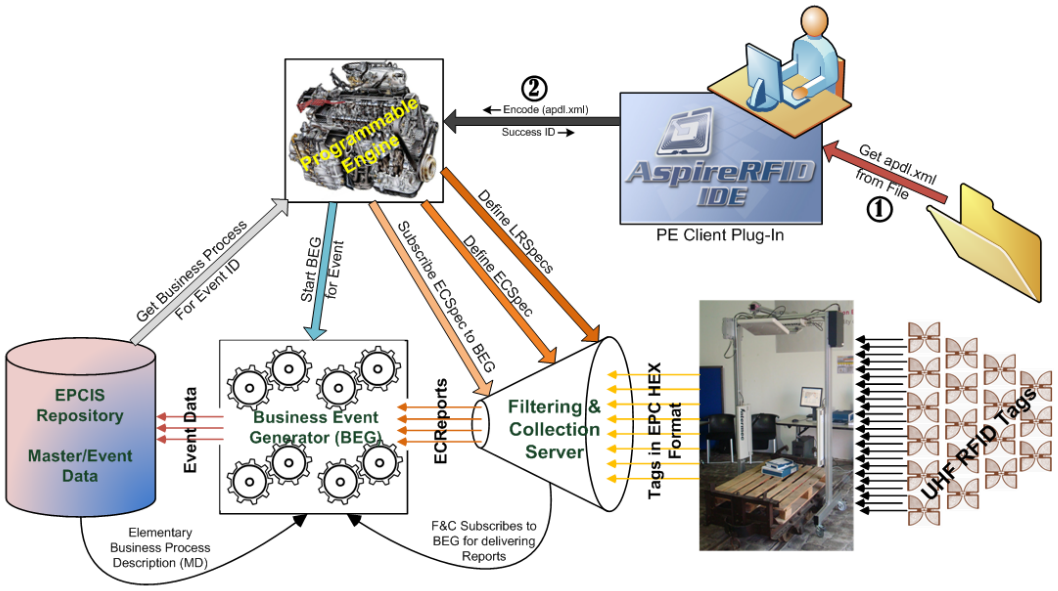


Figure 9 Required AspireRFID Configuring Steps with Programmable Engine

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

- For the conventional way: **7 x "N" Steps**

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

7. Example of RFID Processes and Solutions

In order to demonstrate the presented approach, we illustrate a use-case scenario for a warehouse receiving process. According to the scenario, "Acme", a company that specializes in trading computer parts, places an order to a Microchip Manufacturer for specific CPUs. Acme has several Warehouses, but the goods are meant to be delivered at Warehouse1 and, more specifically, in Section1. It is required that Acme's Warehouse Management System (WMS) is automatically notified upon arrival of the goods. For this reason, Section1 is equipped with an RFID reader and the goods traded are equipped with RFID tags.

Let us now describe the APDL solution for this Business Process. First, we need to define the OLCBProc (as described in Section 5) which will include all the CLCBProcs. Because we are referring to only one Business Location of the "Acme Supply Chain" only one CLCBProc will be required and is shown in Table 7 with ID: "urn:epcglobal:fmcg:bti:acmesupplying". Furthermore, we set up a Master Data Document that describes Acme's Warehouse assets, like Read Points that will be used later from all the EBProc definitions for this specific CLCBProc. Finally, in this CLCBProc definition we can find the "transitions" element which is not used in this example because we are not defining any other CLCBProcs and the EBProc element.

```
<apdl:OLCBProc id="urn:epcglobal:fmcg:bti:openloopsupplychain"
  name="AcmeSupplyChainManagement">
  <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>
              <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>
            </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>
             </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 7 OLCBProc and CLEBProc objects

In order to describe the EBProc shown in Table 10 except defining the “TransitionRestrictions” element, which is empty here because we are not defining any other EBProcs, we are defining the “ExtendedAttributes” element. This element includes the graphical representation of the EBProc object to be used from a graphical editor and the required modules Endpoints to be used from a process engine so as to set up the defined solution.

```
<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 8 EBProc Object

In the excerpt presented above, the “Datafields” element contains a list the basic configuration files (LRSpec, ECSpec and MasterData) of the EBProc definition called “DataField”. First, the ECSpec, in a form as in Table 11 that, in short, configures the ALE layer to produce an ECReport every 4500msec, even if the report is empty (i.e. no tags were found), and because we want to generate an Object Event, referring to Table 3, it contains two report names. For the “bizTransactionIDs” reportSpec we will set the “receiving notes” Class ID’s and for the “transactionItems” reportSpec we will set the “received items” Class ID’s

- So the “receiving notes” Class is:

    - urn:epc:pat:gid-96:145.12.*
- and the “received items” Classes are:
    - urn:epc:pat:gid-96:145.233.*
    - urn:epc:pat:gid-96:145.255.*

```
<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 9 EBProc’s ECSpec DataField

The next DataField is the LRSpec which defines the logical readers that are used to collect the tag data. In Table 12 the dynamic LRSpec definition of an Impinj Speedway LLRP reader is shown where at the configuration time the LRSpec DataField’s name (SmartLabImpinjSpeedwayLogicalReader) will be used as the Logical Reader name which is included also at the ECSpec’s LogicalReader list.

```
<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>
```

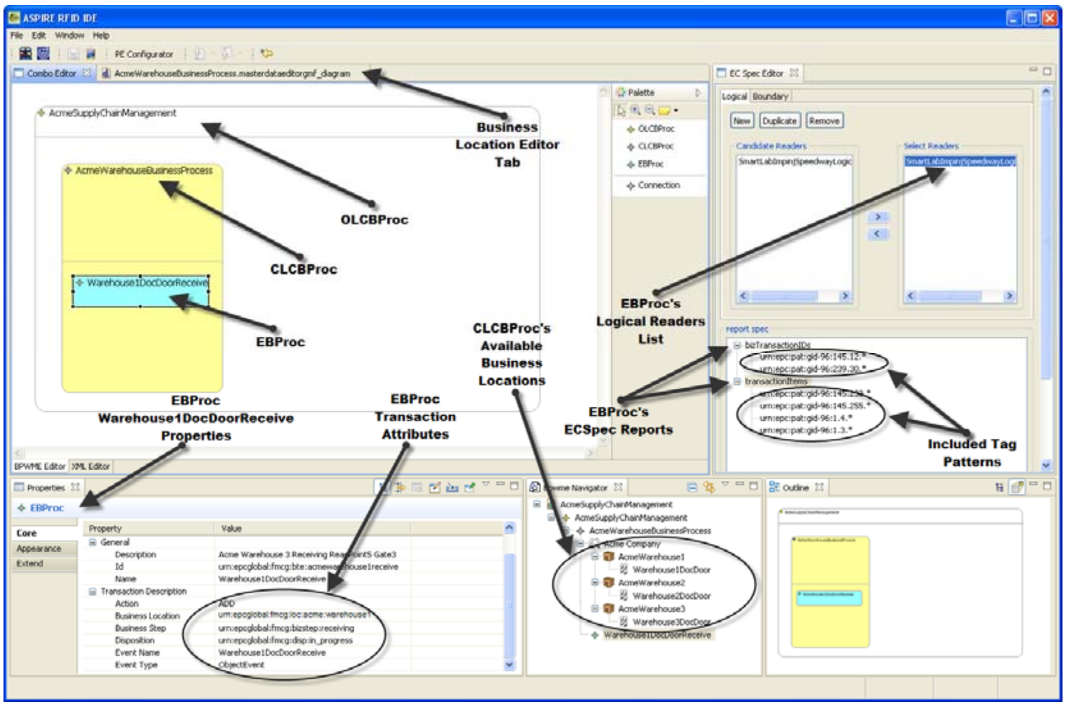
Table 10 EBProc's LRSpec DataField

Next, we define the Master Data document, shown in Table 11 that includes the required information, which are used from the BEG modules, to generate the required RFID Event data. Note that the EPCISMasterDataDocument, as well as the ECSpec and LRSpec comply with the EPCglobal specifications.

```
<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 11 EBProc's Master Data Document DataField

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



report spec
- bizTransactionIDs
  - urn:epc:pat:gid-96:145.12.*
  - urn:epc:pat:gid-96:239.30.*
- transactionItems
  - urn:epc:pat:gid-96:145.25[illegible].*
  - 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

Figure 10 AcmeWarehouse1Recieve Business Process in BPWME plug-in

As we can see in Figure 10 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 EBProc's in it can use them directly and are available at their Transaction Attributes.

If the previous described APDL configuration would be applied to an EPC RFID middleware (e.g. AspireRFID [21]) at runtime it would provide the following solution. ACME gives an order with a specific deliveryID to the Microchip Manufacturer. With this action AspireRFID Connector [23] subscribes to the AspireRFID EPCIS Repository to retrieve events concerning the specific deliveryID.

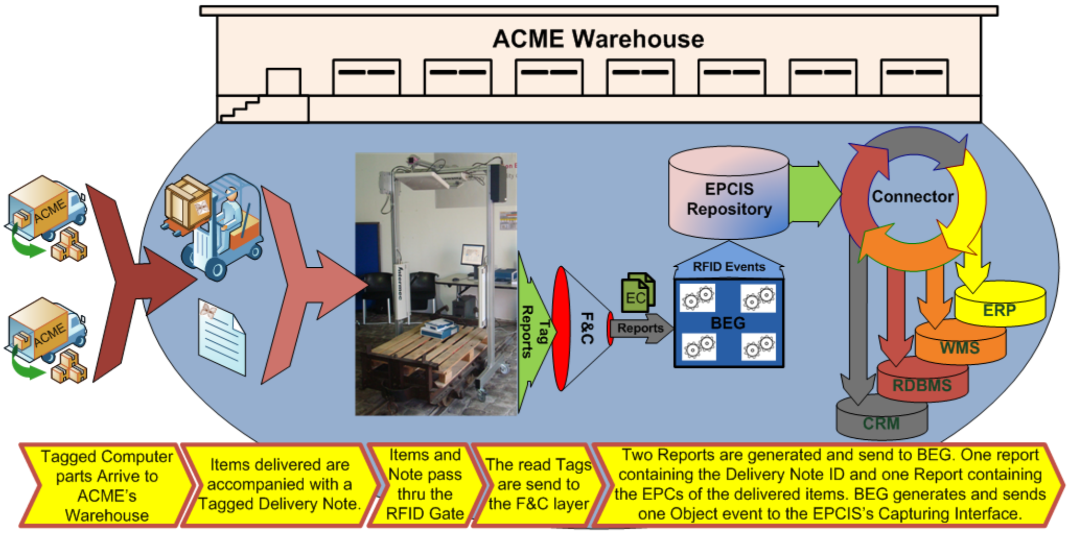

ACME Warehouse
ACME
ACME
EPCIS Repository
Connector
RFID Events
EC
Reports
BEG
Tag Reports
F&C
ERP
WMS
RDBMS
CRM
Tagged Computer parts Arrive to ACME's Warehouse
Items delivered are accompanied with a Tagged Delivery Note.
Items and Note pass thru the RFID Gate
The read Tags are send to the F&C layer
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 11 Acme computer parts Delivery

As visualized in Figure 11 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 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.

The example demonstrates how the different configuration details can be put into action in order to provide concrete solutions to business-process problems. For readability's sake the solution at this example was kept as simple as possible. In real-world scenarios, more business processes and more variables would be involved, leading to more complicated solutions. Note that APDL is used at the deployment of two real world RFID logistics trials organized in the scope of the ASPIRE EC co-funded project (http://www.fp7-aspire.eu).

7.1. Critical Discussion

From the analysis presented so far, the use cases and the respective APDL-based solutions outlined and detailed in this paper we have observed a number of benefits, which most of them has already been discussed, but also limitations that need to be underlined.

- By using the APDL/BPWME combination one is capable to describe the backbone of various RFID Middleware deployments for deferent well defined [24] RFID enabled Supply chain scenarios. This could lead to reusable libraries of RFID solutions, while at the same time enabling the realization of best practices and blueprints.
- APDL is comprehensive for the RFID domain and not for general purpose applications. Through focusing on the needs of RFID applications, it manages to balance simplicity with a comprehensive specification. Furthermore, the fact that APDL is amendable by tools (such as the BPWME) hides any complexity from RFID developers and integrators; the later can rely on visual tools for building a solution.
- APDL in combination with the Programmable Engine (PE) is a solution that, if properly employed, has the potential to offer numerous benefits, such as TCO minimization, standards compliance, facilitation in the development of complex RFID process-based solutions etc. However, APDL is not a solution that is applicable under any circumstances. APDL should be used in complex RFID solutions where all the EPC layers are used [10] and not in simple, standalone RFID applications.

- Also, APDL serves best the Supply Chain scenarios, where the complete EPC framework can be used. In these cases, APDL can be used in order to describe/configure the whole process chain while in cases where only a subset of the EPC layers is used, APDL would most probably be an overkill.
- APDL is a solution that enables a user to configure an EPCglobal compliant RFID middleware. Hence, the deferent features that are supported by the specific middleware and the RFID readers that can be attached to it (e.g., reader capabilities, multiple reads, collision problems, transmission errors, false positive and negative reads, read/write range ratio, etc) are EPC middleware implementation specific and not APDL language specific. The APDL and PE (programmable Engine) combination uses the EPC standard specifications, to configure the targeted middleware, so APDL can support the configuration of these functionalities (e.g. by using the LRSpec which is included at the EBProc's) as long as the middleware supports them.

Finally, concluding the discussion, we would like to mention that it is the Open Source AspireRFID community's intention to promote the APDL/BPWME/PE solution targeting towards its widely adoption and acceptance.

8. Conclusions

This paper introduces a workflow RFID oriented language (i.e. the APDL language) for describing non-trivial end-to-end RFID based solutions. A thorough review of the related work on process languages and standards reveals the absence of a configuration language tailored to RFID solutions. The introduced meta-language leverages the EPCglobal architecture and standards, since APDL solutions make use of EPC specifications towards full describing an RFID based solution. Nevertheless, the introduced language also includes concepts that extend the EPCglobal architecture, mainly in the areas of business events generation and enterprise applications integration.

APDL covers the wide range of solutions, which can be built based on the architecture blueprint provided by EPCglobal. Specifically, a rich set of RFID solutions can be described using constructs such as logical readers, company master data, specifications for tag data filtering, as well as automated generation of business events. Hence, APDL is not constrained to logistics scenarios, but it extends to a wider set of solution that can be expressed using ADPL elements. In the scope of the paper we have illustrated some realistic RFID systems and their description based on APDL.

We envisage that APDL could serve as a basis for an open specification of RFID solutions. Such a specification could greatly facilitate the development, deployment and integration of RFID solutions thus minimizing the total cost of ownership associated with an RFID solution deployment. Along with ease of deployment, APDL can also contribute to the faster prototyping of RFID solutions. It is also noteworthy that APDL is amendable by visual tools. This can further alleviate the task of developing and integrating RFID middleware solutions, through the replacement of low-level programming with visual development activities. The visual tools presented in this paper have already manifested the benefits of visual development.

The evolution of APDL to an open specification for RFID solutions, as well as the development of more integrated tools for visual RFID deployment are already planned as future work. Another future direction concerns the extension of the APDL concept to the wider set of sensor-based applications, which are (from a development and deployment perspective) associated with similar challenges and complexity.

Acknowledgments

Part of this work has been carried out in the scope of the ASPIRE project (FP7-215417) (http://www.fp7-aspire.eu). The authors acknowledge help and contributions from all partners of the project.

Appendix: Complete APDL Schema Definition

```xml
<?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>
```

Table 12 APDL Schema Definition

9. References

[1] V. Stanford, ‘Pervasive Computing Goes to Work: Interfacing to the Enterprise’, IEEE Pervasive Computing, Vol. 1, No. 3, pp.6-12, July 2002.

[2] George Lawton, “Machine-to-Machine Technology Gears Up for Growth”, IEEE Computer Vol. 37, Issue. 9, pp.12-15, September 2004.

[3] International Telecommunication Union, “The Internet of Things, Executive Summary” ITU Internet Reports 2005, November 2005, (electronically available at: http://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf)

[4] Christian Floerkemeier, Christof Roduner, and Matthias Lampe, ‘RFID Application Development with the Accada Middleware Platform’, IEEE Systems Journal, Vol. 1, Issue 2, pp.82-94, December 2007.

[5] S. Prabhu, Xiaoyong Su, Harish Ramamurthy, Chi-Cheng Chu, Rajit Gadh, “WinRFID –A Middleware for the enablement of Radio Frequency Identification (RFID) based Applications”, Invited chapter in Mobile, Wireless and Sensor Networks: Technology, Applications and Future Directions, Rajeev Shorey, Chan Mun Choon, Ooi Wei Tsang, A. Ananda (eds.), John Wiley, available at: http://www.wireless.ucla.edu/rfid/winrfid/.

[6] Achilleas Anagnostopoulos, John Soldatos and Sotiris G. Michalakos, ‘REFiLL: A Lightweight Programmable Middleware Platform for Cost Effective RFID Application Development’, Journal of Pervasive and Mobile Computing (Elsevier), Vol. 5, Issue 1, February 2009, pp. 49-63.

[7] EPCglobal, The EPCglobal Architecture Framework Version 1.3, available online at http://www.epcglobalinc.org/standards/architecture/

[8] EPCglobal standards, http://www.epcglobalinc.org/standards

[9] S. Sarma, “Integrating RFID,” ACM Queue, vol. 2, no. 7, pp. 50–57, 2004.

[10] Nikos Kefalakis, Nektarios Leontiadis, John Soldatos, Didier Donsez, “Middleware Building Blocks for Architecting RFID Systems”, MOBILIGHT 2009, pp. 325-336.

[11] White, S.: Using BPMN to model a BPEL process. BP Trends (2005), available online at http://www.businessprocesstrends.com

[12] C Ouyang, E Verbeek, WMP Van Der Aalst (2007): Formal semantics and analysis of control flow in WS-BPEL.

[13] Wil M.P. van der Aalst, A. H. M. ter Hofstede (2005): YAWL: Yet Another Workflow Language, In Information Systems, Vol. 30, No. 5, pages 245 – 275, Elsevier

[14] Martin Ward: Language Oriented Programming, In Software - Concepts and Tools, Vol.15, No.4, pp 147-161, 1994

[15] Wil M.P. van der Aalst: Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language, QUT Technical report, FIT-TR-2003-06, Queensland University of Technology, Brisbane, 2003, available online at http://www.workflowpatterns.com

[16] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, Charlton Barreto: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, available online at http://www.w3.org/TR/ws-cdl-10/

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

[18] EPCglobal, Object Naming Service (ONS), Version 1.0, EPCglobal Ratified Specification, Version of October 4, 2005.

[19] EPCglobal, EPC Information Services (EPCIS) Version 1.0.1, Specification, EPCglobal, September 2007, available at: www.epcglobalinc.org

[20] EPCglobal, The Application Level Events (ALE) Specification, Version 1.1, February 2008, available at: www.epcglobalinc.org.

[21] The AspireRFID project, http://forge.objectweb.org/projects/aspire/ (forge), http://wiki.aspire.objectweb.org/xwiki/bin/view/Main/WebHome (wiki)

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

[23] 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.
[24] Panos Dimitropoulos and John Soldatos, ‘RFID-enabled Fully Automated Warehouse Management: Adding the Business Context’, International Journal of Manufacturing Technology and Management (IJMTM) 2010 - Vol. 21, No.3/4 pp. 269 - 288.
[25] Nuseibeh, B. and Easterbrook, S., “Requirements engineering: a roadmap”, Int’l Conf. on Software Engineering, Ireland, 2000.
[26] Jan Holmström, Mikko Ketokivi and Ari-Pekka Hameri ”Bridging Practice and Theory: a Design Science Approach” Decision Sciences 40 (1), 2009.
[27] Creswell, J., “Research design: Qualitative, quantitative, and mixed methods approaches”, Sage Pubns, 2008.
[28] Hevner, A., March, S., Park, J. & Ram, S., “Design Science in Information Systems Research”, Management information systems quarterly 28(1), 2004, pp. 75–106.
[29] Jackson, M., “Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices”, Addison Wesley, 1995.
[30] Rhea Wessel, “Staff Jeans to Introduce RFID-enabled Customer Services”, RFID Journal, Oct 2010.
[31] Rifidi Project, http://www.rifidi.org/