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

Programmable RFID Solutions Specification

Due date of deliverable:	M30
Actual Submission date:	01.07.10

Deliverable ID:	**WP4/D4.4b**
Deliverable Title:	Programmable RFID Solutions Specification (Final Version)
Responsible partner:	AIT
Contributors:	Nikos Kefalakis - AIT John Soldatos - AIT Sofoklis Efremidis - AIT Yongming Luo - AIT Nikolaos Konstantinou - AIT Sofyan Mohammad Yousuf - OSI Neeli Rashmi Prasad - AAU Mathieu David - AAU Didier Donsez - UJF Kiev Gama - UJF Gabriel Pedraza - UJF
Estimated Indicative Person Months:	29

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

Revision: 1.1
Dissemination Level: PU

Document Information

Document Name: **Programmable RFID Solutions Specification**
Document ID: **WP4/D4.4b**
Revision: **1.1**
Revision Date: **2 July 2010**
Author: **AIT**
Security: **PU**

Approvals

	Name	Organization	Date	Visa
Coordinator	Neeli Rashmi Prasad	CTIF-AAU	28.06.10	Approved
Technical Coordinator	John Soldatos	AIT		
Quality Manager	Anne Bisgaard Pors	CTIF-AAU	28.06.10	Approved

Reviewers

Name	Organization	Date	Comments	Visa
Nathalie Mitton	INRIA	24 Jun 10	No Further Comments	Approved
Ramiro Samano Robles	IT	24 Jun 10	Consider Comments	Approved
Jean-Phillipe Leclercq	PV	24 Jun 10	No Further Comments	Approved

Document history

Revision	Date	Modification	Authors
a_0.1	07 July 09	Table of Contents	Nikos Kefalakis
a_0.2	20 July 09	Proposed changes to TOC	Sofyan Mohammad Yousuf
a_0.3	24 July 09	Final TOC and Edit guidelines	Nikos Kefalakis
a_0.4	20 Aug 09	Section 8, Section 5	Nikos Kefalakis
a_0.5	21 Aug 09	Section 3.2, Augmented Section 5.2	Mathieu David
a_0.6	31 Aug 09	Section 3.1, Appendix III	Sofyan Mohammad Yousuf
a_0.7	09 Sep 09	Section 2	Sofyan Mohammad Yousuf
a_0.8	13 Sep 09	Section 4.1, Section 4.3, Section 5.3, Augmented Section 2	Sofoklis Efremidis
a_0.9	14 Sep 09	Section 1, Section 9, revised Section 2	Mathieu David
a_1.0	14 Sep 09	Section 6	Nikos Kefalakis

a_1.1	17 Sep 09	Section 2, Section 4.2, Section 4.4, Section 4.5 and Augmented Section 5	Didier Donsez, Kiev Gama, Gabriel Pedraza
a_1.2	18 Sep 09	Augmented Section 5, added Appendixes and General Corrections	Nikos Kefalakis
a_1.3	22 Sep 09	Final Corrections Considering Internal Document Review Comments	Nikos Kefalakis
b_0.1	12 May 10	Added Sections 6.1, 6.2, 6.3 and 6.4	Nikos Kefalakis
b_0.2	15 May 10	Added Section 6 Introduction	Nikolaos Konstantinou
b_0.3	22 May 10	Augmented/Updated Section 6.5	Nikos Kefalakis
b_0.4	11 Jun 10	Merged a version's Sections 2 and 3 to Section 2, Augmented Section 3	Sofyan M. Yousuf
b_0.5	14 Jun 10	Section 1	John Soldatos, Nikos Kefalakis, Sofyan M. Yousuf
b_0.6	15 Jun 10	Added Section 7.1	Nikos Kefalakis, Yongming Luo
b_0.7	16 Jun 10	Added Section 7.2, Updated Section 8	Nikos Kefalakis
b_0.8	18 Jun 10	Augmented Section 2, Section 9	John Soldatos, Nikos Kefalakis
b_0.9	21 Jun 10	Various corrections to the hole document	Nikos Kefalakis
b_1.0	25 Jun 10	Incorporate some Review Comments	Sofyan M. Yousuf
b_1.1	27 Jun 10	Final Corrections	Nikos Kefalakis

Content

Section 1 Executive Summary

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 deliverable we introduce APDL, an XML based specification for describing and configuring RFID solutions. We present the final work involved in investigating and producing the meta-languages for defining; configuring and deploying RFID based solutions. This final version includes the part of the RFID domain specific language specifying the composition of filters, devices, readers, corporate databases, business services etc. into fully fledged RFID solutions.

After analysis of available workflow supporting languages we concluded that they are rather general as they have been designed to model a variety of workflow environments capturing thus most of the well known business processes. Thus specialized concepts like RFID-based processes and RFID related data were rather cumbersome to express in general purpose workflow modeling languages. XPDL would probably be the most appropriate candidate to use taking in considerations the RFID language requirements but following the same requirements due to the generality and complexity of XPDL for describing an RFID Open Loop Business Process is not preferred.

Therefore the need for a new special purpose modeling language that will be able to represent in a clean way RFID related concepts and a workflow editor that would accompany it forced us to design such a Domain Specific Language that uses many of the XPDL's notions but with a different structure which we have named APDL (ApireRFID Process Description Language). APDL is an XPDL hybrid that is simpler to understand, to describe its structure and is specializes on RFID Data and Business Process modeling.

In this regard, the Programmable Meta-Language is a combination of the following Specifications; Logical Readers Specs, ECSpecs, Master Data Document, Middleware Management/Configuration Data and Business Workflow data. All the above are augmented with design data (like XPDL) for the visualization of the RFID solution to the BPWME (Business Process Workflow Management Editor).

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 in this deliverable.

Section 2 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.

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 [10], which specifies several standards [11] and related middleware building blocks for building RFID systems for logistics and warehouse management. 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 [6]. Most of this effort relates to the need for configuring the middleware building blocks of the application [7], while at the same time ensuring the interworking and interoperability of these building blocks.

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 deliverable defines the specification of a meta-language for describing 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).

Generally language oriented programming [74] provides an approach for solving problems using a language suited to a given problem domain. It is a sort of metaprogramming using domain specific languages created by programmers, who map the concepts of the problem domain (e.g. an RFID reader, a configuration file, a sensor) to be expressed in that language. After having such language modeled, the programmer then tries to solve its problem in his/her domain by using that custom language instead of a general purpose programming language (e.g. Java, C). This is exactly what ASPIRE peruses to achieve for the end user (RFID integrator, SME owner). So as someone would be capable to do a high level description of the requirements with a specific Meta Language and the system would handle the rest without the need of a specialized programmer.

Important concepts brought by such approach are the Domain Specific Languages (DSL) themselves and the concept of separation of concerns, which are detailed in the next sections

2.1 Domain Specific Languages

Instead of being general such as, for example, Java and C languages, DSLs focus on expressiveness in a limited domain. By providing notations and constructs custom made to a particular application domain (e.g. RFID applications), DSLs offer significant advantages in expressiveness and have an easier use when compared with General Purpose Programming Languages (GPLs). Also, a larger group of software developers can be reached with DSLs [76].

Some of the decisions that may lead to developing a DSL are the improved software economics, and also the enabling of software development by users with less expertise on the domain but with expertise in programming, or even by end-users with knowledge in the domain, but no programming expertise.

An simple example of DSL is Excel's macro language, which is a DSL for spreadsheet applications adding programmability to Excel's fundamental interactive mode. Another well known DSL is TeX, which was developed for expressing the structure of documents for typesetting purposes. Interestingly enough TeX itself was implemented in another DSL called WEB, which was developed exactly for this purpose. By appropriately establishing domain specific notations, one can increase productivity of the target programmer audience. When using GPLs, it is much harder to achieve the expressiveness of domain-specific notations. Such concepts cannot be mapped in a straightforward manner to functions or objects of libraries developed with General Purpose Programming Language.

The usage of DSLs brings the possibilities for analyzing, verifying, optimizing, parallelizing and transforming DSL constructs that would be much harder or unfeasible if using a GPL. This is mostly due to the fact that patterns of GPL source code that are involved in such a process are either too complex or not well defined.

The usage of a DSL-based front-end is a handy tool that may be used for dealing with a system's configuration and adaptation. However, DSLs are not necessarily executable. They can be used only to represent a domain specific problem, but usually they are run by execution engines.

2.2 Separation of Concerns

Observations on the source code of relatively complex applications [77] show that the same unity of code focuses in different concerns (e.g.: concurrency, security, accounting, distribution, transaction). The separation of concerns [78] is a software engineering paradigm that has the objective of dissociating the different concerns that compose a program. This dissociation makes the code more readable and understandable by keeping these concerns separated from the main application code. As a consequence, this separation of can describe the different concerns of a program and their interrelations in a more abstract manner. This approach allows the construction of applications that manage theses different concerns without needing to change the application source code.

Some of the advantages of the separation of concerns are:

- A program describe in an abstract manner with different concerns dissociated can have an easier implementation. This happens due to the fact that each concern can be programmed independently. [79]
- Reading a program divided in different concerns is easier, because the application code does not contain a mix of all concerns
- Specialists on each concern can work separately without interfering in application code, without needing to know details of the program. If the coupling between the concerns is weak, it becomes easier to be modified as well as reusing each concern independently. In this case, the unity of reuse is no longer the code, but the concern.

2.3 ASPIRE Programmable RFID solutions

Despite the simplicity of the operational principles of RFID technology (i.e. tags responding to readers requests), the design of a complete RFID system encompasses complex interactions not only between different layers of the OSI (Open Systems Interconnection) model, but it also involves several market, privacy, security, and business issues. This heterogeneous landscape calls for a middleware platform which is able to consider all these complex variables in a flexible and modular way, which is able to provide a starting point for future upgrades and innovations, and which considerably reduces the implementation costs of RFID solutions.

APDL has been carefully designed to be simple, intuitive and comprehensive, which facilitates its use by the majority of RFID developers, integrators and consultants. 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 [50] (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 deliverable presents tools and techniques enabling RFID application development and deployment.

The programmability features in ASPIRE aim at easing the configuration of ASPIRE solutions. The ASPIRE programmability functionality offers RFID developers and consultants the possibility to deploy RFID solutions through entering high-level meta-data for a company (including the business context of its RFID deployments), rather than through writing significant amounts of low-level programming statements.

Solutions are deployed simply through explicit representation of business processes with their activities and their logical relations. The Business Process Workflow Management Editor then configures the middleware to execute these processes as specified in the process model.

Along with such ease of deployment the functionality also eliminates the limitations of gathering requirements by IT which are overcome by allowing the owners of the process to participate fully in the design, deployment and management of these processes.

Visual representation as such would simply enable emergence of different views from a common source allowing different participants from different areas of specialty to have a common language.
Hereafter the document is structure as follows:

Section 3 explores various available models, workflows and modeling languages. It first introduces the concept of business process management and the purpose

of the modeling approach. The section also investigates available business process workflow concepts and languages.

Section 4 evaluates available OSS XPDL Editors. Specifically four well recognized XPDL editors were investigated to determine their relevance in providing RFID specific solutions. These evaluations together with the pros and cons of each are presented in this section.

Section 5 details the RFID language specification requirements. It then compares and evaluates the available process languages and the section ends with the decision which and why would be the most suitable language for describing RFID Business Processes.

Section 6 introduces the AspireRFID Process Description Language (APDL). It describes the programmable Meta Language structure and outlines the process description language specifications.

Section 7 presents the available tools that facilitates the APDL's usability and more specifically the Language's Visual Editor and its "interpreter" to the ASPIRE infrastructure.

Section 8 uses an example to describe how the APDL can be used to describe an RFID workflow process.

This document concludes with a section that summarizes and outlines the main findings of the deliverable. We expect the programmability and the introduction of process modeling will significantly boost the adoption of RFID technology, especially for SME communities that wish to use RFID as an innovation vehicle.

Section 3 Available Models, Workflows and Languages Investigation

This section introduces the concept of business process management and the purpose of using a modeling approach. It further explores the various available models, workflows and modeling languages that could be used in order to provide a comprehensive understanding of the subject.

3.1 General

3.1.1 Business Process Management

Today, markets have matured, globalisation and the wide spread internet has given consumers the negotiating power. In such circumstances; to survive and gain a competitive advantage companies are required to look for ways to increase customer satisfaction, improve operations, reduce cost of doing business, and establish new products and services at a low cost with supreme agility. Companies realise that each product or service it produces is the outcome of a number of activities performed [68] and therefore are on the watch for methods, skills and tools that will enable them to create processes that would yield customers to pay to see them do it again and again [66].

The RFID technology holds potential solutions to a wide range of management problems from abilities to increase efficiency of inventorying goods transported in and out of warehouses or distribution centres without unloading or digging through pallets and packaging through to better product visibility hence management of product availability on shelves to reducing problems of shrinkage. While RFID would make products visibility possible, there exists a need to create, implement and monitor these new processes in an efficient and effective manner. A methodology alike RFID that helps make processes more visible and explicit such that they could be manipulated to produce more efficient and effective results. Business process management is just that, it is a new approach to business process innovation and management [66].

> "BPM defines, enables and manages the exchange of business information on the basis of a process view that incorporates employees, customers, partner, application and databases. From a business prospect, BPM streamlines business processes both internal and external, eliminating redundancies and increasing automation, enabling end to end visibility, control and accountability of processes." [68]

BPM is all about the efficient and effective management of business processes, it does not look at machines and systems as the main part of the process at the same time it does not ignore them. It recognises a process incorporates employees, systems and automated machines [65]. It doesn't view IT as being the core of process change but doesn't ignore it as TQM or Six Sigma does [65]. In fact Business Process management is a convergence of management theories like TQM, Six Sigma, BPR with modern technologies like application development, Service oriented architecture, workflows, etc into a unified whole [66].

In a way, BPM uses the good aspects of previous management theories such as creativity and insight from Business Process Reengineering and ignores the discontinuity or radicalisation of processes and process introduction. Therefore BPM should not be looked at as just a digitizing system, or just as another management theory, or as a one stop solution for all management problems. Instead process management should be looked with the view of 'not to automate but obliterate'. This mantra has been followed for the past decade but it is only now that methods and technology have become available to fully enable process management in such manner [66]. Rarely would you find organisations today explicitly trying to reengineer its processes. However, finding large corporations who aren't explicitly focussing on design and management of its processes is also rare [67].

BPM is based on the theory that a product or service company provides is the outcome of a number of activities performed [68]. Business processes make up the organisation and execution of these activities and are a critical source to improvement of the outcomes from these activities. Technology (information technology and information systems) per se plays a vital part in the management and execution of these processes, since more and more of the activities performed by an organisation are supported by information systems. These executions are either solely executed by the information systems (i.e. automated) or performed with the input of employees. Therefore, for an organisation to realise its business goals in an efficient and effective manner require a successful amalgamation of the people and the information systems. Business process and their management are important concepts that facilitate this effective collaboration [68].

Business processes not only are the underpinning driver of an organisation but also are essential towards design and realisation of technology. These technologies such as RFID provide the ground works for rapid creation of new functionalities that realise new products/services and for adapting new functionalities that cater to new market requirements and gain competitive advantage [68].Business Process Management incorporates concepts and technologies from both fields; business administration and computer science to provide a process centric approach towards improvement of business processes i.e. organising companies on the basis of their business processes.

Business process in conceptual terms as defined by Davenport (1993) is "a set of logically related tasks performed to achieve a defined business outcome for a particular customer or market." The term 'logically related' puts emphasis on how tasks are performed as compared to what tasks are performed. In addition, Davenport [64] also elaborates that a process is "a specific ordering of work activities across time and space, with a beginning, an end and clearly defined inputs and outputs." The definition also recognizes that customers could be internal or external therefore while processes are enacted by a single organisation, they could interact with processes performed by other organisations i.e. they occur across or between organisational subunits.

Once business processes are formally established Business Process management therefore is a set of concepts, methods, and techniques that support the design, administration, configuration, enactment, and analysis of business processes [68]. The underline methodology of BPM is the explicit representation of business processes with their activities and their logical relations. Once recognized, these processes can be analysed, improved and enacted. As elaborated in later sections these business processes can be enacted in two ways. First by encouraging employees to follow new procedures and policies constructed. Alternatively, software systems can be used to coordinate the enactment of these business processes. The ASPIRE RFID Middleware Programmability incorporates these concepts and methodologies to do just that by providing a programmable workspace which would allow for the design and mechanised implementation of these processes.

This explicit process representation of business activities is formally known as business process modelling. While there are several graphical notations for business process modelling, their underlying methodology is quite similar. Figure below shows a simplified version of one such modelling notation, the Business Process Modelling Notation (BPMN).

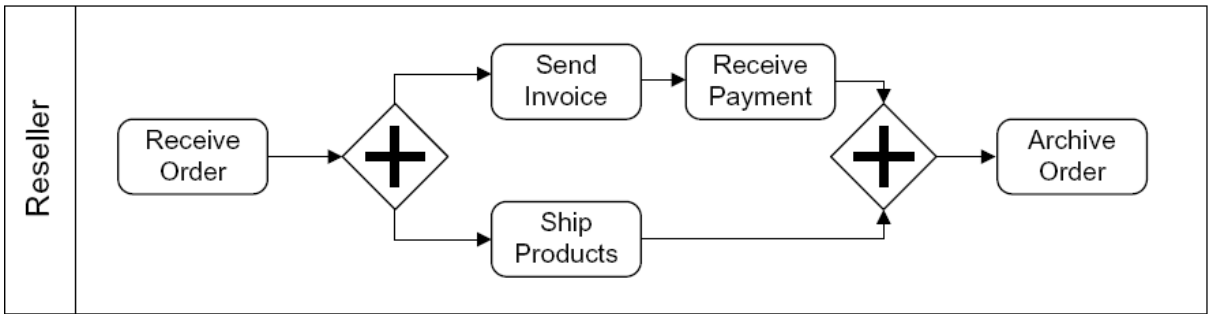


Figure 1 Simple Ordering Process [68]

Figure 1 above shows a business process model of a resellers ordering process. This model can be used as a blueprint to allow the company to organize its work. The company can receive many orders each of which can be processed as described in the blueprint. Each processed order is also called a business process instance. i.e each model acts as a blueprint for a set of business process instances. As mentioned earlier, business process models are the main artifacts for implementing business processes. These could be done either through procedures and policies or through the use of Business Process Management System (BPMS). The BPMS executes these processes and ensures that all business process instances are executed as specified in the process model.

Furthermore IT systems inability to address process improvements effectively to some extent lies in its current techniques of capturing business requirements and translating them into system behavior. Whereby each contributor involved whether that be a project manager, an analyst or programmer brings a significantly different terminology and frame of reference to bear on the problem.

Business process modeling is a new approach to process design and implementation that addresses these problems by allowing the development of a

single definition of business process from which different views of the process can be depicted without causing the disarray of the process. The creation of a unified process representation allows different people with different skills to view and manipulate the process in their way and still build a coherent process. The process view for a business analyst would mean a high level process map. While to a programmer the process would look like a process language comparable to a programmable language. Whereas, to an employee it would look like a process map showing how processes interact. BPM enables this emergence of different views from a common source allowing different participants from different areas of specialty to have a common language.

The use of BPM gets the business people who are the owners of the processes and the IT who devise applications to automate these processes under the same process design environment using graphical notations. The limitations of gathering requirements by IT are overcome by allowing the owners of the process to participate fully in the design, deployment and management of these processes.

Business process management does not require technology implicitly to bring about an improvement in processes or per se require automation. However, BPM is a methodology which uses process modeling to achieve process improvement in a time effective manner. With RFID, the aim is to automate certain processes and achieve a level of real time visibility hence making processes more efficient and effective. In this regard the use of BPM to automate the processes using ASPIRE RFID would yield towards more effective outcomes.

3.1.2 Workflow patterns

BPM is an amalgamation of various concepts and technologies. One such technology is the Workflow management system. A Workflow management system defines, creates, and manages the execution of workflows through the use of software, running on one or more workflow engines [68]. Workflow solutions introduced in the 90's primarily consisted of an engine and a language. Many solutions also included some type of graphical modeling environment, albeit rudimentary. Only a handful included a more robust, UML-based modeler; more likely it was a proprietary modeler [71].

A workflow is a model to represent real work for further assessment, e.g., for describing a reliably repeatable sequence of operations. Likewise a workflow pattern describes the behavior of business processes [69]. It "is the abstraction from concrete form which keeps recurring in specific non arbitrary contexts" [60].

The purpose of establishing workflow patterns was to identify the requirements that a Workflow management system would have in order to model and execute business processes. Patterns based approach was taken to describing these requirements as it offered both a language independent and technology-independent means of expressing their core characteristics in a form that was sufficiently generic to allow for its application to a wide variety of offerings [61].

The patterns specified by Wil van der Aalst et al. [62] range from very simple to very complex and cover the behaviors that can be captured within most business process models [69].

Although initially focused on workflow systems, it soon became clear that the patterns were applicable in a much broader sense and they started being used to examine the capabilities of business process modeling languages such as BPMN, UML Activity Diagrams and EPCs, web service composition languages such as WCSI and business process execution languages such as BPML, XPDL and BPEL [51]. In addition, these patterns have also directly influence the development of BPMN and BPEL standards. Will van der and associates also claimed that most of the proposed patterns can be easily mapped using existing languages or realized through implementation. While, there are patterns that are supported only by a small minority of the work flow management systems. In addition, no contemporary workflow management system supports all patterns [61].

The application of a patterns-based approach to the identification of generic workflow constructs was first proposed by Wil Van Der Aalst et. Al [62], which identified a collection of patterns focused on one specific aspect of process-oriented application development, namely the control flow perspective of the workflows system. The original twenty-one control flow patterns were identified through a comprehensive evaluation of workflow systems and process modeling formalisms [61]. These patterns describing the control-flow perspective of workflow systems are divided into the following categories:

- Basic Control flow Patterns
- Advance branching and synchronization patterns
- Structural Patterns
- Multiple Instances (MI)
- State based patterns
- Cancellation patterns

In Appendix III the patterns in each of the categories as specified by Wil Van Der Aalst et al in the Journal Distributed and Parallel Databases, Volume 14, Issue 3, pages 5-51, July 2003 are outlined. Upon their introduction in 2003 these patterns have formed the basis of many other researches. The views of 2 such researches; Weske [68] and Aalst [62] also incorporated in the following sections.

The above workflow patterns have been used to examine the capabilities of business process modeling languages such as BPMN, UML Activity Diagrams and EPCs, web service composition languages such as WCSI and business process execution languages such as BPML, XPDL and BPEL [61]. In addition, these patterns have also directly influence the development of BPMN and BPEL standards. The workflow patterns have also been used as initial requirements in the design of a workflow language and open-source system called YAWL.

White, S.A [69] reviewed how the two graphical process modeling notations, the BPMN from the Business Process Management Initiative (BPMI), and the UML 2.0

Activity Diagram from the Object Management Group (OMG), can represent the workflow patterns. The solutions of the two notations were compared for technical ability to represent the patterns as well as their readability.

The examination revealed the core similarities and differences between the two notations. First in an assessment of how the two could model the workflow patterns resulted in both being able to adequately model most of the patterns. Furthermore, the point that both notations provided similar solutions to most of the patterns indicates the similarity in presentation between the notations. Minor differences between the two also exist such as in modeling objects shapes and to some extent in the terminology [69]. For instance, while an Activity Diagram has a start node a Business Process Diagram has a Start Event.

Such high rates of similarities between the two diagrams are present because both of them are designed to solve the same problem, modeling of business processes. In contrast, the differences between the two also exist mainly because both target different kind of users. While BPMN was created to provide business people with an easy way of modeling and taking ownership of their processes, the UML focused its efforts on standardizing the modeling for programmes in software development. Although with UML 2.0 development aimed at crafting a more user friendly activity diagram such that it could be used by business people, it is still more technically oriented [69]. Some analysts do see the two notations converge into one since there are huge similarities between the two and both also share the characteristic of being a view (a diagram) for the Business Process Definition metamodel [69].

3.2 Business Process Modeling

Business Process Modeling (BPM) is the representation of current ("as is") and proposed ("to be") enterprise processes, so that they may be compared and contrasted. By comparing and contrasting current and proposed enterprise processes business analysts and managers can identify specific process transformations that can result in quantifiable improvements to their businesses [51]. Large applications must be more than just an aggregate of software modules: these applications must be structured (architected) in a way that the architecture enables scalability and reliable execution under normal or stressed conditions. The structure of these applications must be defined clearly and unambiguously so that:

- Maintenance staff can quickly locate and fix any bugs that may show up long after the original programmers have moved on;
- Developers can add new features that may be required over time by the business users.

Another benefit of an architected structure is that it enables code reuse: design time is the best time to seek to structure an application as a collection of self-contained modules or components. In this context, modeling is the process of architecting and structurally designing a software application before starting the coding phase. Modeling is a critical effort for large software projects, and it is also useful for medium projects. Using a model, developers can assure

themselves that business functionality is complete and correct, that end-user needs are met, and that program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation in code makes changes difficult and expensive to make [52].

3.2.1 Available Business Processes Workflow Definition Concepts

3.2.1.1 Business Process Definition Metamodel

The Business Process Definition Metamodel (BPDM) is a standard definition of concepts used to express business process models (a metamodel), adopted by the OMG (Object Management Group). Metamodels define concepts, relationships, and semantics for exchange of user models between different modeling tools. The exchange format is defined by XSD (XML Schema) and XMI (XML for Metadata Interchange), a specification for transformation of OMG metamodels to XML. BPDM provides abstract concepts as the basis for consistent interpretation of specialized concepts used by business process modelers. For example, the ordering of many of the graphical elements in a BPMN (Business Process Modeling Notation) diagram is depicted by arrows between those elements, but the specific elements can have a variety of characteristics. For example, all BPMN events have some common characteristics, and a variety of specific events are designated by the type of circle and the icon in the circle. The abstract BPDM concepts ensure implementers of different modeling tools will associate the same characteristics and semantics with the modeling elements to ensure models are interpreted the same way when moved to a different tool. Users of the modeling tools do not need to be concerned with the abstractions, they only see the specialized elements [53]. BPDM extends business process modeling beyond the elements defined by BPMN and BPEL (Business Process Execution Language) to include interactions between otherwise-independent business processes executing in different business units or enterprises (choreography). Choreography can be specified independently of its participants, and used as a requirement for the specification of the orchestration implemented by a participant. BPDM provides for the binding of orchestration to choreography to ensure compatibility. Many current business process models focus on specification of executable business processes that execute within an enterprise (orchestration).

3.2.1.2 Business Process Modeling Notation

The Business Process Modeling Notation (BPMN) is a standard modeling notation that provides a graphical notation for expressing business processes in a Business Process Diagram (BPD) in a way that is readily understandable by business users; from the business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor the processes. The BPMN specification also provides a binding between the notation's graphical elements and the constructs of block-structured process execution languages, including Business Process

Modeling Language (BPML) and Business Process Execution Language for Web Services (BPEL-WS) [52].

3.2.1.2.1 BPMN overview

BPMN provides businesses with the capability of understanding their internal business procedures in a graphical notation and will give organizations the ability to communicate these procedures in a standardized manner. Currently, there are many process modeling tools and methodologies. Given that individuals may move from one company to another and that companies may merge and diverge, it is likely that business analysts are required to understand multiple representations of business processes—potentially different representations of the same process as it moves through its life cycle of development, implementation, execution, monitoring, and analysis. Therefore, a standard graphical notation facilitates the understanding of the performance collaborations and business transactions within and between the organizations. This ensures that businesses understand their own environments and the environment of participants in their business, and will enable organizations to adjust to new internal and B2B business circumstances quickly. To do this, BPMN follows the tradition of flowcharting notations for readability but at the same time provides mapping to the executable constructs [52].

3.2.1.2.2 BPMN uses

Business process modeling is used to communicate a wide variety of information to a wide variety of audiences. It is designed to cover this wide range of usage and allows modeling of end-to-end business processes to allow the viewer of the diagram to be able to easily differentiate between sections of a BPMN diagram. There are three basic types of sub-models within an end-to-end BPMN model:

- Private (internal) business processes
- Abstract (public) processes
- Collaboration (global) processes

Private business processes are those that are internal to a specific organization and are the types of processes that have been generally called "workflow" or "BPM processes". A single private business process will map to a single BPEL-WS document. If swim lanes are used, then a private business process will be contained within a single Pool. The Sequence Flow of the Process is therefore contained within the Pool and cannot cross its boundaries. Message Flow can cross the Pool boundary to show the interactions that exist among separate private business processes. Thus, a single BPMN diagram may show multiple private business processes, each mapping to a separate BPEL-WS process.

Abstract processes represent the interactions between a private business process and another process or participant. Only those activities that are used to communicate outside the private business process are included in the abstract process. All other "internal" activities of the private business process are not shown in the abstract process. Thus, the abstract process shows to the outside world the sequence of messages that is required to interact with that business

process. Abstract processes are contained within a Pool and can be modeled separately or within a larger BPMN diagram to show the Message Flow between the abstract process activities and other entities. If the abstract process is in the same diagram as its corresponding private business process, then the activities that are common to both processes can be associated.

A collaboration process depicts the interactions among two or more business entities. These interactions are defined as a sequence of activities that represents the message exchange patterns among the entities involved. A single collaboration process may be mapped to various collaboration languages, such as ebXML BPSS, RosettaNet, or the resultant specification from the W3C Choreography Working Group. Collaboration processes may be contained within a Pool, and the different participant business interactions are shown as Lanes within the Pool. In this situation, each Lane would represent two participants and a direction of travel between them. They may also be shown as two or more Abstract Processes interacting through Message Flow. These processes can be modeled separately or within a larger BPMN diagram to show the Associations between the collaboration process activities and other entities. If the collaboration process is in the same diagram as one of its corresponding private business processes, then the activities common to both processes can be associated.

3.2.1.2.3 Types of BPMN Diagram

Within and between these three BPMN sub-models, many types of diagrams can be created. The following are the types of business processes that can be modeled with BPMN (those with asterisks may not map to an executable language):

- High-level private process activities (not functional breakdown)
- Detailed private business process
  - As-is, or old, business process
  - To-be, or new, business process
- Detailed private business process with interactions among one or more external entities (or "black box" processes)
- Two or more detailed private business processes interacting
- Detailed private business process relationship with Abstract Process
- Detailed private business process relationship with Collaboration Process
- Two or more Abstract Processes
- Abstract Process relationship with Collaboration Process
- Collaboration Process only (e.g., ebXML BPSS, or RosettaNet)
- Two or more detailed private business processes interacting through their Abstract Processes
- Two or more detailed private business processes interacting through a Collaboration Process
  - Two or more detailed private business processes interacting through their Abstract Processes and a Collaboration Process

BPMN is designed to allow all the foregoing types of diagrams. However, it should be cautioned that if too many types of sub-models are combined, such as three

or more private processes with message flow between each of them, then the diagram may become too hard for someone to understand. Thus, we recommend that the modeler pick a focused purpose for the BPD, such as a private process, or a collaboration process.

3.2.1.2.4 Business Process Diagrams

This section provides a summary of the BPMN graphical objects and their interrelationships. One of the goals of BPMN is that the notation be simple and adoptable by business analysts. Also, there is a potentially conflicting requirement that BPMN provide the power to depict complex business processes and map to BPM execution languages. To help understand how BPMN can manage both requirements, the list of BPMN graphic elements is presented in two groups. First, there are the core elements that support the requirement of a simple notation. These are the elements that define the basic look and feel of BPMN. Most business processes can be modeled adequately with these elements. Second, all the elements, including the core elements, help support the requirement of a powerful notation to handle more advanced modeling situations. Further, the graphical elements of the notation are supported by non-graphical attributes that provide the remaining information necessary to map to an execution language or for other business modeling purposes [52].

It should be emphasized that one of the drivers for the development of BPMN is to create a simple mechanism for creating business process models. Of the core element set, there are three primary modeling elements (flow objects):

- Events
- Activities
- Gateways

There are three ways of connecting the primary modeling elements:

- Sequence Flow
- Message Flow
- Association

There are two ways of grouping the primary modeling elements through Swim lanes:

- Pools
- Lanes

Table 1 below displays a list of the core modeling elements that are depicted by the notation.

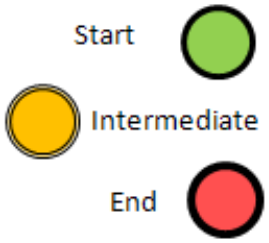
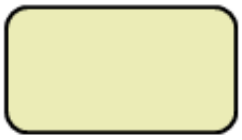
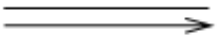
Element	Description	Notation
Event	An event is something that "happens" during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.	
Activity	An activity is a generic term for work that the company performs. An activity can be atomic or nonatomic (compound). The types of activities that are a part of a Process Model are Process, Subprocess, and Task. Tasks and Subprocesses are rounded rectangles. Processes are either unbounded or a contained within a Pool.	
Gateway	A Gateway is used to control the divergence and convergence of Sequence Flow. Thus, it will determine branching, forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control.	
Sequence Flow	A Sequence Flow is used to show the order that activities will be performed in a Process.	
Message Flow	A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the diagram will represent the two entities (participants).	
Association	An Association is used to associate information with flow objects. Text and graphical nonflow objects can be associated with the flow objects.	
Pool	A Pool is a "swimlane" and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.	
Lane	A Lane is a subpartition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize activities.	

Table 1 Business Process Diagram Primary Elements

3.2.2 Activity Diagram (UML) another modeling tool such as BPMN

Activity diagrams are a loosely defined diagram technique for showing workflows of stepwise activities and actions, with support for choice, iteration and concurrency. UML 2 activity diagrams are typically used for business process modeling, for modeling the logic captured by a single use case or usage scenario,

or for modeling the detailed logic of a business rule. Although UML activity diagrams could potentially model the internal logic of a complex operation it would be far better to simply rewrite the operation so that it is simple enough that you don't require an activity diagram. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development [54]. An example of UML activity diagram is shown in the Figure 2 below.

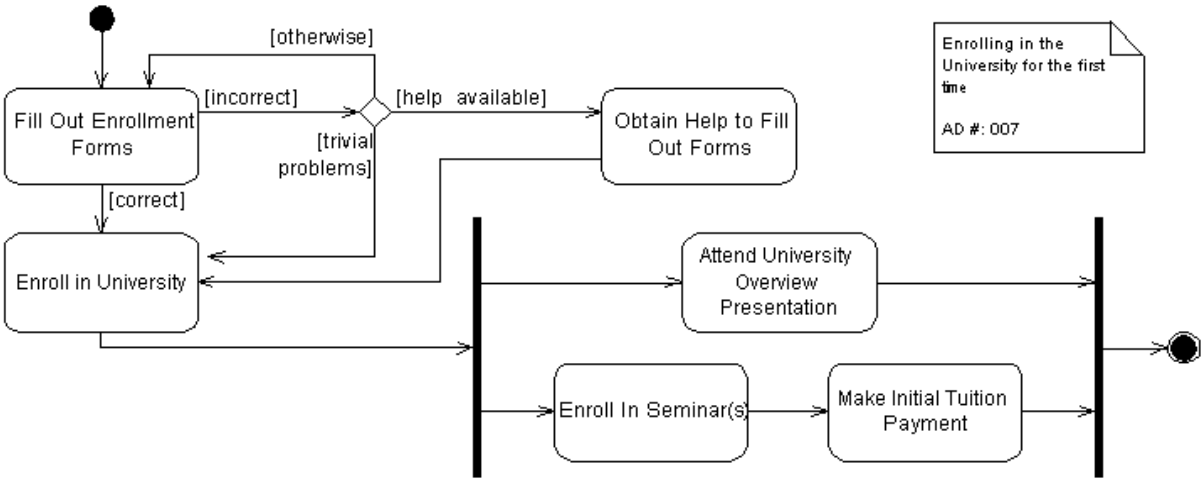


Figure 2 Business Process modeled with an activity diagram [55]

3.2.3 Programming languages for BPM

3.2.3.1 Business Process Modeling Language

The Business Process Modeling Language (BPML) is one example of an effort at BPM standardization. BPML is a metalanguage for the modeling of business processes. It provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional finite-state machine [BPM200501]. The language provides a model for expressing business processes and supporting entities. BPML defines a formal model for expressing abstract and executable processes that address all aspects of enterprise business processes, including activities of varying complexity, transactions and their compensation, data management, concurrency, exception handling, and operational semantics. BPML also provides a grammar in the form of an eXtensible Markup Language (XML) Schema for enabling the persistence and interchange of definitions across heterogeneous systems and modeling tools. BPML itself does not define any application semantics such as particular processes or application of processes in a specific domain; rather, BPML defines an abstract model and grammar for expressing generic processes. This allows BPML to be used for a variety of purposes that include, but are not limited to, the definition of enterprise business processes, the definition of complex Web Services (WS), and, the definition of multiparty collaborations.

3.2.3.2 Business Process Execution Language

Business Process Execution Language (BPEL), short for Business Process Execution Language for Web Services (BPEL-WS) is an OASIS standard executable language for specifying interactions with Web Services. Processes in Business Process Execution Language export and import information by using Web Service interfaces exclusively. BPEL-WS provides a language for the specification of Executable and Abstract business processes. By doing so, it extends the Web Services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration both within and between businesses.

BPEL is an Orchestration language, not a choreography language. The primary difference between orchestration and choreography is executability and control. An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. A choreography specifies a protocol for peer-to-peer interactions, defining, e.g., the legal sequences of messages exchanged with the purpose of guaranteeing interoperability. Such a protocol is not directly executable, as it allows many different realizations (processes that comply with it). A choreography can be realized by writing an orchestration (e.g. in the form of a BPEL process) for each peer involved in it. The orchestration and the choreography distinctions are based on analogies: orchestration refers to the central control (by the conductor) of the behavior of a distributed system (the orchestra consisting of many players), while choreography refers to a distributed system (the dancing team), which operate according to rules but without centralized control [46]. BPEL's focus on modern business processes, plus the histories of WSFL and XLANG, led BPEL to adopt web services as its external communication mechanism. Thus BPEL's messaging facilities depend on the use of the Web Services Description Language 1.1 (WSDL) to describe outgoing and incoming messages.
In addition to providing facilities to enable sending and receiving messages, the BPEL programming language also supports:

- A property-based message correlation mechanism XML and WSDL typed variables
- An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports XPath 1.0 by default
- Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel)
- A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers
- Serialized scopes to control concurrent access to variables

3.2.3.3 XML Process Definition Language (XPDL)

XPDL is the Serialization Format for BPMN. XPDL provides a file format that supports every aspect of the BPMN process definition notation including graphical

descriptions of the diagram, as well as executable properties used at run time. 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.
XPDL Enables a Process Definition Ecosystem. XPDL is extensible so that it allows each different tool to store implementation specific information within the XPDL, and have those values preserved even when manipulated by tools that do not understand those extensions. This is the only way to provide for a "round trip" through multiple tool and still be able to return to the original tool with complete fidelity [57].

XPDL uses an XML-based syntax, specified by an XML schema. The main elements of the language are: Package, Application, Workflow-Process, Activity, Transition, Participant, DataField, and DataType. The Package element is the container holding the other elements. The Application element is used to specify the applications/tools invoked by the workflow processes defined in a package. The element WorkflowProcess is used to define workflow processes or parts of workflow processes. A Patterns and XPDL 4 WorkflowProcess are composed of elements of type Activity and Transition. The Activity element is the basic building block of a workflow process definition. Elements of type Activity are connected through elements of type Transition. There are three types of activities: Route, Implementation, and BlockActivity. Activities of type Route are dummy activities just used for routing purposes. Activities of type BlockActivity are used to execute sets of smaller activities. Element ActivitySet refers to a self contained set of activities and transitions. A BlockActivity executes such an ActivitySet. Activities of type Implementation are steps in the process which are implemented by manual procedures (No), implemented by one of more applications (Tool), or implemented by another workflow process (Subflow). The Participant element is used to specify the participants in the workflow, i.e., the entities that can execute work. There are 6 types of participants: ResourceSet, Resource, Role, OrganizationalUnit, Human, and System. Elements of type DataField and DataType are used to specify workflow relevant data. Data is used to make decisions or to refer to data outside of the workflow, and is passed between activities and subflows [58].

3.2.3.4 Yet Another Workflow Language (YAWL)

Yet Another Workflow Language (YAWL) is a workflow language based on the Workflow patterns. The language is supported by a software system that includes an execution engine, a graphical editor and a worklist handler. The system is available as an Open source software under the LGPL license. The original drivers behind YAWL were to define a workflow language that would support all (or most) of the Workflow Patterns and that would have a formal semantics. Observing that Petri nets came close to supporting most of the Workflow Patterns, the designers of YAWL decided to take Petri nets as a starting point and to extend this formalism with three main constructs, namely or-join, cancellation sets, and multi-instance activities. These three concepts are aimed at supporting five of the Workflow Patterns that were not directly supported in Petri nets, namely synchronizing merge, discriminator, N-out-of-M join, multiple instance with no a priori runtime knowledge and cancel case. In addition, YAWL adds

some syntactical elements to Petri nets in order to intuitively capture other workflow patterns such as simple choice (xor-split), simple merge (xor-join), and multiple choice (or-split). During the design of the language, it turned out that some of the extensions that were added to Petri nets were difficult or even impossible to re-encode back into plain Petri nets. As a result, the original formal semantics of YAWL is defined as a Labeled transition system and not in terms of Petri nets. The fact that YAWL is based on a formal semantics has enabled the implementation of several techniques for analyzing YAWL processes [59].
Yawl provides comprehensive support for control-flow patterns and can thus be considered a highly expressive language. The graphical manifestations of the various concepts for control-flow specification in Yawl are shown in Figure 3. Yawl extends Workflow nets with concepts for the OR-split and the OR-join, for cancellation regions, and for multiple instance tasks. In Yawl terminology transitions are referred to as tasks and places as conditions. As a notational abbreviation, when tasks are in a sequence they can be connected directly (without adding a connecting place). The expressiveness of Yawl allows for models that are relatively compact as no elaborate work-arounds for certain patterns are needed. Therefore the essence of a model is relatively clear and this facilitates subsequent adaptation should that be required. Moreover, by providing comprehensive pattern support Yawl provides flexibility by design and tries to prevent the need for change, deviation, or underspecification [60].

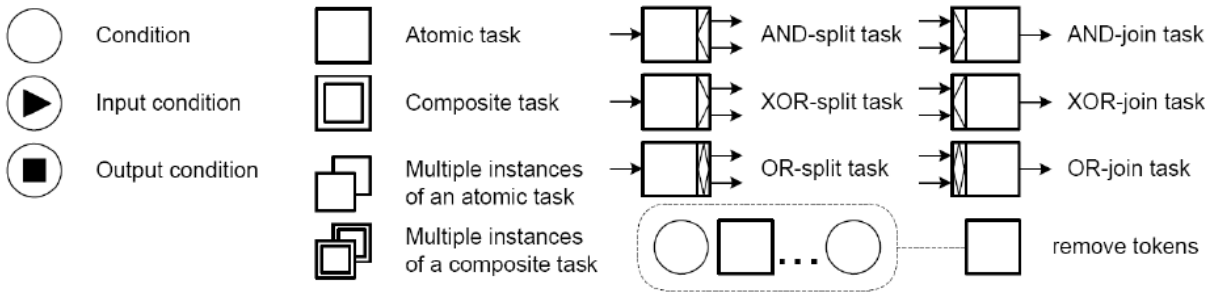


Figure 3 YAWL control-flow concepts

3.2.3.5 Abstract Process Execution Language (APEL) UJF

The APEL (for Abstract Process Engine Language) [62] is an activity-based process modeling language proposed by the Adele Team at UJF. It has a high level formalism for modeling processes and a flexible execution system supporting the dynamic evolution of processes (models or instances).

The formalism contained in APEL has the following concepts: Activity, Product, Port, Resource and Dataflow. An activity is a step in the workflow during which an action is performed. Products are objects (e.g., documents, data) produced, transformed or consumed by activities. Ports are the activity interfaces and they define and control the products that are expected and/or produced by activities. Ports are the only externally visible part of an activity (the encapsulation principle). Input ports perform an AND over their incoming data flows. That means that the port fires when at least one exemplar of each expected product is available in the port. Firing means that products are removed from the port and

the activity is started with that product set as input. When an output port is full it fires (either automatically or manually), which means products are sent to all destination ports. Dataflows describe how products are exchanged among activities. Resources are responsible for activities execution. The APEL metamodel is presented in Figure 4 [63].

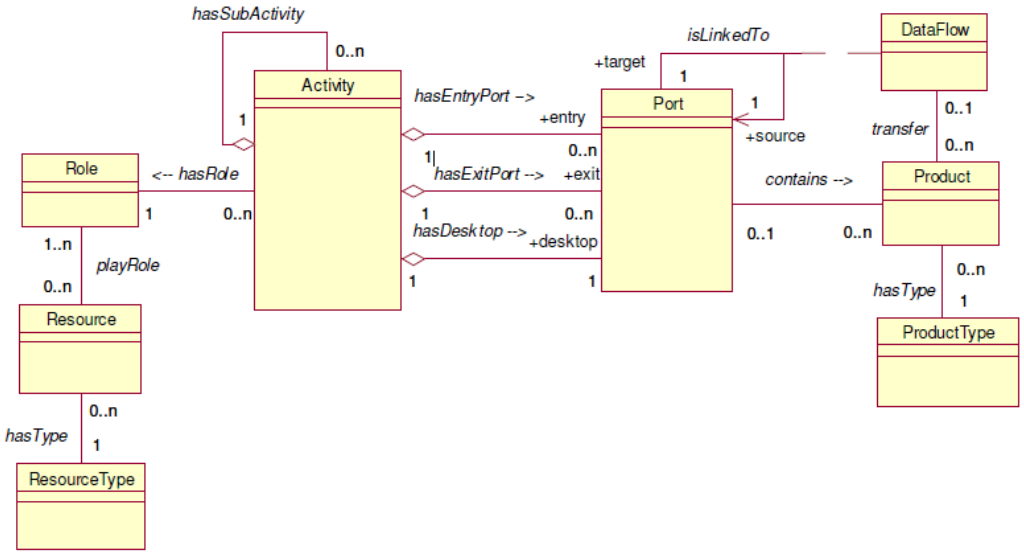


Figure 4 APEL Metamodel [63]

An illustrative example using APEL is presented in Figure 5, giving a reseller process example. The process starts with the Receiver Order activity in which a customer orders a product, followed by two activities executed in parallel. In the first one, the products are shipped using the Ship Products activity; in the other one, an invoice is sent to customer with the Send Invoice activity, and a payment is awaited in Receive Payment activity.

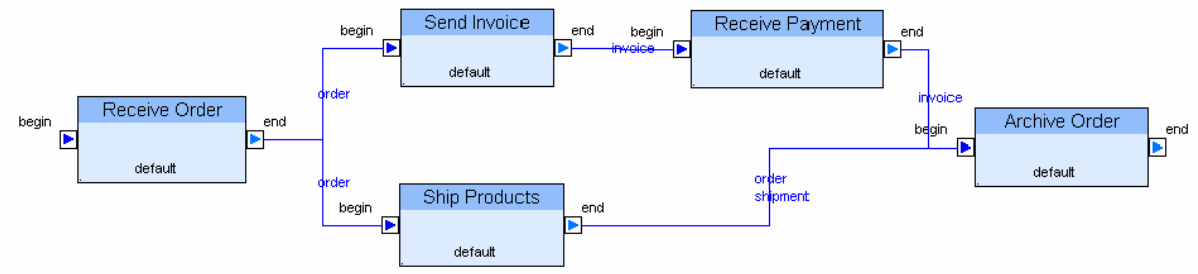


Figure 5 An APEL Control model [63]

Finally the Archive Order activity archives ordering process documents and the process finishes. The real nature of the activities is not defined in the model, and they can be manual, automatic, or a combination of both. The nature, format and content of the data circulating between activities are not defined either, due to the fact that APEL products are placeholders for information circulating in the process. An APEL product can be physical, an electronic document (e.g., a file, a Software configuration), or structured data of any kind (e.g., a data base record, a Java object, an XML document).

Section 4 Available OSS XPDL Editors Investigation

XPDL (XML Process Definition Language) is a standardized XML based formalism that allows the exchange of business process definitions among different modeling tools. XPDL encodes both the syntactic as well as the semantic parts of a business process model, i.e., it can capture both the graphical presentation of the model (including placement coordinates of its constituent components) and the execution semantics for how the components (processes) interact. So XPDL by being in XML format and be able to carry graphical representation data makes it the most suitable candidate for describing RFID Business Processes for the AspireRFID middleware. In this section we are going to investigate some candidate OSS XPDL editors that probably one of them could be used after extending and refactoring (to support RFID business processes) to make it part of the AspireRFID BPWME (Business Process Workflow Management Editor).

4.1 Enhydra JaWE

Enhydra JavaWE is an open source Java Workflow Process Editor that implements the WfMC specifications and uses XPDL for process representation. It allows viewing and editing of XPDL files that conform to the WfMC specifications, while it also supports their validation. Figure 6 is a snapshot from the Enhydra JaWS screen (www.enhydra.org).

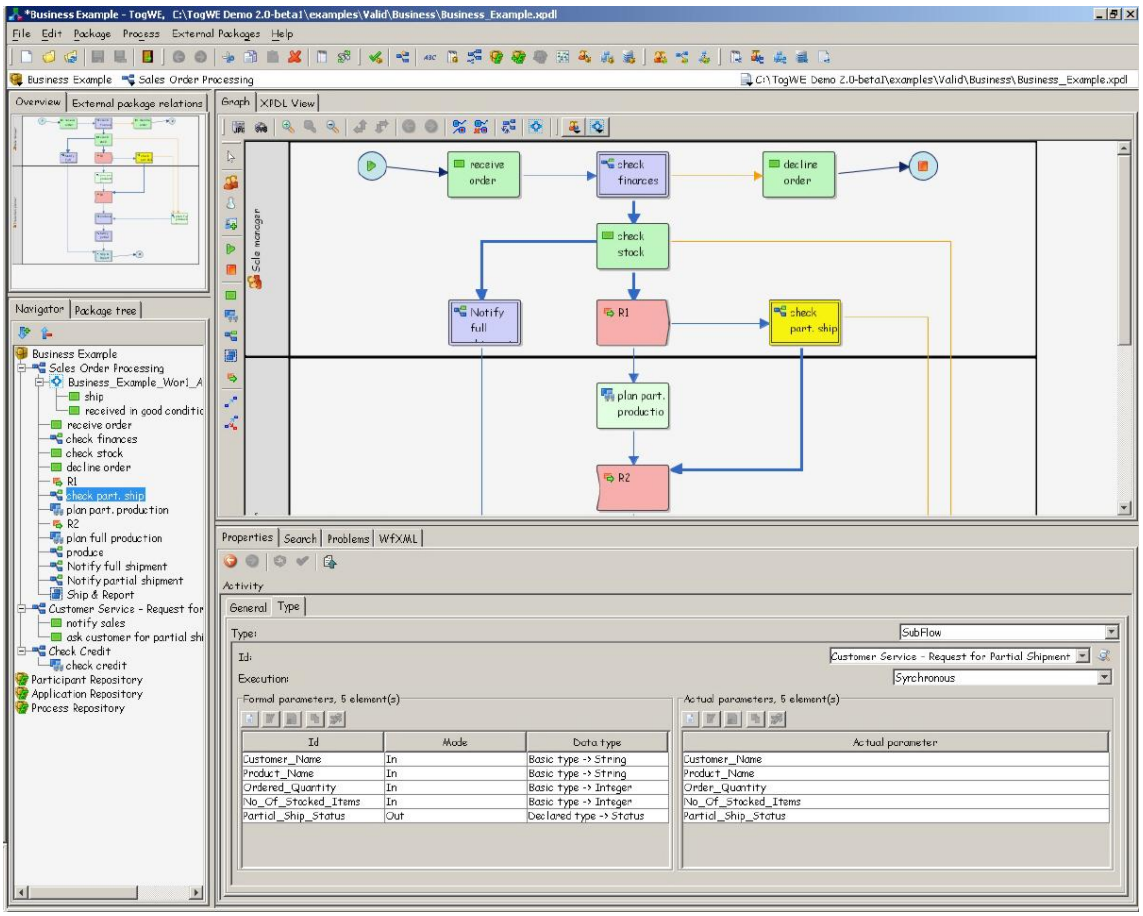


Figure 6 : Snapshot of Enhydra JaWE screen.

The general use of JaWE is shown in Figure 7. The tool allows the definition of a workflow process, which can then exported to a XPDL file. Alternatively XPDL files can be imported by the tool. The XPDL process definition can then be interpreted by a workflow engine.

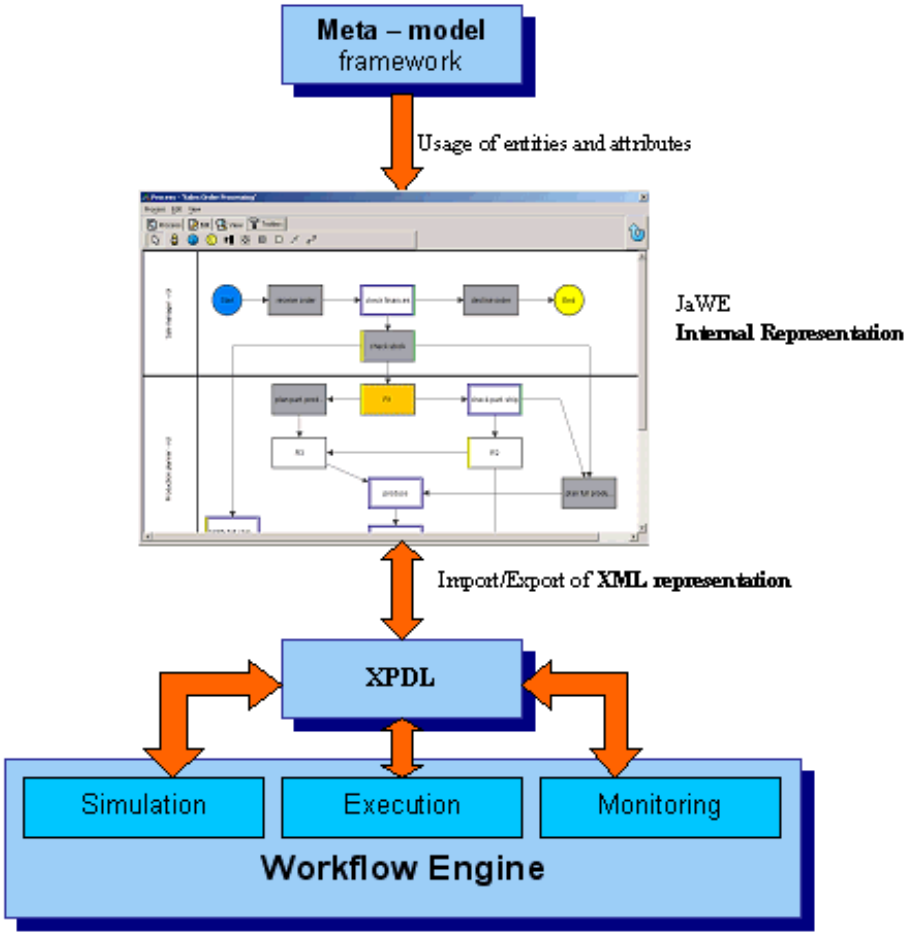


Figure 7 Use of JaWE

4.1.1 Pros

Advantages of JaWE include:

- It provides full XPDL 1.0 support
- Real time XPDL creation
- It is already a product, which can be downloaded for free (distributed under LGPL)

4.1.2 Cons

JaWE supports the XPDL specification, which is rather generic and complicated. Hence, on the negative side JaWE becomes a rather complicated tool to use. For

the purposes of the project a more compact and lightweight formalism would be appropriate.

4.2 Nova Bonita

Bonita consists of an open source BPM solution, now maintained and supported by BonitaSoft, with different components: Bonita Designer, Bonita Runtime and Bonita Console (Figure 8). These components can either be distributed as separate applications or as an integrated graphical environment for the development and execution of BPM based applications.

Figure 8 Screenshot of Bonita's Web Console

4.2.1 Pros

- BPM Designer available as an Eclipse plug-in
- BPM Designer also available as a standalone desktop application
- Nova Bonita has its own execution engine (runtime)
- Execution runtime available is Open source
- Integration of execution engine with its Eclipse plugin
- A web console (Bonita console) for managing the execution engine
- LGPL License

4.2.2 Cons

- No full XPDL 1.0 support (does not support composite data types)
- Designer uses of Java Swing instead of Eclipse's SWT

4.3 Eclipse Java Workflow Tooling

JWT is a set of tools for developing, deploying and testing workflows. JWT provides an adaptable framework for different graphical representations and XML notations, as well as different workflow engines. JWT is actually an ongoing project that aims to provide generic tools for workflow engines both for build-time and runtime. In addition to the graphical editor, a snapshot of which is shown in Figure 9, JWT designs a set of generic APIs for allowing definition and administration of business processes.

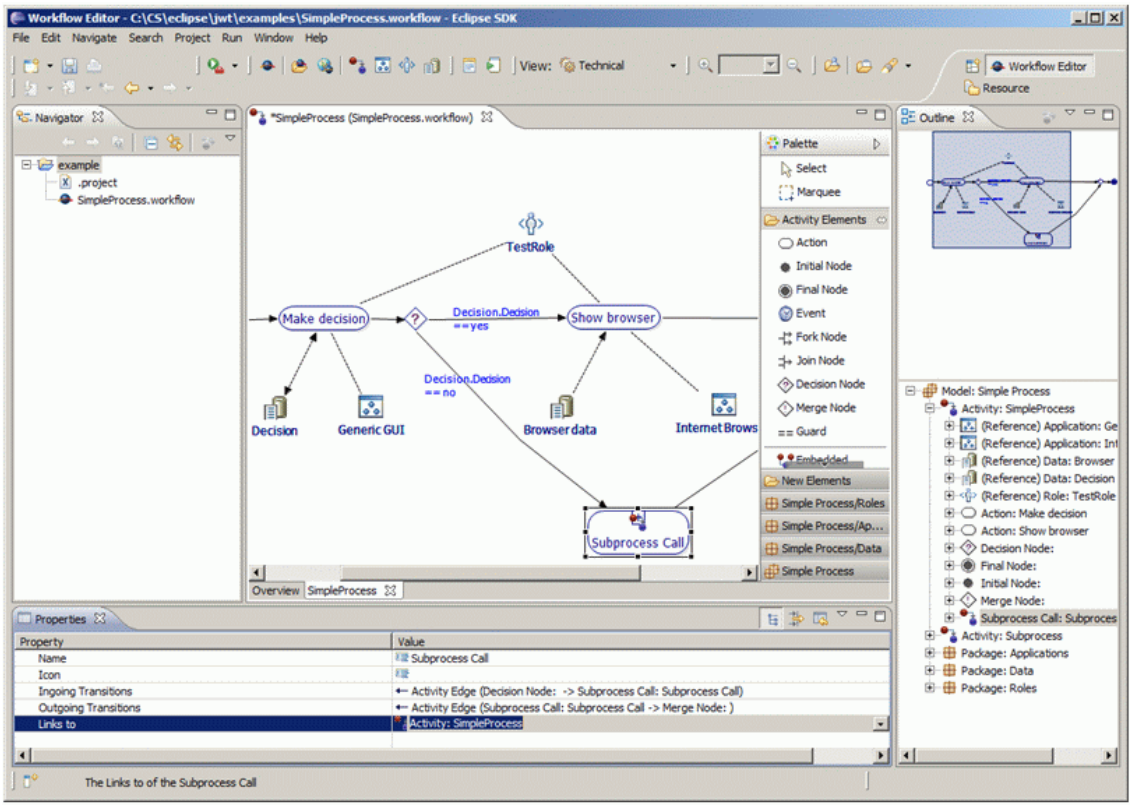


Figure 9 Screenshot of Eclipse JWT

4.3.1 Pros

- Java JWT is easy to use and allows graphical creation of business processes
- New elements creation support (without the need of programming)
- It supports the whole business process lifecycle (design, development, runtime, monitoring)
- It is an open and extensible framework
- The components of JWT can be used on their own or combined with an existing tooling
- Multiple views allow for business and technical specific representations of the modeled process as well as supporting different standards (e.g. UML Activity Diagram, EPC, ...)
- Transformations allow to import and export workflows in many different representations (e.g. XPDL, BPMN, BPEL, STP-IM, ...)

- It is an Eclipse plug-in

4.3.2 Cons

On the negative side:

- JWT is an ongoing project currently in its incubation phase
- It does not provide full XPDL 1.0 support
- No real time XPDL creation is supported
- JWT lacks the ability to convert from XPDL to JWT workflow

4.4 YAPROC

YAPROC (Yet Another Process) is an Eclipse platform plug-in (seen in Figure 10) which provides all the ability of developing standard workflow processes based on XPDL. YAPROC is built on top of Enhydra Shark and provides features such as reporting, runtime process viewer and activity management.

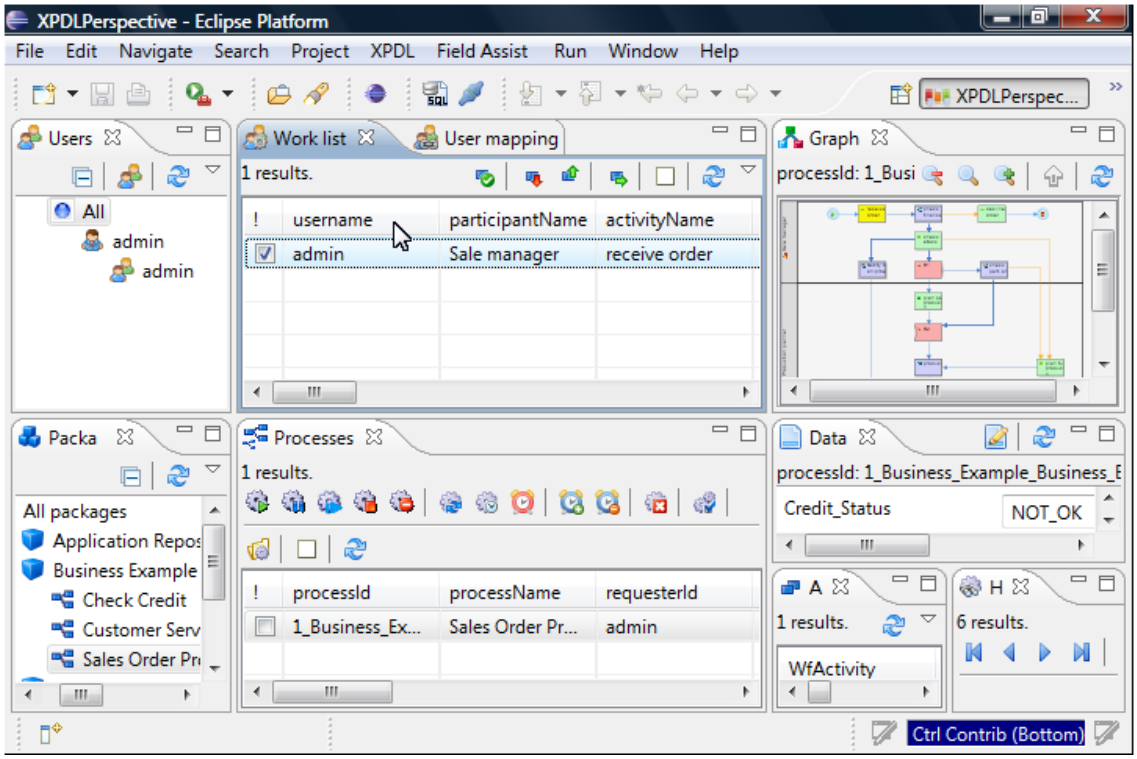


Figure 10 Screenshot of YAPROC

4.4.1 Pros

- Uses Enhydra JaWE for the workflow editor (with its pros and cons)
- XPDL 1.0 compatible
- Provides Managing i/f
- Eclipse plug-in
- LGPL V3.0 license

4.4.2 Cons

- Too bound with the Enhydra shark
- The cons of the Enhydra JaWE apply here also.

4.5 FOCAS

The ADELE team at the UJF develops a process and workflow engine called FOCAS (Framework for Orchestration, Aggregation and Composition of Services). Although not an XPDL editor, its concepts are shown here for comparative purposes. In FOCAS, Service compositions are described using a process to express control and data flow between services (i.e. service orchestration). FOCAS works as plug-in integrated to the Eclipse IDE (Figure 11) and constructed on top of CADSE (Computer Aided Domain Specific Engineering environments), which is an "intelligent" high level Eclipse workspace aware of the domain concepts, and knows the "best" way to map these concepts toward programming artifacts (e.g. files, folders, projects).

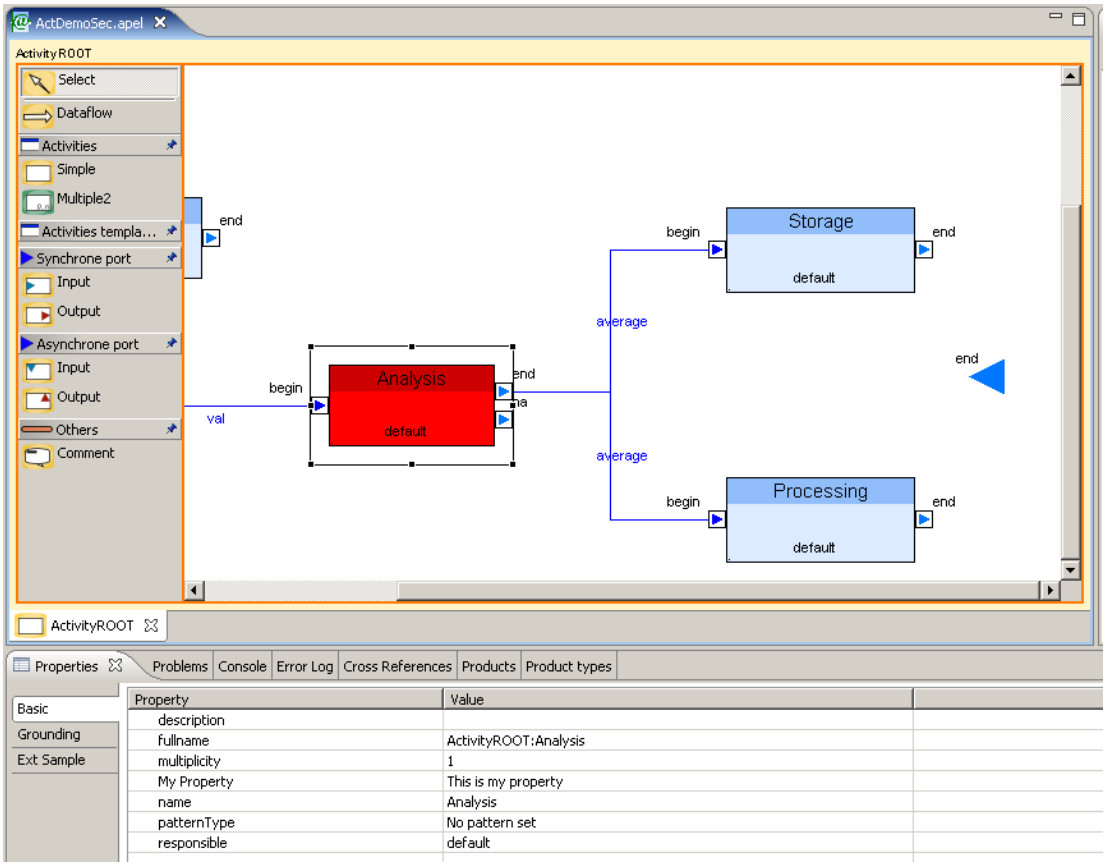


Figure 11 Screenshot of a process being edited in FOCAS

4.5.1 Pros

- Processes are designed using a high abstraction level language APEL.
- Non-functional properties can be added extending the basic environment, using annotation techniques

- Service platform independent
- Process patterns can be saved from a process model and used in several process definitions.
- New functional domains can be added to create richer process-based applications
- Processes can be enriched with behavior to adapt them to specific domains
- Extensible for Non-functional properties (security, distribution already available)

4.5.2 Cons

- Not based on XPDL
- No administration interface
- Need of implementing mappings between abstract APEL processes and concrete services

Section 5 Selecting the most Suitable for an RFID Language Specification

5.1 RFID Language Specification Requirements

The RFID Domain Specific language specifications requirements should follow the guidelines listed below:

- It should be as simple as possible,
- Be Domain-Oriented,
- Should be able to support RFID processes and Data,
- Be capable to describe a Composite/Elementary RFID Process as shown in at the example depicted in Figure 12 below,
- Be capable of carrying graphical representation data (e.g., XPDL),
- Be able to be mapped to XML for the AspireRFID programmable engine,
- Be XML Based,
- Amenable by Tools,
- Would be based on early experience with the AspireRFID tools / configurators / IDE,
- The Goal would be to become an Open Specification for RFID Solutions
- Should be standard and extensible,
- To allow stakeholders to build RFID solutions

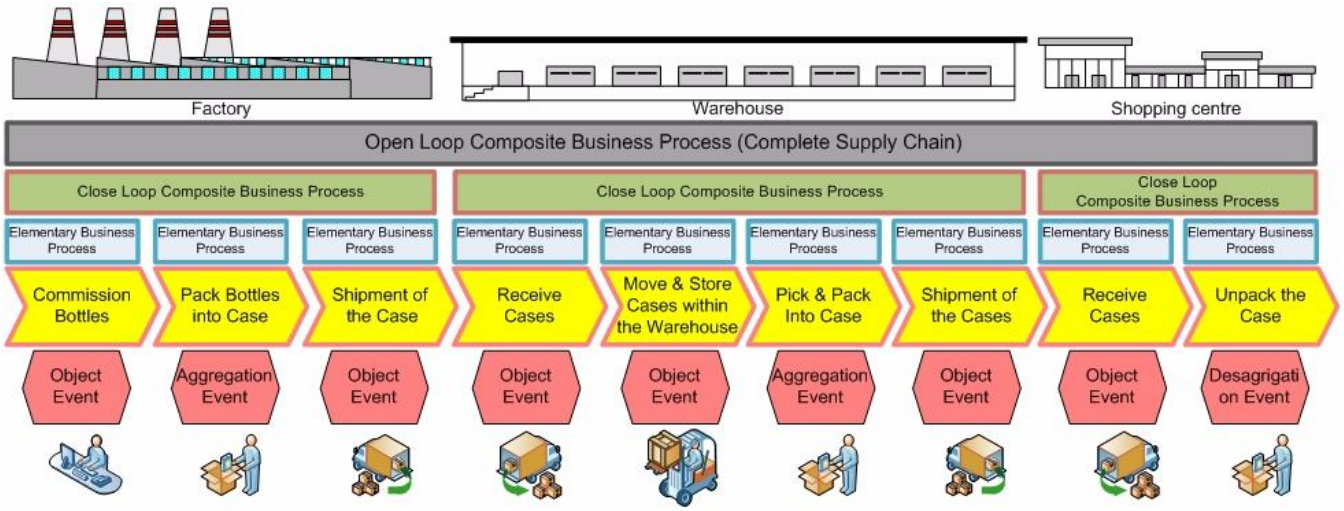


Figure 12 Composite/Elementary Business Process relationship/hierarchy

The Programmable Meta-Language as shown in Figure 13 below should also be a combination of the following Specifications:

- Physical reader Specs
- Logical Readers Specs
- ECSpecs
- Master Data Document
- Middleware Management/Configuration Data (BEG, Connector)
- Business Workflow data

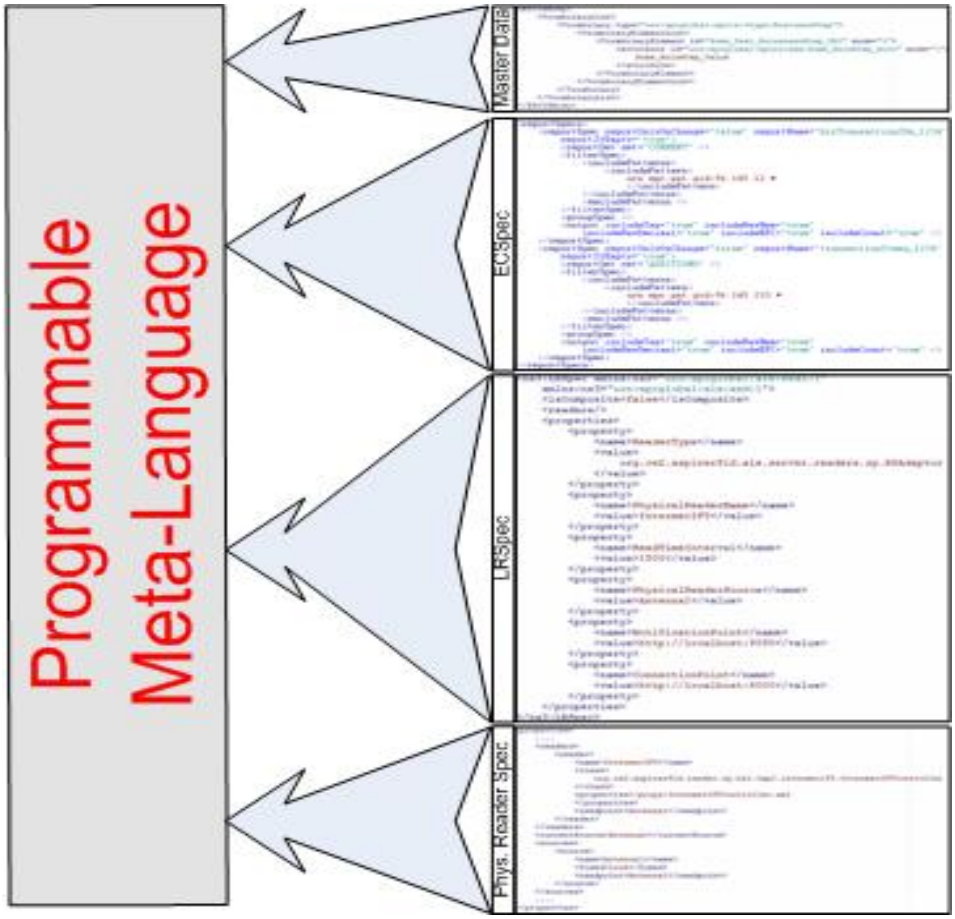


Figure 13 Programmable Meta-Language Data Support requirements.

All the above should be augmented with design data (e.g. XPDL) for the visualization of the RFID solution to the BPWME (Business Process Workflow Management Editor) tool.

5.2 Comparison of available Process Languages

In this section we are going to compare some of the available Business Process Languages so as to help us with the selection of the most suitable candidate for the AspireRFID middleware.

How Does XPDL Compare to BPEL?
BPEL and XPDL are entirely different yet complimentary standards. BPEL is an "execution language" designed to provide a definition of web services orchestration. It defines only the executable aspects of a process, when that process is dealing exclusively with web services and XML data. BPEL does not define the graphical diagram, human oriented processes, subprocess, and many other aspects of a modern business process: it simply was never defined to carry the business process diagram from design tool to design tool [57].

How Does YAWL Compare to BPEL?
YAWL is sometimes seen as an alternative to BPEL. A major advantage of BPEL is that it is driven by a standardization committee supported by several IT industry players. As a result, BPEL is supported by a significant number of tools (both proprietary and open-source) while YAWL has a single implementation at present. Also, several researchers have captured the formal semantics of subsets of BPEL in terms of various formalisms, including Petri nets, Process algebra and Finite state machine. This has paved the way for the development of static analysis tools for BPEL that can compete with the static analysis capabilities provided by the YAWL system. On the other hand, it has been noted that standard BPEL fails to support human tasks, that is, tasks that are allocated to human actors and that require these actors to complete actions, possibly involving a physical performance. A number of BPEL engines already provide extensions to BPEL for human tasks, but these extensions are yet to be standardized. In contrast, YAWL provides a unified interface for worklist services based on Web services standards. This interface allows developers to build their own worklist service to support human tasks according to their needs. In addition, the YAWL system comes with a default worklist service that supports several types of human task allocation and handling. Another advantage of YAWL is its support for the Workflow Patterns, although the gap between YAWL and BPEL in this respect may be reduced by new constructs that are included in BPEL version 2.0

How Does APEL Compare to XPDL?
APEL is a language used to express a process from an abstract view using a graphical formalism. However, in order to execute one specification in APEL, the language must be extended to add other important aspects of a process definition such as data and resources (humans and applications). Only one implementation of the editor and execution engine is available, and they are not available as open source. In the other hand, XDPL proposes a language much richer, with concepts supporting all aspects of a business process, concerns as subprocesses, data, applications, and humans resources are treated by XPDL. There are several implementations of editors and engines supporting XPDL, most of them are open source software. Moreover, XPDL is a standard defined by the WfMC while APEL is an individual initiative.

5.3 Decision

The workflow supporting languages that were presented in the previous sections 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. They have not tuned to a specific domain and as such it is hard to express concepts of a specialized domain. The same statement holds for the case of the ASPIRE RFID specific domain. Specialized concepts like RFID-based processes and RFID related data are rather cumbersome to express in general purpose workflow modeling languages.

From all the previously described languages, XPDL would probably be the most appropriate candidate to use taking in considerations the RFID language requirements described above. But following the same requirements due to the generality and complexity of XPDL for describing an RFID Open Loop Business Process we are forced to create a hybrid that would be simpler to understand, to describe its structure and specialized on RFID Business Processes. Furthermore there is not an Open Source XPDL Editor available to meet our needs for describing an RFID Business Process and the effort required for refactoring one of the editors described above to support such functionalities would be too much.

Therefore the need for a new special purpose modeling language that will be able to represent in a clean way RFID related concepts and a workflow editor that would accompany it becomes evident. The approach that was followed in the project was to design such a Domain Specific Language that would use some of the XPDL's notions, named APDL (ApireRFID Process Description Language), which is presented in the next section.

Section 6 AspireRFID Process Description Language (APDL)

APDL [83] is oriented towards solutions that comply with the EPCglobal Architecture. According to this architecture [10] the modules that compose an end-to-end RFID solution can be logically considered to be layered as depicted in Figure 14. 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.

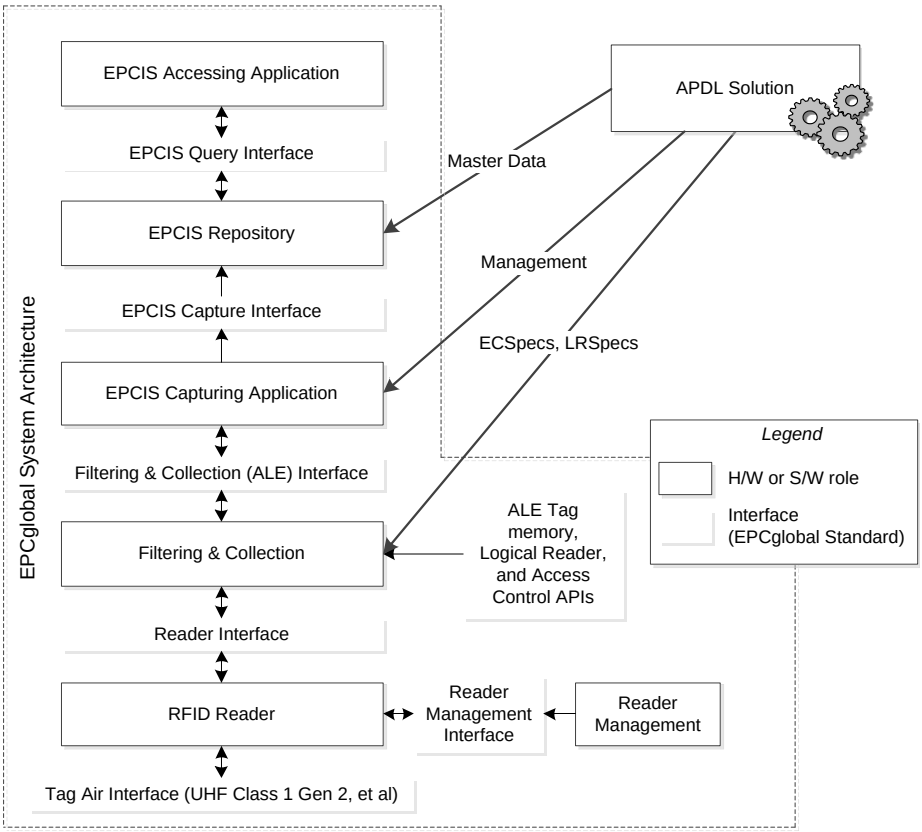


Figure 14 Middle Middleware configuration using APDL [10]

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

- Defining the event cycle specifications (ECSpecs) [1],
- Defining the Logical Reader specifications (LRSpecs) [1],
- And, finally, providing the EPCIS with the required Master Data (EPCIS Master Data Document) that partially manages how Application Level Events (ALE) will be stored in the EPCIS repository [9].

6.1 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) module one must configure the ECSpec, which is a complex type that describes an Event Cycle [1] 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 (EPC Information Sharing) layer Master Data Vocabularies are defined. The Master Data Vocabularies contain additional data that provides the necessary context for interpreting Event Data [9]. 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 order to integrate the Information Sharing layer with the F&C layer (Figure 14), 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 [1]) from the Filtering and collection module. Sources of data include filtered, collected EPC 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 [9].

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 15 and Figure 12, 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).

6.2 Defining APDL's Business Process

Figure 15 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.

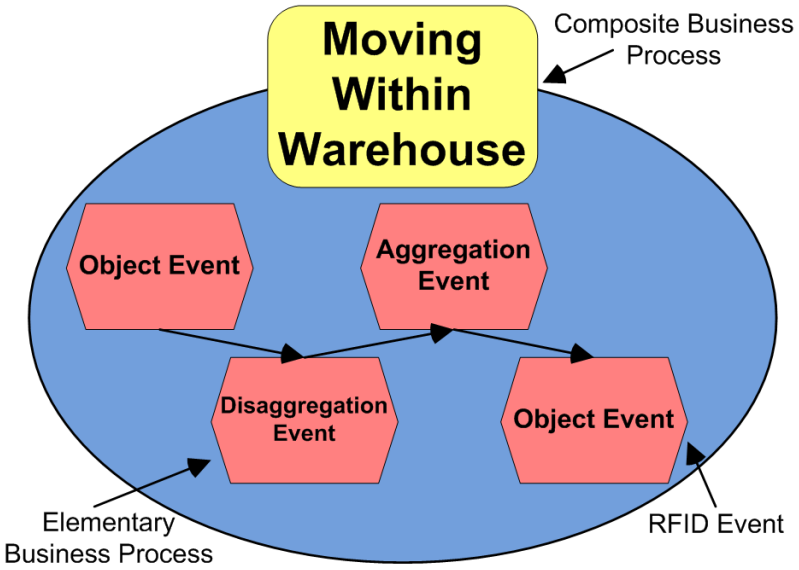


Figure 15 Decomposing an Inter-enterprise Business process

Figure 12 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 12 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 12, 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.

6.3 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 1 maps these associations.

R = Required O = Optional	ObjectEvent	AggregationEvent	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) [13][9]

Taking into consideration the above table we have bound the required ECReports [20] that need to be produced (defined at the ECSpec) from the F&C layer so as to be captured from the BEG Layer and eventually generate the equivalent EPC RFID Events as shown 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 required from the BEG layer so as to produce the equivalent RFID Event 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 [1].

- 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 [1].
- 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 [50] so as to describe the processes workflow and to achieve the visualization of the RFID solution.

6.4 Programmable Meta-Language Structure

From an implementation perspective, an APDL document is based on XML syntax. As far as it concerns its vocabulary, the namespaces in Table 4 are used.

Elements	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 16 below. The root element which contains the description of a complete supply chain management scenario is the Open Loop Composite Business Process (<*OLCBProc*/>).

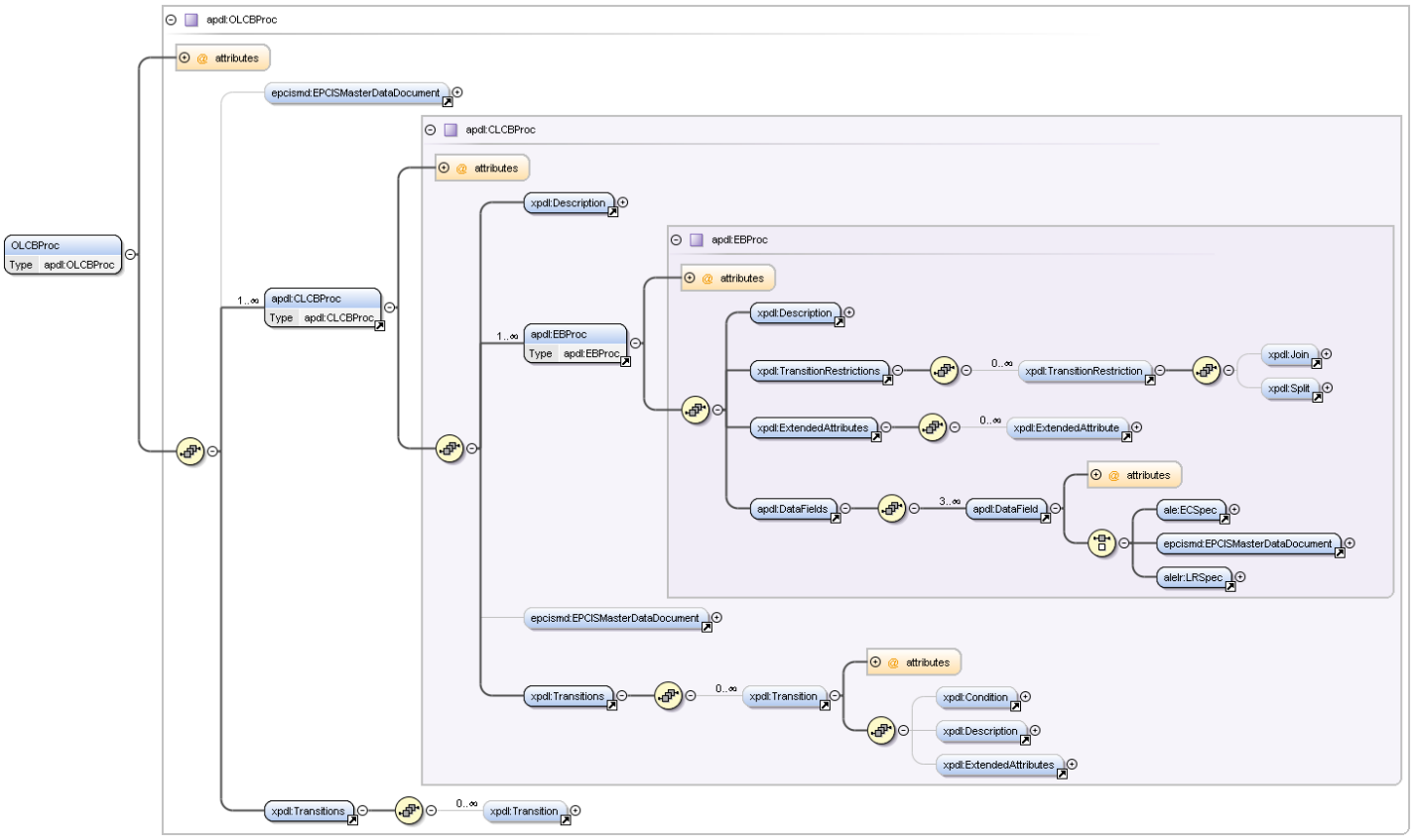


Figure 16 APDL's Schema graphical representation

"OLCBProc" contains a set of elements called Close Loop Composite Business Process (<*CLCBProc*/>) that are capable of describing a complete close loop supply chain scenario and the element of Transitions (<*Transitions*/>) which carries the Close Loop Composite Business processes context-related semantics description of Transitions between them which is based on the XPDL V1.0 specification [17].

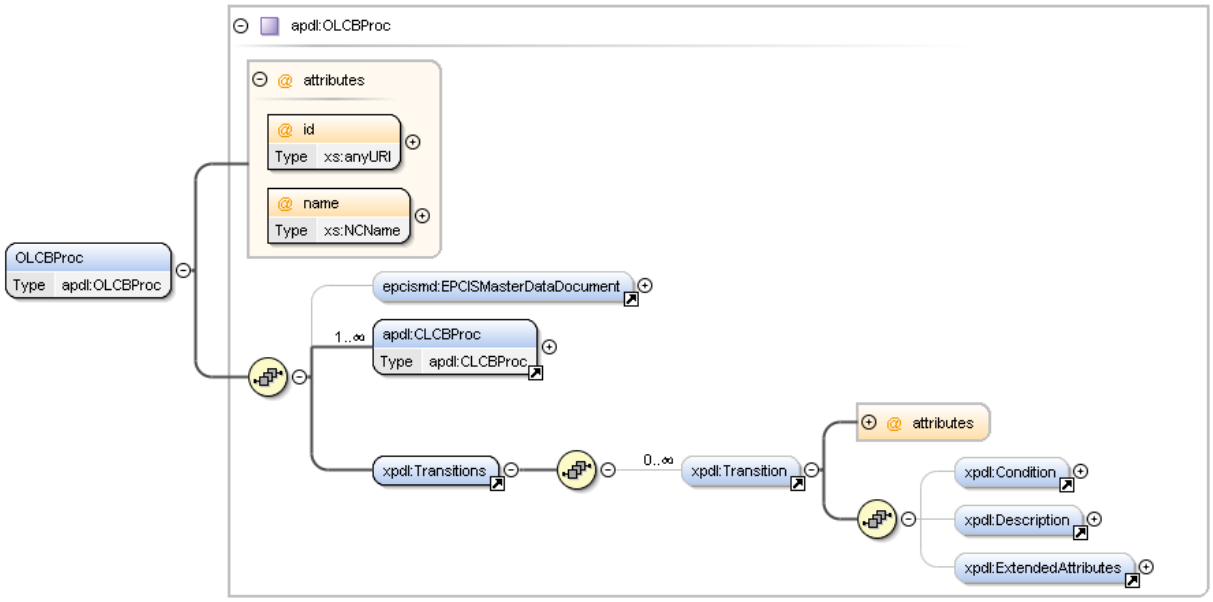


Figure 17 APDL' Schema design decomposition: OLCBProc

Each of the "CLCBProc" elements, shown in Figure 18, are consisted of a set Elementary Business Process (<*EBProc*/>) elements that describe the elementary Business Transactions, the CLCBProc's Master Data in the form of an EPCIS Master Data Document (<*epcismd:EPCISMasterDataDocument*/>) and the object of Transitions (<*Transitions*/>) which carries the Elementary Business Processes context-related semantics description of Transitions between them which is based on the XPDL V1.0 specifications [50]. 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.

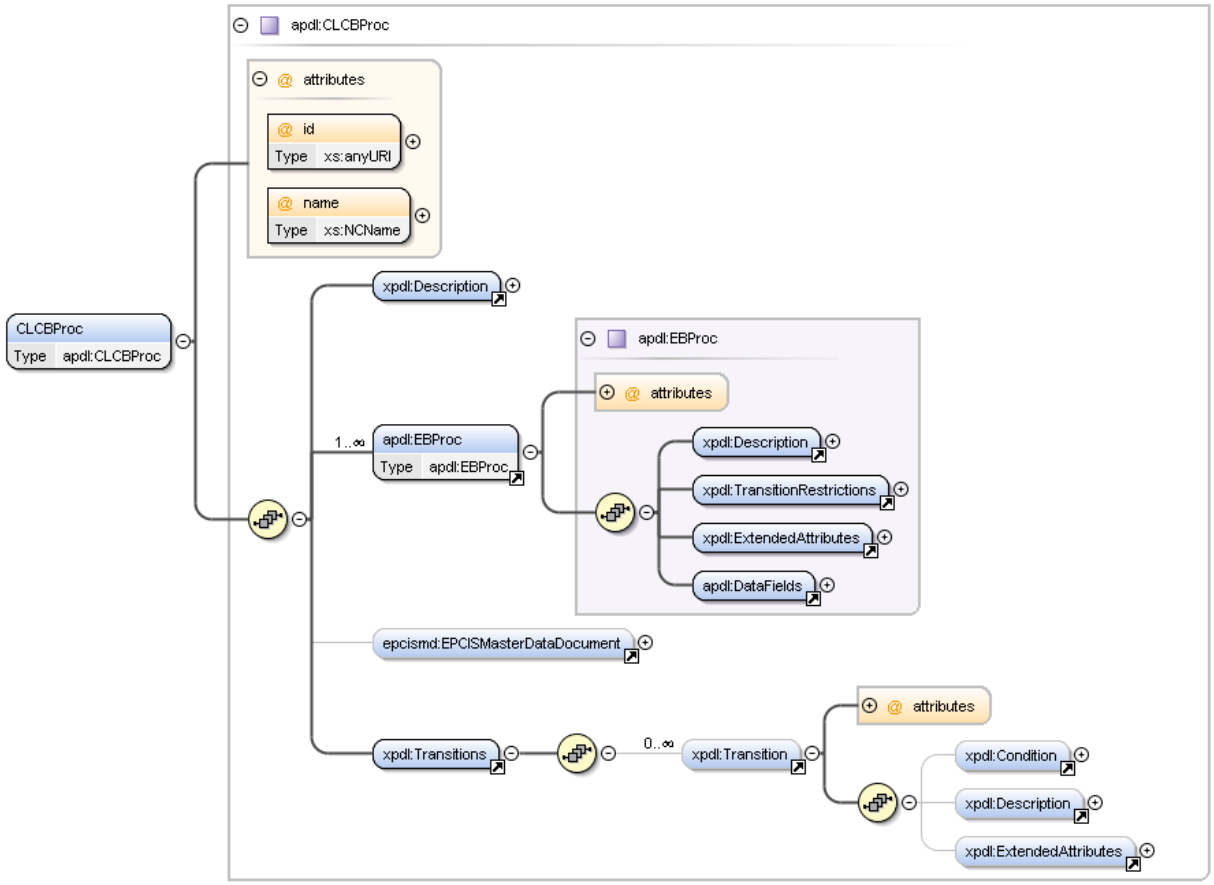


Figure 18 APDL' Schema design decomposition: CLCBProc

The EBProc element, shown in Figure 19, is the most important in the APDL specification since it contains the elementary business process description appropriately modeled following the RFID Business logic we have defined (see Section 6.3). This element contains:

- The TransitionRestrictions (<*xpdl:TransitionRestrictions*/>) element [50],
- The ExtendedAttributes (<*xpdl:ExtendedAttributes*/>) element, Which is used to store the basic required RFID middleware configuration data.
- A set of DataFields (<*apdl:DataFields*/>), that include the required:
    a. ECSpec (<*ale:ECSpec*/>),
    b. LRSpec (<*alelr:LRSpec*/>), and
    c. Master Data (<*epcismd:EPCISMasterDataDocument*/>) for describing a specific elementary business process transaction following the RFID Business logic.
- And finally a description (<*xpdl:Description*/>) element.

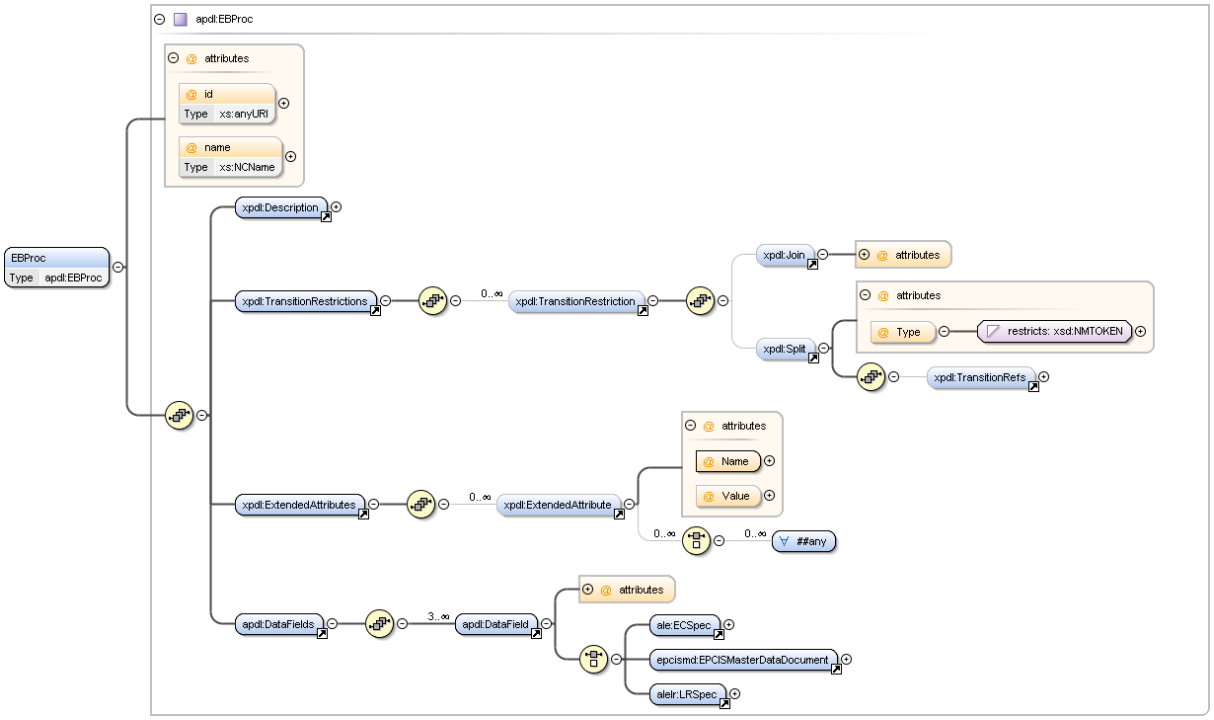


Figure 19 APDL' Schema design decomposition: EBProc

The complete APDL xml schema can be found at the APPENDIX II at the end of this document.

6.5 Programmable Meta-Language Definition

6.5.1 APDL Main Elements

The tree main elements that construct the AspireRFID Process Description Language are the:

- OLCBProc (Open Loop Composite Business Process)
- CLCBProc (Close Loop Composite Business Process)
- And EBProc (Elementary Business Process)

And are described in detail below.

6.5.1.1 Open Loop Composite Business Process (OLCBProc)

The *OLCBProc,* show in Table 5, is the parent element that is capable of describing a complete RFID enabled supply chain management scenario from the manufacturer to the retailer (see Figure 12 above). It is consisted of a list of CLCBProc elements, their Transitions and a Master Data Document that contains global company data. At this level the "epcismd:EPCISMasterDataDocument" is used to store only Standard Vocabulary types and more specifically:

- the "urn:epcglobal:epcis:vtype:BusinessStep",
- the "urn:epcglobal:epcis:vtype:Disposition" and
- the "urn:epcglobal:epcis:vtype:BusinessTransactionType" types.

Finally the *OLCBProc*'s name and id conclude this element's description.

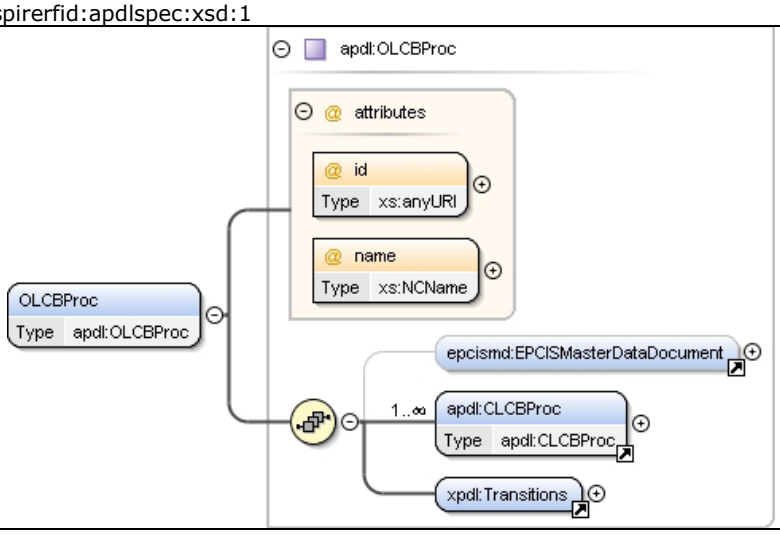
<table>
<tr><td>Namespace</td><td colspan="2">urn:ow2:aspirerfid:apdlspec:xsd:1</td></tr>
<tr><td>Diagram</td><td colspan="2"> 
</td></tr>
<tr><td>Type</td><td colspan="2">apdl:OLCBProc</td></tr>
<tr><td>Properties</td><td colspan="2">content: complex</td></tr>
<tr><td>Model</td><td colspan="2">epcismd:EPCISMasterDataDocument{0,1} , apdl:CLCBProc+ , xpdl:Transitions</td></tr>
<tr><td>Children</td><td>Name</td><td>Description</td></tr>
<tr><td></td><td>apdl:CLCBProc</td><td>Close Loop Composite Business Process (see 6.5.1.2)</td></tr>
<tr><td></td><td>epcismd:EPCISMasterDataDocument</td><td>Stores the urn:epcglobal:epcis:vtype:BusinessStep, the urn:epcglobal:epcis:vtype:Disposition, and the urn:epcglobal:epcis:vtype:BusinessTransactionType vocabularies [9].</td></tr>
<tr><td></td><td>xpdl:Transitions</td><td>CLCBProc Transitions (see 6.5.2)</td></tr>
<tr><td>XSD Element</td><td colspan="2"><pre>
<xs:complexType name="OLCBProc">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="1"
      ref="epcismd:EPCISMasterDataDocument" />
    <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>
</pre></td></tr>
<tr><td>Instance</td><td colspan="2"><pre>
<apdl:OLCBProc id="" name="">
  <epcismd:EPCISMasterDataDocument creationDate="" schemaVersion="">
    {0,1}</epcismd:EPCISMasterDataDocument>
    <apdl:CLCBProc id="" name="">{1,unbounded}</apdl:CLCBProc>
    <xpdl:Transitions>{1,1}</xpdl:Transitions>
</apdl:OLCBProc>
</pre></td></tr>
<tr><td>Attributes</td><td colspan="2"><table>
<tr><th>QName</th><th>Type</th><th>Fixed</th><th>Default</th><th>Use</th><th>Description</th></tr>
<tr><td>id</td><td>xs:anyURI</td><td></td><td></td><td>required</td><td>The OLCBProc's ID</td></tr>
<tr><td>name</td><td>Xs:NCName</td><td></td><td></td><td>required</td><td>The OLCBProc's Name</td></tr>
</table></td></tr>
<tr><td>Source</td><td colspan="2"><pre><xs:element name="OLCBProc" type="apdl:OLCBProc" /></pre></td></tr>
</table>

Table 5 OLCBProc element

6.5.1.2 Close Loop Composite Business Process (CLCBProc)

The *CLCBProc* element, shown in Table 6, is the responsible element for describing a complete close loop supply chain management scenario that is comprised from many elementary business processes (see Figure 12 above), their transitions and a Master Data Document that contains Local company data. At this level the "epcismd:EPCISMasterDataDocument" is used to store only User Vocabulary types and more specifically:

- the "urn:epcglobal:epcis:vtype:BusinessLocation" and

- the "urn:epcglobal:epcis:vtype:ReadPoint" types.

Also the name and the id is required to distinct the Close Loop Composite Business Processes one from another.

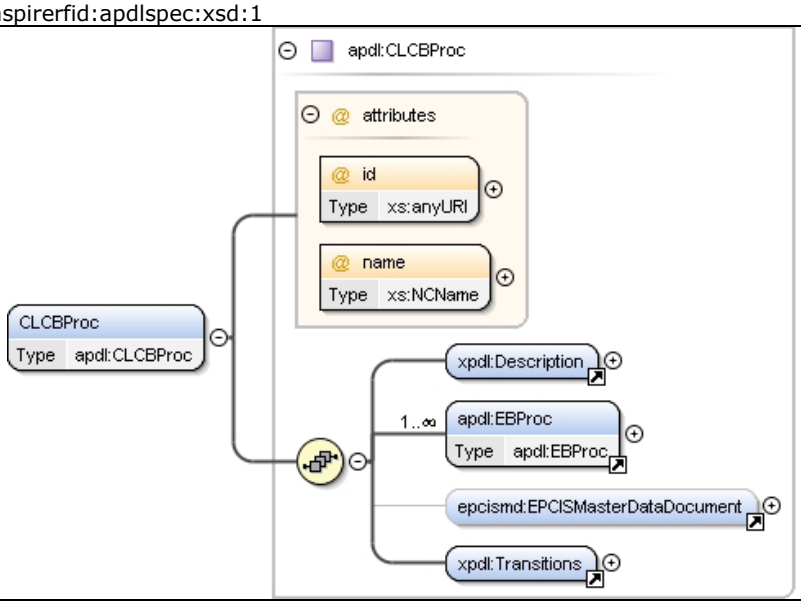
<table>
<tr><td>Namespace</td><td colspan="2">urn:ow2:aspirerfid:apdlspec:xsd:1</td></tr>
<tr><td>Diagram</td><td colspan="2"> 
</td></tr>
<tr><td>Type</td><td colspan="2">apdl:CLCBProc</td></tr>
<tr><td>Properties</td><td colspan="2">content: complex</td></tr>
<tr><td>Used by</td><td colspan="2">Complex Type apdl:OLCBProc</td></tr>
<tr><td>Model</td><td colspan="2">xpdl:Description , apdl:EBProc+ , epcismd:EPCISMasterDataDocument{0,1} , xpdl:Transitions</td></tr>
<tr><td>Children</td><td colspan="2"><table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>apdl:EBProc</td><td>Elementary Business Process (see 6.5.1.3)</td></tr>
<tr><td>epcismd:EPCISMasterDataDocument</td><td>The EPCISMasterDataDocument is used here to store only the urn:epcglobal:epcis:vtype:BusinessLocation and the urn:epcglobal:epcis:vtype:ReadPoint EPCIS Vocabulary types [9].</td></tr>
<tr><td>xpdl:Transitions</td><td>CLCBProc Transitions (see 6.5.2)</td></tr>
<tr><td>xpdl:Description</td><td>The description of the CLCBProc (see 6.5.3.1)</td></tr>
</table></td></tr>
<tr><td>XSD Element</td><td colspan="2"><pre>
<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>
</pre></td></tr>
<tr><td>Instance</td><td colspan="2"><pre>
<apdl:CLCBProc id="" name="">
	<xpdl:Description>{1,1}</xpdl:Description>
	<apdl:EBProc id="" name="">{1,unbounded}</apdl:EBProc>
	<epcismd:EPCISMasterDataDocument creationDate="" schemaVersion="">
		{0,1}</epcismd:EPCISMasterDataDocument>
		<xpdl:Transitions>{1,1}</xpdl:Transitions>
</apdl:CLCBProc>
</pre></td></tr>
<tr><td>Attributes</td><td colspan="2"><table>
<tr><th>QName</th><th>Type</th><th>Fixed</th><th>Default</th><th>Use</th><th>Description</th></tr>
<tr><td>id</td><td>xs:anyURI</td><td></td><td></td><td>required</td><td>The CLCBProc's ID</td></tr>
<tr><td>name</td><td>Xs:NCName</td><td></td><td></td><td>required</td><td>The CLCBProc's Name</td></tr>
</table></td></tr>
<tr><td>Source</td><td colspan="2"><pre><xs:element name="CLCBProc" type="apdl:CLCBProc" /></pre></td></tr>
</table>

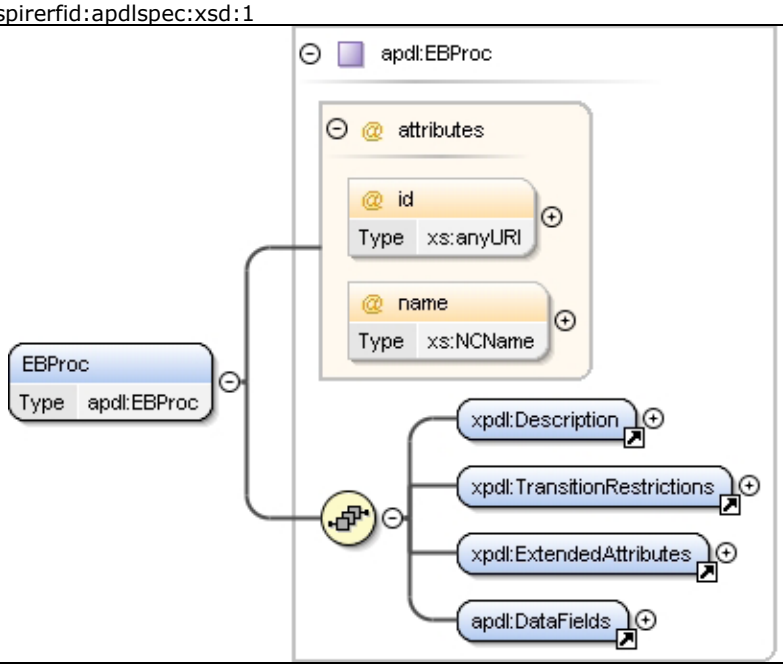
Table 6 CLCBProc element

6.5.1.3 Elementary Business Process (EBProc)

The EBProc element, shown in Table 7 below, is the most important element in the APDL definition because it is responsible for describing an Elementary Business Process (see Figure 12 above) by carrying all the information in a way the AspireRFID middleware requires and "understands" so as to get programmed.

This element more specifically contains:

- The EBProc's ID, defined as the elements attribute, which will be the Transaction URI (physical primary key) that will be stored to the company's Master Data Transaction vocabulary.
- The EBProc's name, defined also as the elements attribute, which will be used at the Transaction vocabulary as an attribute.
- The description (<*xpdl:Description*/>) of the Elementary Business Process that will also be used at the Transaction vocabulary as an attribute
- A TransitionRestrictions (<*xpdl:TransitionRestrictions*/>) element, containing a set of TransitionRestriction (<*xpdl:TransitionRestriction*/>) elements[50].This element is used to describe the transitions between the EBProc elements.
- An ExtendedAttributes (<*xpdl:ExtendedAttributes*/>) element, containing a set of ExtendedAttribute (<*xpdl:ExtendedAttribute*/>) elements. This element is used in order to store the basic configuration data, for instance the ECSpec Subscription URI, and the element's graphical representation data (x/y coordinates). In particular, the following key-value pairs are stored: XOffset, YOffset, CellHeight, CellWidth which stores the required variables for the workflow graphical representation. The set of ExtendedAttributes also includes a set of configuration variables. 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,
    e. the EPCIS query interface endpoint, and
    f. the BEG Engine's management interface Endpoint.
- And Finally a set of DataFields (<*apdl:DataFields*/>), that include the required RFID middleware specification files which are:
    - The EPCISMasterDataDocument (<*epcismd:EPCISMasterDataDocument*/>) which will carry the Transaction description which will be stored at the Company's Master Data Transaction vocabulary bind with the EBPrps's ID described above.
    - The ECSpec file (<*ale:ECSpec*/>) for setting the F&C's server filtering configurations with using at the F&C's defining command the name the EBProc ID.
    - And the LRSpec (<*alelr:LRSpec*/>) for setting the F&C's server Logical Readers configurations using as Logical reader name the name included at the defined ECSpec.

Namespace	urn:ow2:aspirerfid:apdlspec:xsd:1
Diagram	
Type	apdl:ELCBProc
Properties	content: complex
Used by	Complex Type apdl:CLCBProc
Model	xpdl:Description , xpdl:TransitionRestrictions , xpdl:ExtendedAttributes , apdl:DataFields

Children

Name	Description
apdl:DataFields	The EBProc's Data Fields (see 6.5.1.3.3)
xpdl:ExtendedAttributes	The EBProc's Extended Attributes (see 6.5.1.3.2)
xpdl:TransitionRestrictions	The EBProc's Transition Restrictions (see6.5.1.3.1)
xpdl:Description	The description of the CLCBProc (see 6.5.3.1)

XSD Element

```
<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>
```

Instance

```
<apdl:EBProc id="" name="">
        <xpdl:Description>{1,1}</xpdl:Description>
        <xpdl:TransitionRestrictions>{1,1}</xpdl:TransitionRestrictions>
        <xpdl:ExtendedAttributes>{1,1}</xpdl:ExtendedAttributes>
        <apdl:DataFields>{1,1}</apdl:DataFields>
</apdl:EBProc>
```

Attributes

QName	Type	Fixed	Default	Use	Description
id	xs:anyURI			required	The EBProc's ID
name	Xs:NCName			required	The EBProc's Name

Source

```
<xs:element name="EBProc" type="apdl:EBProc" />
```

Table 7 EBProc element

6.5.1.3.1 TransitionRestrictions Element

Transaction Restriction which is borrowed from the XPDL V1.0 specifications [50] (Section 7.5.8) shown in Table 8 below provides further restrictions and context-related semantics description of Transitions. In general, normal transition restrictions may be declared at the level of the EBP boundary within the surrounding process, whereas specialized flow conditions (subflow, or the

internal part of a route activity) operate "internal" to the EBP (but may reference activities within the surrounding process definition). Further information about the Transition Restrictions usage and schema can be found at the XPDL V1.0 [50] Section 7.5.8.

Namespace	http://www.wfmc.org/2002/XPDL1.0
Diagram	
Properties	content: complex
Used by	Complex Type apdl:EBProc Element xpdl:Activity
Model	xpdl:TransitionRestriction
Children	xpdl:TransitionRestriction

XSD Element	`<xsd:element name="TransitionRestrictions">`
	`<xsd:complexType>`
	`<xsd:sequence>`
	`<xsd:element ref="xpdl:TransitionRestriction" minOccurs="0" maxOccurs="unbounded"/>`
	`</xsd:sequence>`
	`</xsd:complexType>`
	`</xsd:element>`

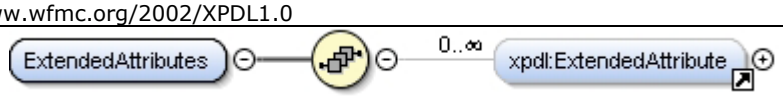
Instance	`<xpdl:TransitionRestrictions>`
	`<xpdl:TransitionRestriction>{0,unbounded}</xpdl:TransitionRestriction>`
	`</xpdl:TransitionRestrictions>`

Source	`<xsd:element name="TransitionRestrictions">`
	`<xsd:complexType>`
	`<xsd:sequence>`
	`<xsd:element ref="xpdl:TransitionRestriction" minOccurs="0" maxOccurs="unbounded" />`
	`</xsd:sequence>`
	`</xsd:complexType>`
	`</xsd:element>`

Table 8 TransitionRestrictions element

6.5.1.3.2 ExtendedAttributes Element

The ExtendedAttributes element, shown in Table 9 below is comprised from a List of elements named ExtendedAttribute and are used to store the EBProc basic configuration Data, like the ECSpec Subscription URI and BEG listening Port, and the EBProc graphical representation data.

Namespace	http://www.wfmc.org/2002/XPDL1.0
Diagram	
Properties	content: complex
Used by	Elements xpdl:DataField, xpdl:Transition Complex Type apdl:EBProc
Model	xpdl:ExtendedAttribute
Children	xpdl:ExtendedAttribute | The EBProc Extended Attribute (see 6.5.1.3.2.1)

XSD Element	`<xsd:element name="ExtendedAttributes">`
	`<xsd:complexType>`
	`<xsd:sequence>`
	`<xsd:element ref="xpdl:ExtendedAttribute" minOccurs="0" maxOccurs="unbounded" />`
	`</xsd:sequence>`
	`</xsd:complexType>`
	`</xsd:element>`

Instance	`<xpdl:ExtendedAttributes>`
	`<xpdl:ExtendedAttribute Name="" Value="">{0,unbounded}</xpdl:ExtendedAttribute>`
	`</xpdl:ExtendedAttributes>`

Source	`<xsd:element name="ExtendedAttributes">`

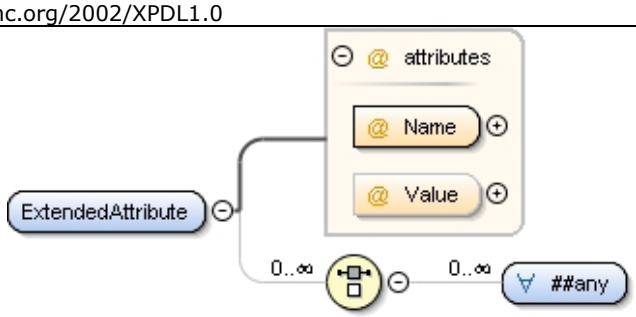
	<xsd:complexType> <xsd:sequence> <xsd:element ref="xpdl:ExtendedAttribute" minOccurs="0" maxOccurs="unbounded" /> </xsd:sequence> </xsd:complexType> </xsd:element>

Table 9 ExtendedAttributes element

6.5.1.3.2.1 ExtendedAttribute Element

The ExtendedAttribute element, shown in Table 10 below, contains a name/value pair that main objective is to store the following EBProc's attributes:

- The EBProc's Coordinates ExtendedAttribute which are responsible for the Business Process Workflow Management Editor graphical representation by providing:
  - The element's *XOffset* in the workspace,
  - the element's *YOffset* in the workspace,
  - the element's *CellHeight* in the workspace, and
  - the element's *CellWidth* in the workspace.
- And the AspireRFID "Basic Configuration" ExtendedAttribute which are used to store various attributes required by the AspireRfid middleware to configure it like:
  - The EC Spec Subscription URI "*ECSpecSubscriptionURI*" required by the Ale Configuration client to define where the generated reports should be delivered.
  - the ALE Client endpoint "*AleClientEndPoint*",
  - the ALE Logical Reader Client endpoint "*AleLrClientEndPoint*",
  - the EPCIS Capture interface endpoint "*EpcisClientCaptureEndPoint*",
  - the EPCIS query interface endpoint "*EpcisClientQueryEndPoint*", and
  - the BEG Engine's management interface Endpoint "*BegEngineEndpoint*".

Namespace	http://www.wfmc.org/2002/XPDL1.0					
Diagram						
Properties	content: complex					
	mixed: true					
Used by	Element xpdl:ExtendedAttributes					
Model	ANY element from ANY namespace					
Attributes	**QName**	**Type**	**Fixed**	**Default**	**Use**	**Description**
	Name	xsd:NMTOKEN			required	The Extended Attribute's name
	Value	xsd:string			required	The Extended Attribute's value
XSD Element	<xsd:element name="ExtendedAttribute"> <xsd:complexType mixed="true"> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:any minOccurs="0" maxOccurs="unbounded"					

	processContents="lax" /> </xsd:choice> <xsd:attribute name="Name" type="xsd:NMTOKEN" use="required" /> <xsd:attribute name="Value" type="xsd:string" /> </xsd:complexType> </xsd:element>
Source	<xsd:element name="ExtendedAttribute"> <xsd:complexType mixed="true"> <xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" /> </xsd:choice> <xsd:attribute name="Name" type="xsd:NMTOKEN" use="required" /> <xsd:attribute name="Value" type="xsd:string" /> </xsd:complexType> </xsd:element>

Table 10 ExtendedAttribute element

6.5.1.3.3 DataFields Element

The DataFields element shown in Table 11 below is a list of elements (DataField) that contains all the required AspireRFID specification files (ECSpecs, LRSpecs, Master Data) for describing a specific Elementary Business Process (Transaction Event).

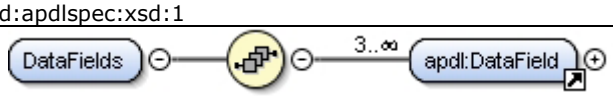
Namespace	urn:ow2:aspirerfid:apdlspec:xsd:1
Diagram	
Properties	content: complex
Used by	Complex Type apdl:EBProc
Model	apdl:DataField{3,unbounded}
Children	apdl:DataField | The EBProc Data list (see 6.5.1.3.3.1)
XSD Element	<xs:element name="DataFields"> <xs:complexType> <xs:sequence> <xs:element minOccurs="3" maxOccurs="unbounded" ref="apdl:DataField" /> </xs:sequence> </xs:complexType> </xs:element>
Instance	<apdl:DataFields> <apdl:DataField name="" type="">{3,unbounded}</apdl:DataField> </apdl:DataFields>
Source	<xs:element name="DataFields"> <xs:complexType> <xs:sequence> <xs:element minOccurs="3" maxOccurs="unbounded" ref="apdl:DataField" /> </xs:sequence> </xs:complexType> </xs:element>

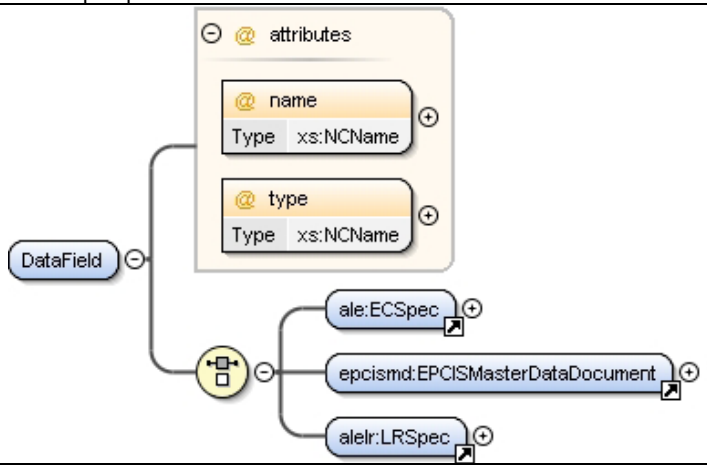
Table 11 DataFields element

6.5.1.3.3.1 DataField Element

Each DataField element can contain specification files the combination wich is capable of describing an Elementary Business Process. A DataField can either be:

- EPC ECSpec document [1],
- EPCIS Master Data Document [9] or
- EPC LRSpec document [1]

Except from the above specification files, which are required, the DataField element also carries the element's ID, Name and Type which are optional and used for future XPDL compatibility purposes.

<table>
<tr><td>Namespace</td><td>urn:ow2:aspirerfid:apdlspec:xsd:1</td></tr>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td>Properties</td><td>content: complex</td></tr>
<tr><td>Used by</td><td>Element apdl:DataFields</td></tr>
<tr><td>Model</td><td>ale:ECSpec | epcismd:EPCISMasterDataDocument | alelr:LRSpec</td></tr>
<tr><td>Children</td><td>
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>ale:ECSpec</td><td>EBProc's ECSpec (see 6.5.1.3.4.2)</td></tr>
<tr><td>alelr:LRSpec</td><td>EBProc's LRSpec (see 6.5.1.3.4.3)</td></tr>
<tr><td>epcismd:EPCISMasterDataDocument</td><td>EBProc's EPCIS Master Data Document (see 6.5.1.3.4.1)</td></tr>
</table>
</td></tr>
<tr><td>XSD Element</td><td>
<pre>
<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>
</pre>
</td></tr>
<tr><td>Instance</td><td>
<pre>
<apdl:DataField name="" type="">
	<ale:ECSpec creationDate="" includeSpecInReports="false"
		schemaVersion="">
		{1,1}</
		ale:ECSpec>
		<epcismd:EPCISMasterDataDocument creationDate=""
			schemaVersion="">
			{1,1}</
			epcismd:EPCISMasterDataDocument>
			<alelr:LRSpec creationDate=""
				schemaVersion="">{1,1}</alelr:LRSpec>
</apdl:DataField>
</pre>
</td></tr>
<tr><td>Attributes</td><td>
<table>
<tr><th>QName</th><th>Type</th><th>Fixed</th><th>Default</th><th>Use</th><th>Description</th></tr>
<tr><td>name</td><td>xs:NCName</td><td></td><td></td><td>required</td><td>DataField Name</td></tr>
<tr><td>type</td><td>Xs:NCName</td><td></td><td></td><td>required</td><td>DataField Type</td></tr>
</table>
</td></tr>
<tr><td>Source</td><td>
<pre>
<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" />
</pre>
</td></tr>
</table>

	<xs:attribute name="type" use="required" type="xs:NCName" /> </xs:complexType> </xs:element>

Table 12 DataField element description

6.5.1.3.4 EBProc's Complex Data Types

APDL uses complex data typed from the EPC specifications [11] that are inborn supported from the specification language. These data types are described below

6.5.1.3.4.1 EPCISMasterDataDocument

APDL uses the epcismd:EPCISMasterDataDocument element [9] to store the Company's Master Data. These Master Data are stored to the EPCIS Repository with the form of specific Vocabularies. Vocabularies are used extensively within EPCIS to model conceptual and physical entities that exist in the real world. EPC Specifications distinguish these vocabularies into two logical units that follow different patterns in the way they are defined and extended over time. These two kinds are the Standard and the User Vocabulary.

> "A **Standard Vocabulary** represents a set of Vocabulary Elements whose definition and meaning must be agreed to in advance by trading partners who will exchange events using the vocabulary. Standard Vocabulary elements tend to be defined by organizations of multiple end users, such as EPCglobal, industry consortia outside EPCglobal, private trading partner groups, and so on. The master data associated with Standard Vocabulary elements are defined by those same organizations, and tend to be distributed to users as part of a specification or by some similar means." [9]

> "A **User Vocabulary** represents a set of Vocabulary Elements whose definition and meaning are under the control of a single organization. User Vocabulary elements are primarily defined by individual end user organizations acting independently. The master data associated with User Vocabulary elements are defined by those same organizations, and are usually distributed to trading partners through the EPCIS Query Control Interface or other data exchange / data synchronization mechanisms. New vocabulary elements within a given User Vocabulary are introduced at the sole discretion of an end user, and trading partners must be prepared to respond accordingly." [9]

The following table (Table 13) summarizes the vocabulary types defined in the EPCIS Specs:

Vocabulary Type	**User / Standard**	**URI**
BusinessStepID	Standard	urn:epcglobal:epcis:vtype:BusinessStep
DispositionID	Standard	urn:epcglobal:epcis:vtype:Disposition
BusinessTrasactionTypeID	Standard	urn:epcglobal:epcis:vtype:BusinessTransactionType
BusinessLocationID	User	urn:epcglobal:epcis:vtype:BusinessLocation
ReadPointID	User	urn:epcglobal:epcis:vtype:ReadPoint
BusinessTransaction	User	urn:epcglobal:epcis:vtype:BusinessTransaction

Table 13 EPCIS Vocabulary Types

APDL has used the Standard/User definition to hierarchically place the required vocabularies within its structure. So "*BusinessStepID*", "*DispositionID*" and "*BusinessTrasactionTypeID*" was placed at the OLCBProc level as they keep "Standard" information to be used from the whole supply chain. "*BusinessLocationID*" and "*ReadPointID*" has been placed at the CLCBProc level as it keeps data relative to a specific location that the CLCBProc describes and can be used from all the EBProces within it. And finally "*BusinessTransaction*" Vocabulary Type has been placed at the EBProc level as it keeps Data concerning a specific Business Step.

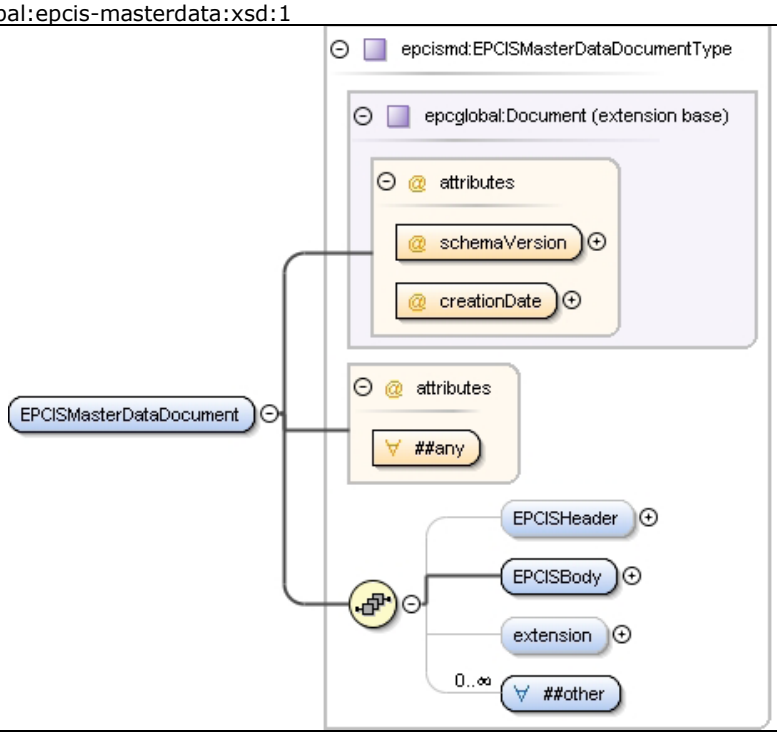
<table>
<tr><td>Namespace</td><td colspan="5">urn:epcglobal:epcis-masterdata:xsd:1</td></tr>
<tr><td>Diagram</td><td colspan="5"> 
</td></tr>
<tr><td>Type</td><td colspan="5">epcismd:EPCISMasterDataDocumentType</td></tr>
<tr><td>Properties</td><td colspan="5">content: complex</td></tr>
<tr><td rowspan="2">Used by</td><td colspan="5">Element apdl:DataField</td></tr>
<tr><td colspan="5">Complex Types apdl:CLCBProc, apdl:OLCBProc</td></tr>
<tr><td>Model</td><td colspan="5">EPCISHeader{0,1} , EPCISBody , extension{0,1} , ANY element from ANY namespace OTHER than 'urn:epcglobal:epcis-masterdata:xsd:1'</td></tr>
<tr><td>Children</td><td colspan="5">EPCISBody, EPCISHeader, extension</td></tr>
<tr><td>XSD Element</td><td colspan="5"><xsd:element name="EPCISMasterDataDocument" type="epcismd:EPCISMasterDataDocumentType" /></td></tr>
<tr><td>Instance</td><td colspan="5"><epcismd:EPCISMasterDataDocument creationDate="" schemaVersion="">
<EPCISHeader>{0,1}</EPCISHeader>
<EPCISBody>{1,1}</EPCISBody>
<extension>{0,1}</extension>
</epcismd:EPCISMasterDataDocument></td></tr>
<tr><td>Attributes</td><td>QName</td><td>Type</td><td>Fixed</td><td>Default</td><td>Use</td></tr>
<tr><td></td><td>creationDate</td><td>xsd:dateTime</td><td></td><td></td><td>required</td></tr>
<tr><td></td><td>schemaVersion</td><td>xsd:decimal</td><td></td><td></td><td>required</td></tr>
<tr><td>Source</td><td colspan="5"><xsd:element name="EPCISMasterDataDocument" type="epcismd:EPCISMasterDataDocumentType" /></td></tr>
</table>

Table 14 EPCISMasterDataDocument element

APDL for being able to set up the EBProc's Event description uses a compatible with the AspireRFID architecture (for the EPCIS layer) EPCISMasterDataDocument element [9] shown in Table 14 above which carries the information shown in Table 15 below [80].

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

Table 15 Business Transaction Attributes

6.5.1.3.4.2 ECSpec

APDL for being able to set up the EBProc's Filtering Needs uses a compatible with the AspireRFID architecture (for the F&C layer) ECSpec [1] (*<ale:ECSpec/>*) element shown in Table 16 below which will be configured the way D4.3a [80] defines at section 7.2.1.1 without the report name IDs concatenated to them. For the report names ID at the configuration time the EBProc's ID will be used.

```
<xs:element name="ECSpec" type="ale:ECSpec"></xs:element>
```

Table 16 ECSpec element

6.5.1.3.4.3 LRSpec

APDL for being able to set up the EBProc's Logical Readers uses a compatible with the AspireRFID architecture (for the F&C layer) dynamic definition LRSpec element [1] (*<alelr:LRSpec/>*) shown in Table 17 below. For the name of the logical reader the DataField's name would be used that is defining the reader.

```
<xs:element name="LRSPec" type="alelr:LRSpec"></xs:element>
```

Table 17 LRSpec element

6.5.2 Transitions

The Transitions description philosophy is borrowed from the XPDL V1.0 (XML Process Description Language). Transition Information is defined as the possible transitions between activities which in our case are the Elementary Business Process (EBProc) and the conditions that enable or disable them (the transitions) during workflow execution. Further control and structure restrictions may be expressed in the EBProc (defined as activity in XPDL) definition (see TransitionRestrictions). More details about Transition Information and Restrictions can be found at the XPDL specifications V1.0 [50] Section 7.6 and 7.5.8 respectively.

Namespace	http://www.wfmc.org/2002/XPDL1.0
Diagram	
Properties	content: complex
Used by	Complex Types apdl:CLCBProc, apdl:OLCBProc
Model	xpdl:Transition
Children	xpdl:Transition

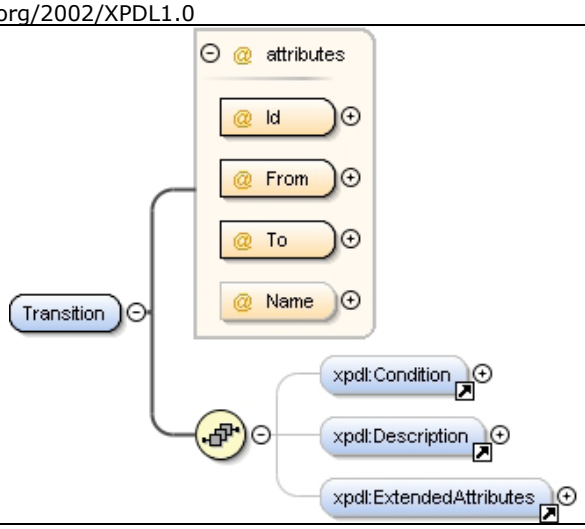
XSD Element	`<xsd:element name="Transitions">`
	`  <xsd:complexType>`
	`  <xsd:sequence>`
	`  <xsd:element ref="xpdl:Transition" minOccurs="0"`
	`  maxOccurs="unbounded" />`
	`  </xsd:sequence>`
	`  </xsd:complexType>`
	`</xsd:element>`

Instance	`<xpdl:Transitions>`
	`  <xpdl:Transition From="" Id="" Name=""`
	`To="">{0,unbounded}</xpdl:Transition>`
	`</xpdl:Transitions>`

Source	`<xsd:element name="Transitions">`
	`  <xsd:complexType>`
	`  <xsd:sequence>`
	`  <xsd:element ref="xpdl:Transition" minOccurs="0"`
	`  maxOccurs="unbounded" />`
	`  </xsd:sequence>`
	`  </xsd:complexType>`
	`</xsd:element>`

Table 18 XPDL Transitions element

Elementary Business Processes are related to one another via flow control conditions (transition information). XPDL 1.0 defines for each individual transition to have three elementary properties, the from-EBP, the to-EBP and the condition under which the transition is made. Transition from one EBP to another may be conditional (involving expressions which are evaluated to permit or inhibit the transition) or unconditional. The transitions within a process may result in the sequential or parallel operation of individual EBPs within the process. The information related to associated split or join conditions is defined within the appropriate EBP (see TransitionRestrictions), split as a form of "post EBP" processing in the from-EBP, join as a form of "pre-EBP" processing in the to-EBP. This approach allows the workflow control processing associated with process instance thread splitting and synchronization to be managed as part of the associated EBP, and retains transitions as simple route assignment functions. The scope of a particular transition is local to the process definition, which contains it and the associated activities.

Namespace	http://www.wfmc.org/2002/XPDL1.0
Diagram	
Properties	content: complex
Used by	Element xpdl:Transitions
Model	xpdl:Condition{0,1} , xpdl:Description{0,1} , xpdl:ExtendedAttributes{0,1}
Children	xpdl:Condition, xpdl:Description, xpdl:ExtendedAttributes

XSD Element	`<xsd:element name="Transition">`
	`  <xsd:complexType>`
	`  <xsd:sequence>`
	`  <xsd:element ref="xpdl:Condition" minOccurs="0" />`
	`  <xsd:element ref="xpdl:Description" minOccurs="0" />`
	`  <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0" />`
	`  </xsd:sequence>`
	`  <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required" />`
	`  <xsd:attribute name="From" type="xsd:NMTOKEN" use="required" />`
	`  <xsd:attribute name="To" type="xsd:NMTOKEN" use="required" />`
	`  <xsd:attribute name="Name" type="xsd:string" />`
	`  </xsd:complexType>`
	`</xsd:element>`

Instance	`<xpdl:Transition From="" Id="" Name="" To="">`
	`  <xpdl:Condition Type="">{0,1}</xpdl:Condition>`
	`  <xpdl:Description>{0,1}</xpdl:Description>`
	`  <xpdl:ExtendedAttributes>{0,1}</xpdl:ExtendedAttributes>`
	`</xpdl:Transition>`

Attributes

QName	Type	Fixed	Default	Use
From	xsd:NMTOKEN			required
Id	xsd:NMTOKEN			required
Name	xsd:string			required
To	xsd:NMTOKEN			required

Source

```
<xsd:element name="Transition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xpdl:Condition" minOccurs="0" />
      <xsd:element ref="xpdl:Description" minOccurs="0" />
      <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required" />
    <xsd:attribute name="From" type="xsd:NMTOKEN" use="required" />
    <xsd:attribute name="To" type="xsd:NMTOKEN" use="required" />
    <xsd:attribute name="Name" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
```

Table 19 XPDL Transition element

6.5.3 Basic Elements

6.5.3.1 Description

For describing the various elements APDL uses a simple string type element shown in Table 20 below named Description.

```
<xs:element name="Description" type="xs:string" />
```

Table 20 Description element

Name	Description
Description	Description of various elements of Type sting

Table 21 Description element description

Section 7 Tools that facilitates APDL's Usability

7.1 Business Process Workflow Management Editor (BPWME)

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 20.

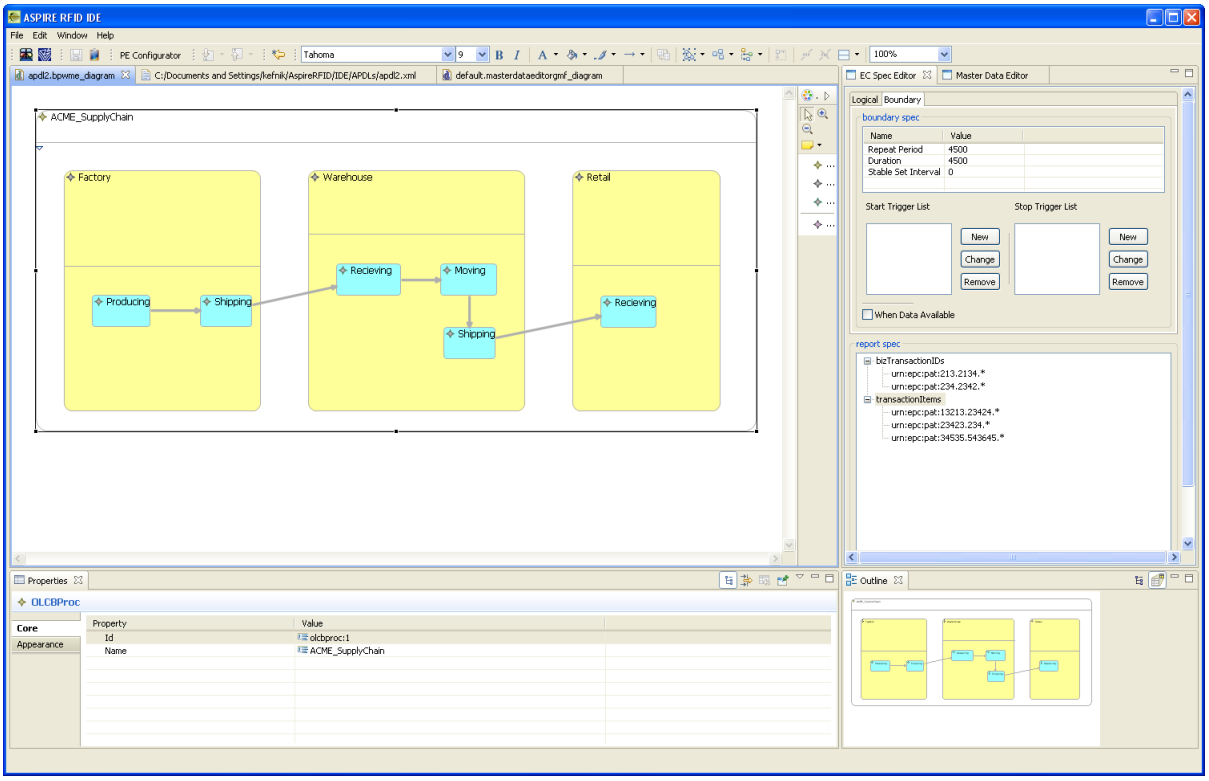


Figure 20 BPWME Designer Editor View

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.

In Figure 21 below we can see the Logical Reader Editing tab. There someone could define the Logical Readers that will be used at a CLCBProc and could be reused from the EBProces that belongs to it.

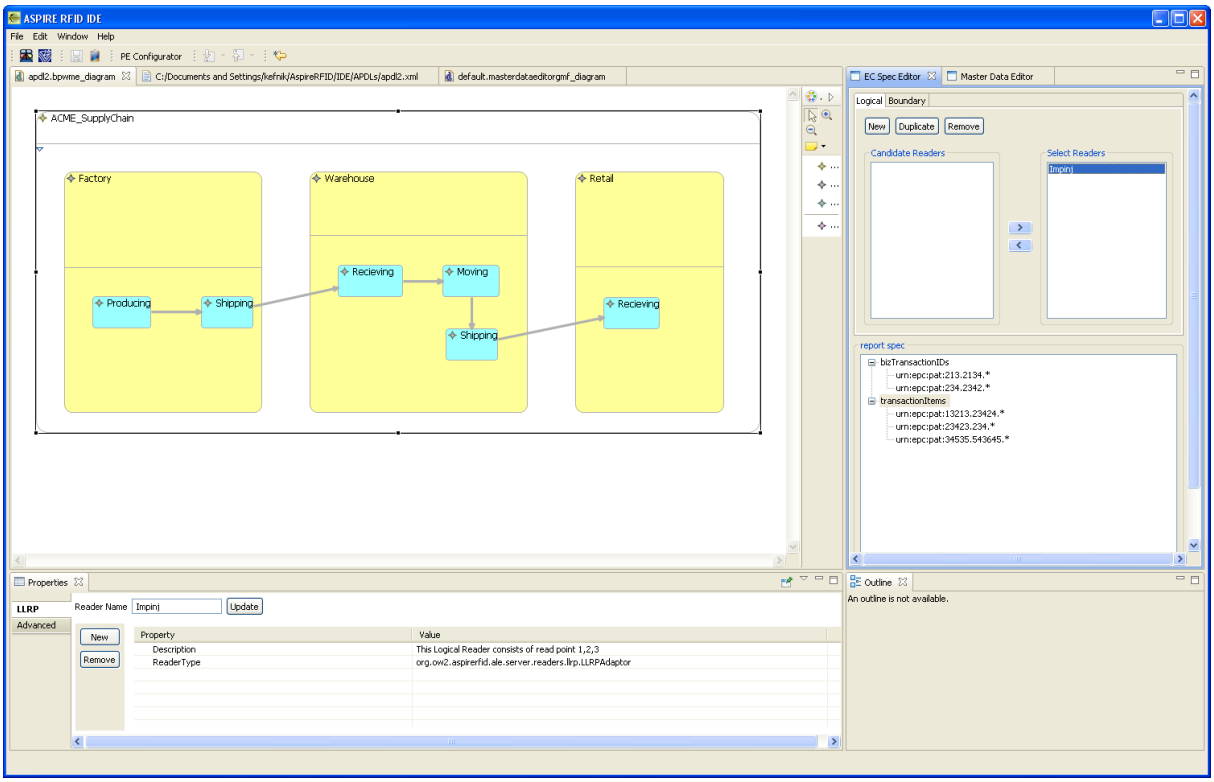


Figure 21 CLCBProc's available Logical Readers Editing

In Figure 22 below, at the lower left (properties) tab, we can see the Logical Reader Editing Advanced tab were someone can set up new "standardized" attributes that could be used at the different Readers configuration. This way if a company uses its own proprietary RFID Reader HAL it could extend the LRSpec configurator with additional attributes.

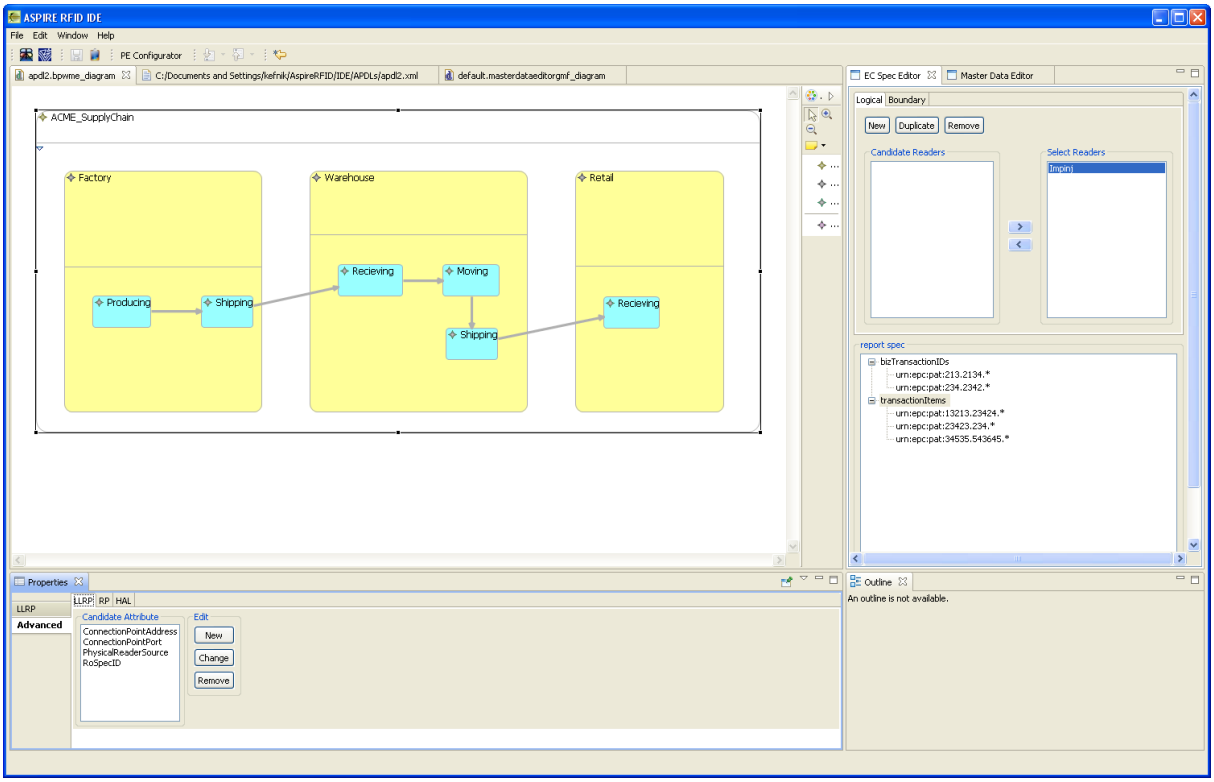


Figure 22 CLCBProc's Advanced Logical Readers configuration Editing

In Figure 23 below, at the upper right Tab, we can see the ECSpec configuration. Depending on the Event which we have chosen at the EBProc creation the equivalent reports filters are required from the user to be filled. These reports can be filed by clicking on the report name where at the Properties tab automatically the specific's report list appears.

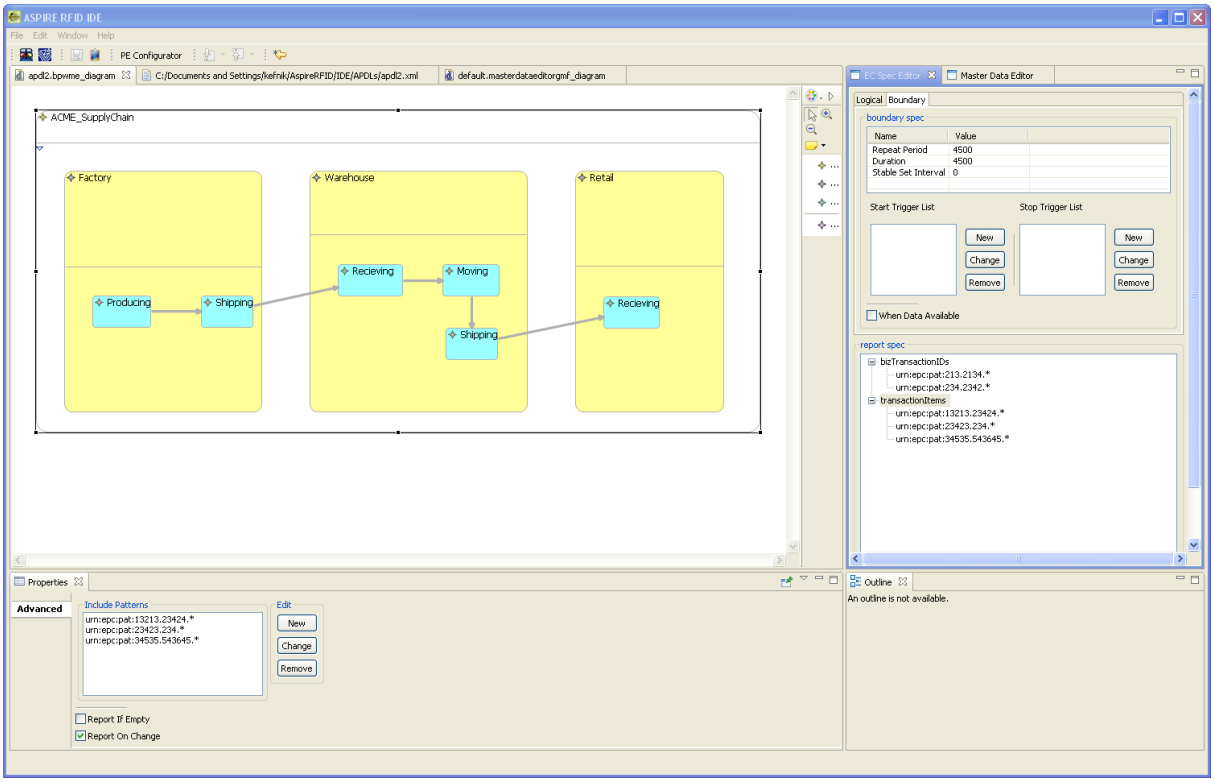


Figure 23 EBProc's ECSpec Configuration

In Figure 24 below, at the upper right (Master Data Editor) Tab, we can see where someone could edit the EBProc's Transaction attributes that are consisted from User and Standard Vocabularies. The CLCBProc's Business Location can be edited with the help of the Business Location Diagram, shown below, where after editing it the User could use the read point provided list to add to the EBProc's BizTransaction configuration he sets up.

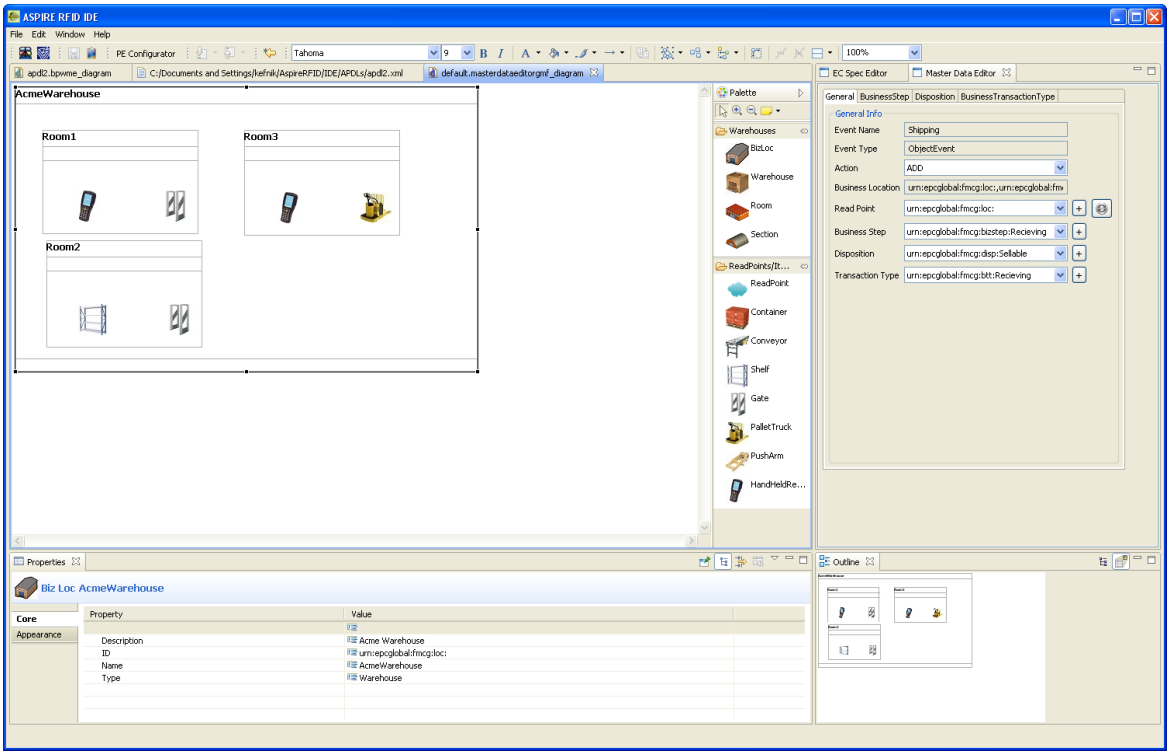


Figure 24 CLCBProc BizLocation & EBProc MasterData Editing

In Figure 25 below, at the upper right tab, we can see the Standard Vocabulary attributes configuration (e.g. Business Step) that could be used from the Hole Company (the same list for all the CLCBProcs).

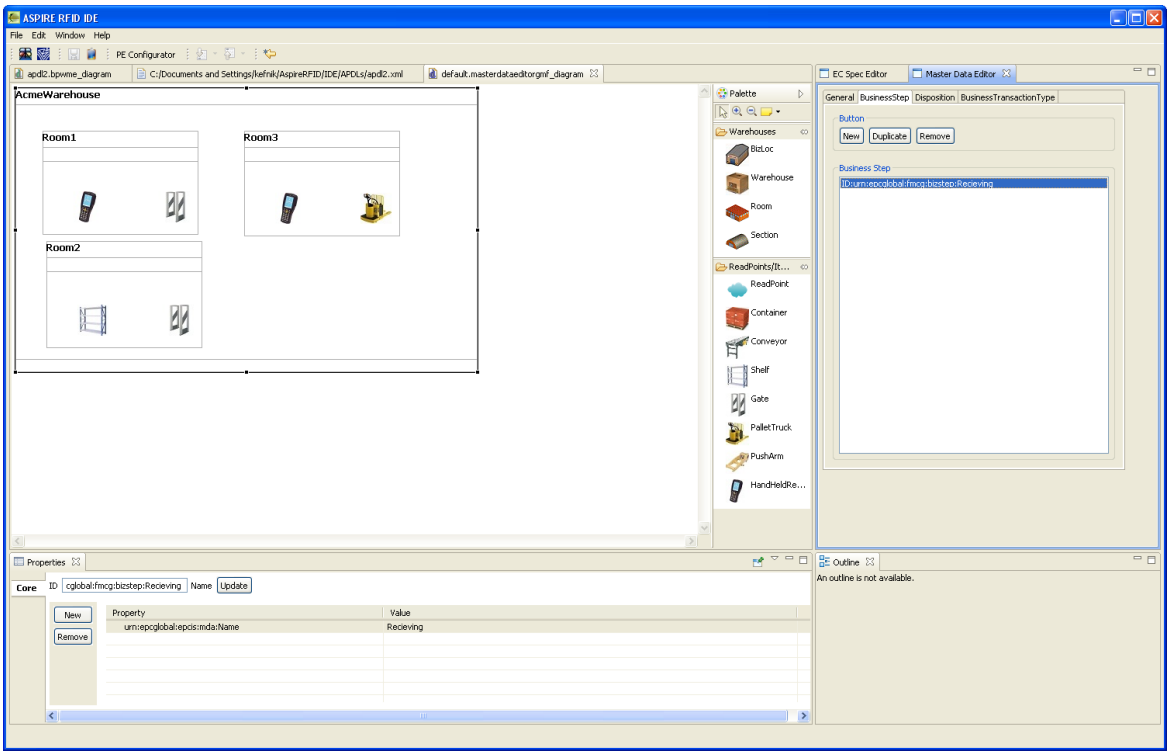


Figure 25 CLCBProc BizLocation & Standard Vocabulary (BizStep) Editing

Finally in Figure 26 below we can see the APDL XML file that is produced and automatically provided every time we choose the specific Tab.

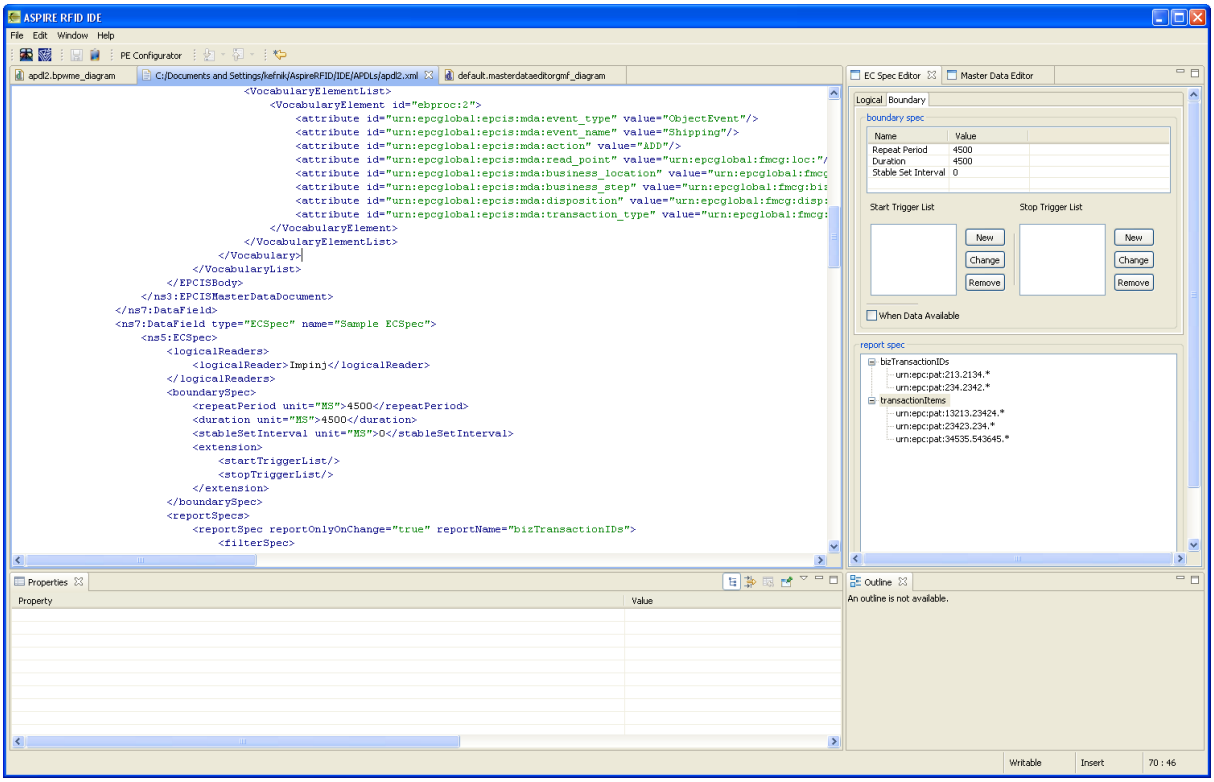


Figure 26 BPWME's created APDL XML file preview

7.2 Programmable Engine

A middleware layer that facilitates the RFID development and augments the BPWME and APDL's functionality is the Programmable Engine (PE) [82] module.

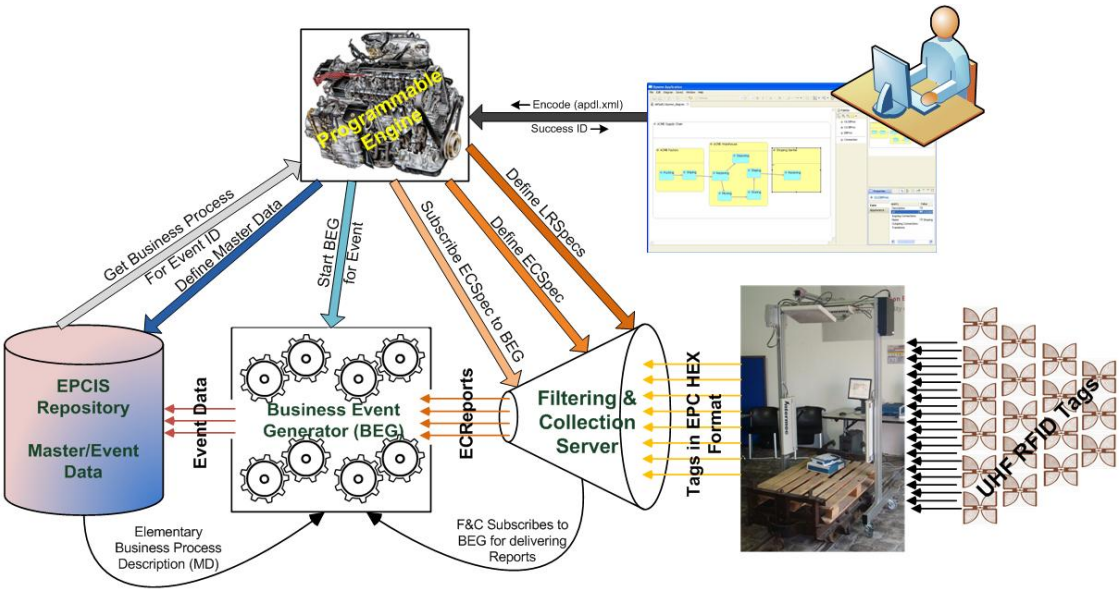


Figure 27 Programmable Engine

The PE, as shown in Figure 27, 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).

Section 8 Describing an RFID Workflow Process using APDL

8.1 Overview

In this Section we will use the Receiving Example provided at deliverable D4.3a (Programmable Filters – FML Specification). At the D4.3a'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 this example we will describe how an APDL (AspireRFID Process Description Language) specification file should be defined so as to be able to configure the whole AspireRFID middleware to serve a warehouse receiving process. So let's start by remembering the problem description.

8.2 Describing the Problem

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

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

8.3 Solution Requirements

An RFID Portal should be placed to ACME's Warehouse1 Section1 entrance point which will be called ReadPoint1. The RFID portal will be equipped with one Reader WarehouseRfidReader1. The received goods should get equipped with preprogrammed RFID tags from their "Manufacturer". The received goods should be accompanied with a preprogrammed RFID enabled delivery document. And finally AspireRFID middleware (Figure 28 below) should be configured for the specific scenario.

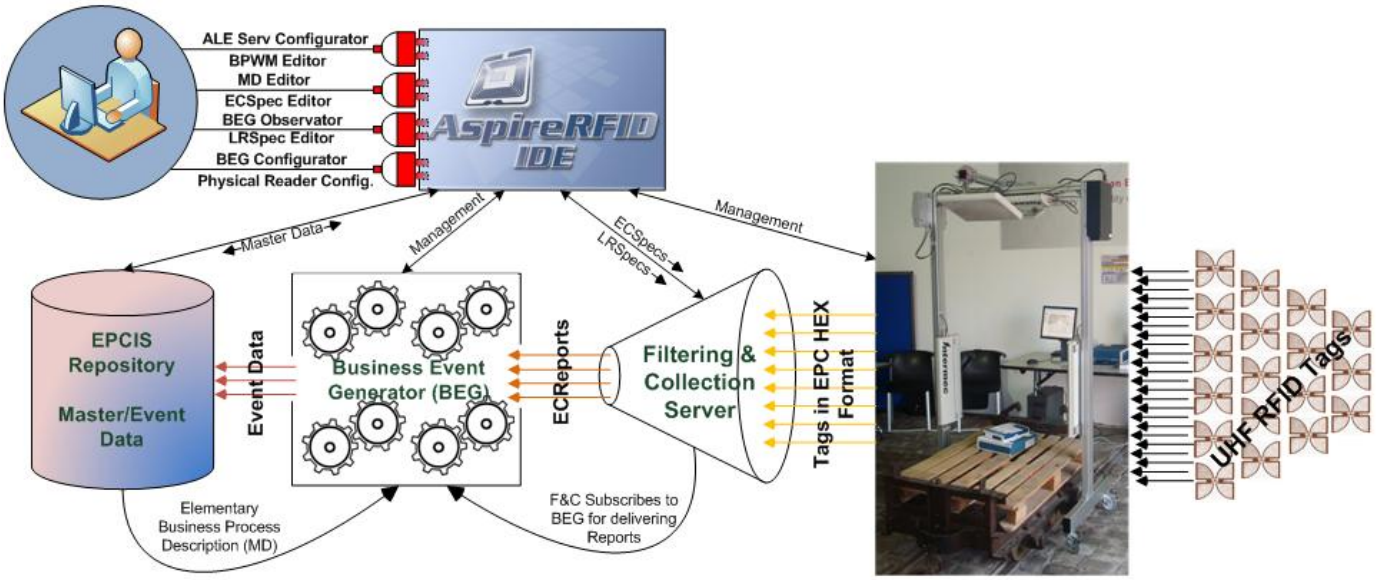


Figure 28 AspireRFID Architecture

8.4 Building the Required APDL Specification File

First, we need to define the OLCBProc 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 22 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 22 CLCBProc Element

In order to describe the EBProc shown in Table 23 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 23 AcmeWarehouse3Ship EBProc

DataFields contains the specification files required to configure the AspireRFID Filtering & Collection server (by defining the ECSpec and LRSpec) and the Business Event Generator (By defining the Transaction Vocabulary at the EPCIS's repository Master Data thru an EPCISMasterDataDocument).

8.4.1 Filtering and collection Module Required Data Fields

8.4.1.1 ECSpec definition

To Configure the Filtering and collection Module an ECSpec is required for creating Object Events for the Class of "products" and the Class of "receiving notes" that we expect to pass through the gate and that concerns our transaction. 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.*

So the ECSpec DataField that should be created is shown in Table 24 below. Note that at the configuration time the ECSpec name that will be used is the ECSpec DataField ID and at the ECRepot names the EBProc's ID will be concatenated to them for example the *bizTransactionIDs* will become *bizTransactionIDs@urn:epcglobal:fmcg:bte:acmewarehouse1receive* and the *transactionItems* will become *transactionItems@urn:epcglobal:fmcg:bte:acmewarehouse1receive* that are required to be delivered to the BEG engine.

```
<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 24 ECSpec DataField

Note that in Appendix I Table 28 the complete APDL XML document describing this example can be found.

8.4.1.2 LRSpec Definition

For the LRSpec DataField definition the dynamic LRSpec definition of an Impinj Speedway LLRP reader is used as shown in Table 25 below where at the configuration time the LRSpec DataField's name (SmartLabImpinjSpeedwayLogicalReader) will be used as the Logicals 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 25 LRSpec DataField

8.4.2 BEG Module Required Data Field

The Business Event Generator needs to get the Transaction Event to serve which is the Warehouse1DocDoorReceive (with URI urn:epcglobal:fmcg:bte:acmewarehouse1receive) and the description of it from the Information Sharing module repository which should be set up using the information from Table 26 below.

Business Transaction Attribute Name	Business Transaction Attribute Value
urn:epcglobal:epcis:mda:event_name	Warehouse1DocDoorReceive
urn:epcglobal:epcis:mda:event_type	ObjectEvent
urn:epcglobal:epcis:mda:business_step	urn:epcglobal:fmcg:bizstep:receiving
urn:epcglobal:epcis:mda:business_location	urn:epcglobal:fmcg:loc:acme:warehouse1
urn:epcglobal:epcis:mda:disposition	urn:epcglobal:fmcg:disp:in_progress
urn:epcglobal:epcis:mda:read_point	urn:epcglobal:fmcg:loc:45632.Warehouse1DocDoor
urn:epcglobal:epcis:mda:transaction_type	urn:epcglobal:fmcg:btt:receiving
urn:epcglobal:epcis:mda:action	ADD

Table 26 Master Data (Specifying a Transaction Event)

So we create an EPCISMasterDataDocument DataField shown in Table 27 below. Note that we are not including the required from the BEG engine ECReport names at the description as we did at the D4.3a's example because this information can get retrieved from the EBProc's ID and the Event Type. Because this is an Object Event we know that two reports are required the *bizTransactionIDs* and the *transactionItems* where the EBProc ID *urn:epcglobal:fmcg:bte:acmewarehouse1receive* will get concatenated.

```
<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 27 EPCISMasterDataDocument DataField

As mentioned before note that in Appendix I Table 28 the complete APDL XML document describing this example can be found.

8.5 Process Description

As described in Section 9.6 of D4.3a (Programmable Filters – FML Specification) 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.

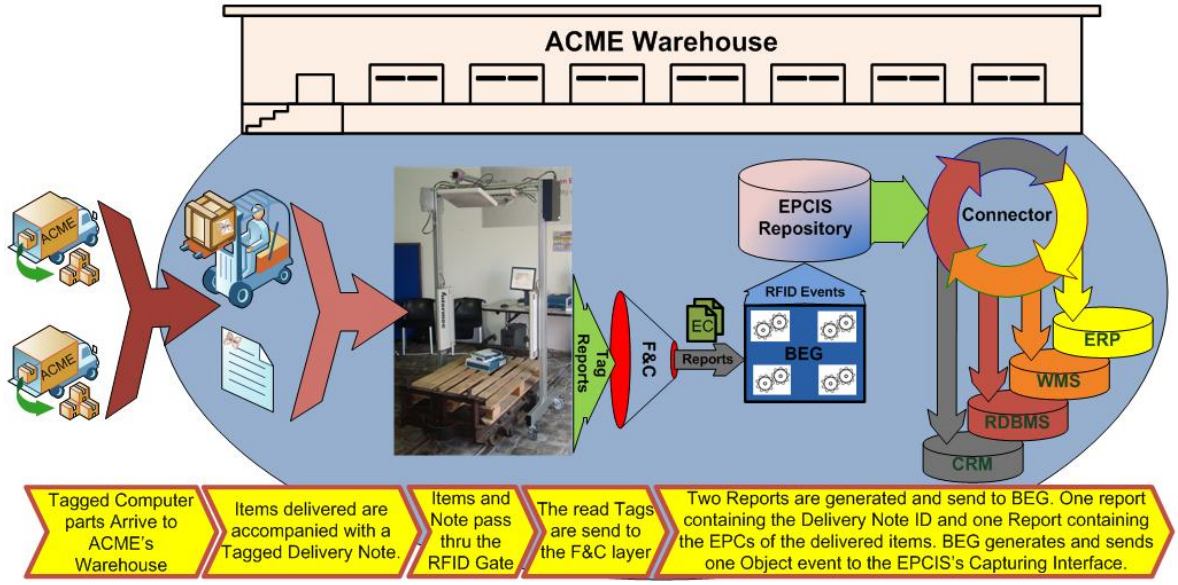


Figure 29 Acme computer parts Delivery

As visualized at Figure 29 above 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.

Section 9 Conclusions

This deliverable defines a process oriented meta-language (i.e. the APDL language) for describing non-trivial end-to-end RFID based solutions. The document was particularly concentrated on the concept of language-oriented programming, and provided a detailed overview of the available models, workflows and languages. It has presented the Business Process Modeling (Definition) and its different tools (Modeling Notation, Diagrams), as well as activity diagrams (UML) and a list of Business Process Modeling Languages (BPML, BPEL, XPDL, YAWL). The conducted review revealed the absence of a configuration language tailored to RFID solutions.

Further, this deliverable focused on some specific types of Open Source Software Business Process Modeling Languages based on the XML Process Definition Language (XPDL). In this way, a description of Enhydra JaWE, Nova Bonita, Eclipse Java Workflow Tooling and YAPROC has been provided. These 4 XPDL editors have then been analyzed and compared to determine which suited the best to the ASPIRE paradigm. The analysis revealed that none of the above mentioned editors were suitable enough for the needs of ASPIRE, or the effort needed to adapt them to ASPIRE would be too big. All the above led us to design our own Business Process Modeling Language the AspireRFID Process Description Language (APDL) and later on a design tool, which was further described in details (in Section 7.1).

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 this deliverable we have illustrated some realistic RFID systems and their description based on APDL (Section 8).

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.

Section 10 List of Acronyms

ALE	Application Level Event
APDL	AspireRFID Process Description Language
API	Application Product Interface
ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generator
BPD	Business Process Diagram
BPDM	Business Process Definition Metamodel
BPEL	Business Process Execution Language
BPM	Business Process Management
BPM	Business Process Modeling
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
CLCBProc	Close Loop Composite Business Process
DoW	Description of Work
EBProc	Elementary Business Process
EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Services
ERP	Enterprise Resource Planning
F&C	Filtering and Collection
FML	Filter Markup Language
HAL	Hardware Abstraction Layer
HF	High Frequency
HTTP	HiperText Transfer Protocol
IDE	Integrated Development Environment
iPOJO	injected POJO
IT	Information Technology
JMX	Java Management Extensions
LLRP	Low Level Reader Protocol
OBR	OSGi Bundle Repository
OLCBProc	Open Loop Composite Business Process
OSGI	Open Service Gateway Initiative
OSS	Open Source Software
POJO	Plain Old Java Object
RFID	Radio Frequency Identification
RP	Reader Protocol
SME	Small and Medium Enterprise
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCO	Total Cost of Ownership
TCP	Transfer Control Protocol
UHF	Ultra High Frequency
UML	Universal Markup Language
WADL	Wired Application Description Language
WMS	Warehouse Management System

WP	Work Package
WSDL	Web Services Description Language
XML	Extensible Markup Language
XPDL	XML Process Definition Language
YAWL	Yet Another Workflow Language

Section 11 List of Figures

Section 12 List of Tables

Section 13 References and bibliography

[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] S. Sarma, "Integrating RFID," ACM Queue, vol. 2, no. 7, pp. 50–57, 2004.
[7] Nikos Kefalakis, Nektarios Leontiadis, John Soldatos, Didier Donsez, "Middleware Building Blocks for Architecting RFID Systems", MOBILIGHT 2009, pp. 325-336.
[8] EPCglobal, "The Application Level Events (ALE) Specification, Version 1.1", February. 2008, available at: http://www.epcglobalinc.org/standards/ale
[9] EPC Information Services (EPCIS) Specification, Version 1.0.1, September 21, 2007 available at: http://www.epcglobalinc.org/standards/epcis/
[10] EPCglobal, The EPCglobal Architecture Framework Version 1.3, available online at http://www.epcglobalinc.org/standards/architecture/
[11] EPCglobal standards, http://www.epcglobalinc.org/standards
[12] Russell Scherwin and Jake Freivald, Reusable Adapters: The Foundation of Service-Oriented Architecture, 2005.
[13] BEAWebLogic RFID Enterprise Server™, "Understanding the Event, Master Data, and Data Exchange Services", Version 2.0, Revised: October 12, 2006.
[14] 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".
[15] Architecture Review Committee, "The EPCglobal Architecture Framework," EPCglobal, July 2005, available at: http://www.epcglobalinc.org.
[16] 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).
[17] Benita M. Beamon, "Supply chain design and analysis: Models and methods", International Journal of Production Economics, Vol. 55 pp. 281-294, 1998
[18] Zhekun Li, Rajit Gadh, and B. S. Prabhu, "Applications of RFID Technology and Smart Parts in Manufacturing", Proceedings of DETC04: ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference September 28-October 2, 2004, Salt Lake City, Utah USA.
[19] "Business Process Modelling FAQ". http://www.BPModeling.com/faq/. Retrieved on 2008-11-02.

[20] "BPMN FAQ". http://www.BPMNforum.com/FAQ.htm. Retrieved on 2008-11-02.
[21] Thomas Dufresne & James Martin (2003). "Process Modeling for E-Business". INFS 770 Methods for Information Systems Engineering: Knowledge Management and E-Business. Spring 2003
[22] Williams, S. (1967) "Business Process Modeling Improves Administrative Control," In: Automation. December, 1967, pp. 44 - 50.
[23] BPMMaturity.com
[24] ITWire. “IDS Scheer Launches in Australia.” July 5, 2007.
[25] Asbjørn Rolstadås (1995). "Business process modeling and reengineering". in: Performance Management: A Business Process Benchmarking Approach. p. 148-150.
[26] Brian C. Warboys (1994). Software Process Technology: Third European Workshop EWSPT'94, Villard de Lans, France, February 7-9, 1994 : Proceedings. p. 252.
[27] The Business Model Ontology - A Proposition In A Design Science Approach, Thesis by Alexander Osterwalder, 2004
[28] See e.g., ISO 12052:2006, [1]
[29] Workflow/Business Process Management (BPM) Service Pattern June 27, 2007. Accessed 29 nov 2008.
[30] FEA (2005) FEA Records Management Profile, Version 1.0. December 15, 2005.
[31] FEA Consolidated Reference Model Document. May 2005.
[32] Paul R. Smith & Richard Sarfaty (1993). Creating a strategic plan for configuration management using Computer Aided Software Engineering (CASE) tools. Paper For 1993 National DOE/Contractors and Facilities CAD/CAE User's Group.
[33] Business Process Reengineering Assessment Guide, United States General Accounting Office, May 1997.
[34] Wil M.P. van der Aalst, "Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management", Springer Lecture Notes in Computer Science, Vol 3098/2004.
[35] Wil M.P. van der Aalst, "Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language", Eindhoven University of Technology, PDF.
[36] Jiang Ping, Q. Mair, J. Newman, "Using UML to design distributed collaborative workflows: from UML to XPDL", Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings, ISBN 0-7695-1963-6.
[37] W.M.P. van der Aalst, "Don't go with the flow: Web services composition standards exposed", IEEE Intelligent Systems, Jan/Feb 2003.
[38] Jürgen Jung, "Mapping Business Process Models to Workflow Schemata An Example Using Memo-ORGML And XPDL", Universität Koblenz-Landau, April 2004, PDF.
[39] Volker Gruhn, Ralf Laue, "Using Timed Model Checking for Verifying Workflows", José Cordeiro and Joaquim Filipe (Eds.): Proceedings of the 2nd Workshop on Computer Supported Activity Coordination, Miami, USA, 23.05.2005 - 24.05.2005, 75-88. INSTICC Press ISBN 972-8865-26-0.
[40] Nicolas Guelfi, Amel Mammar, "A formal framework to generate XPDL specifications from UML activity diagrams", Proceedings of the 2006 ACM symposium on Applied computing, 2006.
[41] Peter Hrastnik, "Execution of business processes based on web services", International Journal of Electronic Business, Volume 2, Number 5 / 2004.

[42] Petr Matousek, "An ASM Specication of the XPDL Language Semantics", Symposium on the Effectiveness of Logic in Computer Science, March 2002, PS.
[43] F. Puente, A. Rivero, J.D. Sandoval, P. Hernández, and C.J. Molina, "Improved Workflow Management System based on XPDL", Editor(s): M. Boumedine, S. Ranka, Proceedings of the The IASTED Conference on Knowledge Sharing and Collaborative Engineering, St. Thomas, US Virgin Islands, November 29-December 1, 2006, ISBN 0-88986-433-0.
[44] Petr Matousek, "Verification method proposal for business processes and workflows specified using the XPDL standard language", PhD thesis, Jan 2003.
[45] Albert Rainer (2004). "Web-centric business process modelling". International Journal of Electronic Business 2 (5).
[46] Y Xiao, D Chen, M Chen (2004). "Research of Web Services Workflow and its Key Technology Based on XPDL". Proc. 2004 IEEE International Conference on Systems, Man and Cybernetics 3: Pages 2137–2142. doi:10.1109/ICSMC.2004.1400643. ISBN 0-7803-8566-7.
[47] Stefan Jablonski (2005). "Processes, Workflows, Web Service Flows: A Reconstruction". Data management in a connected world: essays dedicated to Hartmut Wedekind on the occasion of his 70th Birthday (Lecture Notes in Computer Science). Berlin: Springer. doi:10.1007/11499923_11. ISBN 3540262954.
[48] Thomas Hornung, Agnes Koschmider, Jan Mendling, "Integration of Heterogeneous BPM Schemas: The Case of XPDL and BPEL", Technical Report JM-2005-03, Vienna University of Economics and Business Administration, 2006 PDF.
[49] Wei Ge, Baoyan Song, Derong Shen, Ge Yu, "e_SWDL: An XML Based Workflow Definition Language for Complicated Applications in Web Environments" Web Technologies and Applications: 5th Asia-Pacific Web Conference, APWeb 2003, Xian, China, April 23-25, 2003. Proceedings, ISSN 0302-9743.
[50] Workflow Management Coalition Workflow Standard, “Workflow Process Definition Interface -- XML Process Definition Language V1.0”, Document Number WFMC-TC-1025, October 25, 2002
[51] Business Processes Modeling Forum: http://www.bpmodeling.com
[52] Minoli Daniel, “Enterprise Architecture A to Z: Frameworks, Business Process Modeling, SOA, and Infrastructure Technology”, CRC Press, 2008.
[53] Business Process Definition Metamodel: http://en.wikipedia.org/wiki/BPDM
[54] Scott W. Ambler, “Agile Model Driven Development with UML 2”, the object primer 3rd Edition, Cambridge University Press, 2004.
[55] Scott W. Ambler, “The Elements of UML 2.0 Style”, Cambridge University Press, 2005.
[56] Business Process Execution Language: http://en.wikipedia.org/wiki/BPEL
[57] XPDL Support and Resources: http://www.wfmc.org/xpdl.html
[58] Wil M.P. Van der Aalst, “Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language”, BPM Center Report, 2003.
[59] Yet Another Workflow Language: http://en.wikipedia.org/wiki/YAWL
[60] Wil M.P. Van der Aalst, M. Adams, A.H.M. ter Hofstede, M. Pesic, and H. Schonenberg, “Flexibility as a Service”, BPM Center Report, 2008.
[61] Aalst, W. M., Mulyar, S., Russell, N., & Arthur, H. (2006). WORKFLOW CONTROL-FLOW PATTERNS - A Revised View.
[62] Aalst, W. v., Hofstede, A. t., Kiepuszewski, B., & Barros, A. (2003). Workflow patterns. Distributed and Parallel Databases , 5-51.
[63] Dauten, D. (1996). The Max Strategy: How a businessman got stuck at an airport and learned to make his career take off. New York: William Morrow and Company.

[64] Davenport, T. (1993). Process Innovation: Reengineering work through information technology. Boston: Harvard Business School Press.
[65] Jeston, J., & Nelis, J. (2008). Business Process Management: Practical Guidelines to Successful Implementation. Elsevier Ltd.
[66] Smith, H., & Fingar, P. (2003). Business Process Management the third wave. Tampa: Meghan-Kniffer Press.
[67] Voss, C., & Huxham, C. (2004). Problems, Dilemmas and Promising Practices. Proceedings of the 11th Annual, (pp. 309-318).
[68] Weske, M. (2007). Business Process Management: Concepts, Languages, Architectures. Springer.
[69] White, S. A. (2004, March). Process Modeling Notations. BPTrends .
[70] Zullighoven, H., & Riehle, D. (1996). Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems , 3-13.
[71] Lombardi. (2008). Retrieved July 23, 2009, from Lombardi_getting started with BPM: www.lombardi.com
[72] Estublier, J., Dami, S., Amiour, M.: APEL: A graphical yet executable formalism for process modeling. Automated Software Engineering: An International Journal 5(1) (1998) 61–96
[73] Pedraza, G., Dieng, I., Estublier: J. Multi-concerns Composition for a Process Support Framework. 1st European Workshop in Model-Driven Tool & Process Integration, Berlin, Germany, 2008
[74] Ward, M.: Language-Oriented Programming. Software - Concepts and Tools, pp. 147-161, 1994
[75] Mernik, M., Heering, J., and Sloane, A. M. 2005. When and how to develop domain-specific languages. ACM Comput. Surv. 37, 4 (Dec. 2005), 316-344
[76] B. Griswold, E. Hilsdale, J. Hugunin, W. Isberg, G. Kiczales, M. Kersten. "Aspect-Oriented Programming with AspectJ". Available on http://aspectJ.org[LH95] C. V. Lopes, W. L. Hursch, “Separation of Concerns”, College of Computer Science, Northeastern University, Boston, February 1995
[77] [Kic92] G. Kiczales. "Towards a New Model of Abstraction in the Engineering of Software". In Proceedings of IMSA'92. Workshop on Refection and Meta-Level Architecture, 1992.
[78] [OI94] H. Okamura, Y. Ishikawa. "Object Location Control Using Meta-level Programming". In Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP'94), pages 299-319, Lecture Notes in Computer Science Vol. 821, Springer-Verlag, Bologa, Italy, July 1994.
[79] [ABS+94] M. Aksit, J. Bosch, W. van der Sterren, L. Bergmans. "Real-Time Specification Inheritance Anomalies and Real-Time Filters". In Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP'94), Lecture Notes in Computer Science, Vol. 821, pages 386-407, Springer-Verlag, Bologa, Italy, July 1994.
[80] John Soldatos, Nikos Kefalakis, Nektarios Leontiadis, et. al., “Programmable Filters – FML Specification”, ASPIRE Project Public Deliverable D4.3b, Semptember 2009, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables
[81] The AspireRfid project, http://forge.objectweb.org/projects/aspire/ (forge), http://wiki.aspire.objectweb.org/xwiki/bin/view/Main/WebHome (wiki)
[82] Nikos Kefalakis, John Soldatos, Yongming Luo, et. al., “ASPIRE Programmable Engine (APE) (Interim Version)”, ASPIRE Project Public Deliverable D4.2a, December 2009, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables

[83] Nikos Kefalakis, John Soldatos, Nikolaos Konstantinou and Neeli R. Prasad "APDL: An XML-based Language for Integrated RFID Solutions", Submitted to the Journal of Systems and Software (Elsevier), May 2010

APPENDIXES

APPENDIX I. ACME's Complete APDL Solution XML

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

  <!--
    Copyright © 2008-2010, Aspire Aspire is free software; you can redistribute
    it and/or modify it under the terms of the GNU Lesser General Public License
    version 2.1 as published by the Free Software Foundation (the "LGPL"). You
    should have received a copy of the GNU Lesser General Public License along
    with this library in the file COPYING-LGPL-2.1; if not, write to the Free
    Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
    02110-1301 USA. This software is distributed on an "AS IS" basis, WITHOUT
    WARRANTY OF ANY KIND, either express or implied. See the GNU Lesser General
    Public License for the specific language governing rights and limitations.
    Author: Nikos Kefalakis (nkef@ait.edu.gr)
  -->

<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>
```

Table 28 ACME's Complete APDL Solution XML

APPENDIX II. APDL 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">

   <!--
      Copyright © 2008-2010, Aspire Aspire is free software; you can
      redistribute it and/or modify it under the terms of the GNU Lesser General
      Public License version 2.1 as published by the Free Software Foundation
      (the "LGPL"). You should have received a copy of the GNU Lesser General
      Public License along with this library in the file COPYING-LGPL-2.1; if
      not, write to the Free Software Foundation, Inc., 51 Franklin Street,
      Fifth Floor, Boston, MA 02110-1301 USA. This software is distributed on an
      "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied.
      See the GNU Lesser General Public License for the specific language
      governing rights and limitations.
   -->
   <!--
            Author: Nikos Kefalakis (nkef@ait.edu.gr)
    -->


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

   <xs:complexType name="OLCBProc">
      <xs:sequence>
         <!--
            At this level the "epcismd:EPCISMasterDataDocument" is used to store
            only Standard Vocabulary types and more specifically the
            "urn:epcglobal:epcis:vtype:BusinessStep", the
            "urn:epcglobal:epcis:vtype:Disposition" and the
            "urn:epcglobal:epcis:vtype:BusinessTransactionType" types.
         -->
         <xs:element minOccurs="0" maxOccurs="1"
            ref="epcismd:EPCISMasterDataDocument" />
         <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" />
         <!--
            At this level the "epcismd:EPCISMasterDataDocument" is used to store
            only User Vocabulary types and more specifically the
            "urn:epcglobal:epcis:vtype:BusinessLocation" and the
            "urn:epcglobal:epcis:vtype:ReadPoint" types.
         -->
         <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 29 APDL schema

APPENDIX III. Control-Flow Perspective of Workflow Systems Patterns

I. Basic Control Flow Patterns

This class of patterns captures elementary aspects of process control. Basic control flow patterns include Sequence, Parallel Split, Synchronization, Exclusive Choice, and Simple Merge.

1. Sequence pattern

Description: An activity in a workflow process is enabled after the completion of a preceding activity in the same process.

Example: An activity print-receipt is executed after the execution of activity issue-ticket.

2. Parallel Split (AND-Split)

Description: A point in the process model where a single thread of control splits into multiple threads of control which are executed concurrently.

Example: When a temperature-alert-high is received, trigger the *reduce_to_desire_temprature* activity and the *inform_admin* activity immediately.

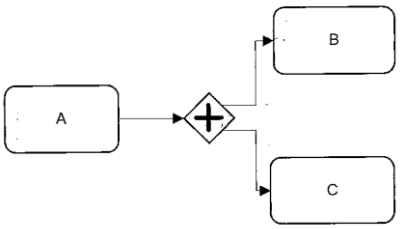


Figure 30 And Split Pattern [68]

3. Synchronization (AND-Join)

Description: is a point in the process model where multiple concurrent threads(executed in any order or in parallel) are converge into one single thread; do not proceed with the execution of the following activities until all these preceding activities have completed.

Example: The ship-goods activity runs immediately after both the pack-goods and produce_dispatch_list activities are completed.

4. Exclusive Choice (XOR-SPLIT)

Description: an XOR split or exclusive or split is a point in a process model where precisely one of the several branches available is chosen based on the outcome of a logical expression associated with the branch.

Example: After the review election activity is complete, either the declare results or the recount votes activity is undertaken.

5. *Simple Merge (XOR-Join)*

Description: A point in the work flow process where two or more alternative branches come together without synchronization. In other words the merge will be triggered once any of the incoming transitions are executed.
Example: Activity archive_claim is enabled after either pay_damage or contact_customer is executed.

II. Advanced Branching and Synchronization Patterns

This section outlines a series of patterns that have more complex branching and merging concepts which arise in business processes.

6. *Multi Choice (Or split)*

Description: a point in a process model where, based on the outcome of distinct logical expressions associated with each of the branches, one or more branches are chosen.

Example: Depending on the nature of the emergency call, one or more of the despatch-police, despatch-fire-engine and despatch-ambulance activities is initiated.

7. *Synchronizing merge (Or Join)*

Description: A point in a process model where multiple paths converge into one single thread. Synchronize needs to take place if more than one path is taken. Whereas if only one path is taken the alternative branches should reconverge without synchronization.

Example: Continuing example above. Once all the emergency vehicles arrive at the accident, the transfer- patient activity starts.

8. *Multiple Merge*

Description: A multi merge is a point in a process model where two or more concurrent threads join without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started only once for every incoming branch that gets activated.

Example: A simple example of this would be two activities audit application and process application running in parallel which should both be followed by an activity close case.

9. *Discriminator*

Description: a point in a process model that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and "ignores" them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again. This allows a discriminator to be used in the context of a loop.

The importance of gathering the ignored branch as part of the functional behaviour of the discriminator pattern is that without it there would be no way to distinguish a second iteration of a loop from a late branch of its first iteration.

Example: A paper needs to be sent to external reviewers. The paper is accepted if both reviews are positive. But if the first review that arrives is negative, the author(s) should be notified without having to wait for the second review.

10.N-out-of-M-Join

Description: Is a point in a process model where M parallel paths converge into one. The subsequent activity is activated once N paths have completed; completion of all remaining parts should be ignored. Similar to the discriminator, once all incoming branches have triggered, the join resets itself so that it can be performed again.

Example: A request of quotation process, in which quotations are invited from five companies. Upon receiving three quotations the quotation process can be processed. The last two quotations can be ignored.

III. Structural Patterns

In this section two patterns are presented which illustrate typical restrictions imposed on work flow specifications.

11.Arbitrary Cycles

Description: The ability to represent cycles in a process model that have more than one entry or exit point. I.e. does not impose any structural restrictions on the types of loops that can exist in the process model.

Example: Figure below provides an illustration of the pattern with two entry points: p3 and p4.

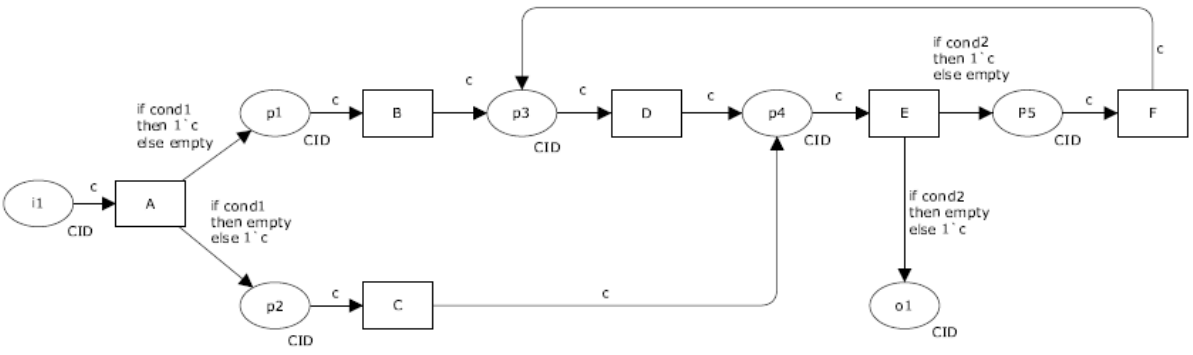


Figure 31 Example of Arbitrary Cycles, Source: (Aalst, Mulyar, Russell, & Arthur, 2006)

12.Implicit Terminations

Description: A given process instance should terminate when there are no remaining work items that are able to be done either now or at any time in the

future. Unlike other control flow patterns, role of implicit termination pattern is different. It does not relate activity instances with one another instead it represents a termination condition of an overall process. Usually termination is explicit in process languages because there is exactly one state in the process that marks it termination. If there are many states then termination is implicit [58].

IV. Patterns with multiple instances

This section outlines patterns with multiple instances. These multiple instance patterns describe situations where one activity in a process model can have more than one running, active instance at the same time.

13.Multiple Instances without Synchronization
Description: Generates multiple instances of an activity without the need to synchronise these activity instances afterwards.

Example: An order list is received which contains a number of order lines. For each order line a check activity needs to be executed. These activities are run to completion concurrently and do not trigger any subsequent activity. And do not require synchronization at completion.

14.Multiple Instances with Priori Design Time Knowledge
Description: Within a given process instance, multiple instances of an activity are generated with the number of activity instances of the activity model known at design time. While these instances are independent and running concurrently, they have to be synchronized at completion for subsequent activity to be triggered.

Example: An annual report has to be signed by all 6 directors before being published.

15.Multiple Instances with Priori Run Time Knowledge
Description: Within a given process instance, multiple instances of an activity are generated. The required number of instances varies and depends on characteristics of the case, resource availability, and inter-process communications, but is known before activity instance has to be created. While these instances are independent and running concurrently, they have to be synchronized at completion for subsequent activity to be triggered.

Example: While processing an order for multiple books, the activity check availability is executed for each individual book.

16.Multiple Instances without Priori Run Time Knowledge
Description: Within a given process instance, multiple instances of an activity are generated. The required number of instances varies and depends on a number of run time factors; resource availability and inter-process communications, but is known until the final activity instance has completed. At any time, whilst instances are running, it is possible for additional instances to be initiated. While

these instances are independent and running concurrently, they have to be synchronized at completion for subsequent activity to be triggered.

Example: The requisition of 30 computers involves an unknown number of deliveries since the number of computers per delivery is unknown. Once each delivery is processed, it can be determined whether next delivery is to come by taking the difference between goods requested and goods delivered.

V. State based patterns

This section illustrates patterns that capture the implicit behaviour of processes based not on the current case but on the environment or other parts of the processes. In this context, the state of a process instance includes the broad collection of data associated with current execution including the status of various activities as well as process-relevant working data such as activity and case data elements

17.Deferred Choice

Description: A point in the process model where one of several branches is chosen. Unlike the XOR-split, where the choice is made explicitly (e.g. based on data or a decision) instead several alternatives are offered to the environment. However, once the environment activates one of the branches the other alternative branches are withdrawn. It is important to note that the choice is delayed until the processing in one of the alternative branches is actually started, therefore the moment of choice is deferred to a point in time that is as late as possible.

Example: Upon receiving the products there are two ways to transport the products to the department. The selection is based on the availability of the corresponding resources. Therefore, the choice is deferred until a resource is available.

18.Interleaved Parallel Routing

Description: Execute a number of activities in any order (e.g. based on availability of resources), the order is decided at run time, and does not execute any of these activities at the same time/simultaneously.

Example: A bank performs two activities on each account annually; add interest and charge credit card costs. These activities can be conducted in any order but not at the same time since both update the account there can be executed at the same time.

19.Milestone

Description: an activity is only enabled when the process instance is in a specific state. For instance, enable a certain activity at any time before the milestone is reached, after which the activity can no longer be executed.

Example: a budget travel agent allows routing of bookings to be changed as long as the ticket has not been issued.

VI. Cancellation patterns

In this section two patterns are presented that deal with cancellation of activities and cases.

20. Cancel activity
Description: An enabled activity is withdrawn if the execution has not started. If the execution has started, it is disabled and, where possible, the currently running instance is halted and removed.

Example: The assess damage activity is undertaken by two insurance assessors. Once the first assessor has completed the activity, the second is cancelled.

21. Cancel Case
Description: Removing a complete process instance. Even if parts of the process are instantiated multiple times, all descendants are removers while process instance is recorded as completed unsuccessfully.

Example: A customer withdraws a mortgage application before the final decision is made because he/she decides not to buy the house anymore.