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 Filters – FML Specification

Due date of deliverable: M24
Actual Submission date: 18/12/09

Deliverable ID:	**WP4/D4.3b**
Deliverable Title:	Programmable Filters – FML Specification
Responsible partner:	IT-Ramiro Robles (editor)
Contributors:	John Soldatos (AIT) (main contributor) Nikos Kefalakis (AIT) (main contributor) Nektarios Leontiadis (AIT) (main contributor) Loïc Schmidt (INRIA), Nathalie Mitton (INRIA)
Estimated Indicative Person Months:	24

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

Revision: 0.4b (final)
Dissemination Level: PU

Document Information

Document Name: **Programmable Filters – FML Specification (Interim Version)**
Document ID: **WP4/D4.3b**
Revision: **0.4b (final)**
Revision Date: **16 December 2009**
Author: **IT(editor), AIT (main contributor), INRIA (contributor)**
Security: **PU**

Approvals

	Name	Organization	Date	Visa
Coordinator	Neeli Rashmi Prasad	CTIF-AAU	18.12.2009	Approved
Technical Coordinator	John Soldatos	AIT		
Quality Manager	Anne Bisgaard Pors	CTIF-AAU	18.12.2009	Approved

Reviewers

Name	Organization	Date	Comments	Visa
Nathalie Mitton	INRIA			
Neeli R. Prasad	AAU	17.12.2009		

PMs Spend per Partner

Organization	PMs Spend	Comments
AAU	0.25	Review and Quality Control
INRIA	1.5	Section 10
AIT	7	AIT is the main contributor of the work described within Sections 3, 4, 5, 6, 7, 8 and 9 of the deliverable. AIT has also implemented and contributed this work to the AspireRfid project (http://wiki.aspire.ow2.org/)
IT	0.5	Editor and reviewer

Document history

Revision	Date	Modification	Authors
a_0.1	20 Mar 09	First draft	Nikos Kefalakis
a_0.2	23 Mar 09	Edited Programmable Filters Specification (creating business logic)	Nikos Kefalakis
a_0.3	24 Mar 09	Augmented Executive Summary and Introduction	Ramiro Samano Robles
a_0.4	26 Mar 09	Added Available Tools and the Complete Example	Nikos Kefalakis
a_0.5	01 Apr 09	Conclusion, augmented Introduction	Nektarios Leontiadis, John Soldatos
a_0.6	14 Apr 09	Section 3,Section 4, augmented Section 2	Nikos Kefalakis, John Soldatos
a_0.7	15 Apr 09	Added Section 10	Loïc Schmidt, Nathalie Mitton
a_0.8	29 Apr 09	Minor Corrections	Nikos Kefalakis
b_0.1			Ramiro Samano Robles
b_0.2	20 Nov 09	Augmented/Corrected Section 4, Par. 6.4, Section 7, Par. 8.3 and Section 9	Nikos Kefalakis
b_0.3	23 Nov 09	Changed Section 1	John Soldatos
b_0.4	12 Dec 09	Editorial modifications.	Ramiro Samano Robles
b_0.5	17 Dec 09	Review	Neeli R. Prasad

Content

Section 1 Executive Summary

Conventional applications of RFID such as toll payment and access control can be handled by a relatively simple and unchallenging middleware platform. However, recent advances in microelectronics and radiofrequency (RF) transceivers have paved the way for advanced RFID business applications at the item level (such as inventory control and supply chain management). This, in turn, has created the need for more complex middleware tools to deal with the huge amount of generated tag data and with all the particularities of these new RFID applications.

To fill this gap, the ASPIRE project aims at developing configurable middleware, tools and techniques for filtering RFID tag streams. This deliverable is devoted to the specification and implementation of tools and techniques towards programmable/configurable filtering of RFID tag streams in the scope of advanced RFID solutions. Filtering of unwanted raw RFID tag data is a crucial functionality of an RFID middleware platform. However, in ASPIRE the filtering functionality has been further extended to the context of higher level abstraction layers. Hence, in this deliverable filtering is mainly concerned with filters that can automatically create business events and subsequently route RFID information to the required back-end enterprise applications (such as Enterprise Resource Planning –ERP- and Warehouse Management Systems –WMS-). These filters operate typically based on high-level business semantics rather than low-level filtering rules which process bits and bytes over the RFID tag streams. For such lower level filtering, ASPIRE relies on the management and configuration of related EPCGlobal (Electronic Product Code Global) compliant middleware modules and related specifications (such as EPC Application Level Events (EPC-ALE) and EC Specifications (ECSpecs)).

In order to make the distinction between low-level and business-level filtering, this deliverable positions its developments in the scope of the overall ASPIRE architecture (described in detail in deliverable D2.3b and briefly described in the introductory section). Furthermore, the present document illustrates the data structures and constructs that are used towards defining and capturing business context, given that the business-level filters are defined based on these constructs. In particular, it is illustrated how business events are used to capture context concerning business locations, readpoints, items, as well as business processes and steps, according to the EPCglobal standards. Accordingly, the deliverable introduces filters for mapping RFID data streams into business events. These filters are specified in an XML-based language (conveniently called FML (Filter Markup Language) in the ASPIRE context) and convey business logic. A main characteristic of these filters is that they can be reused across different RFID systems and applications. This considerably reduces the time required to create new applications from scratch, thereby speeding up the development process. In addition to reusability, FML provides modularity, since it enables RFID integrators to express complex business logics as a collection of distinct reusable filters. Overall, the presented filters boost the programmable nature of the ASPIRE project, which asks for versatility and reusability in terms of RFID

filtering. Indeed, the proposed filtering markup language is versatile and easy to understand for non-specialized developers and RFID integrators.

The business-level filtering mechanisms of this deliverable are part of the overall ASPIRE Integrated Development Environment (IDE), since they are incorporated in the wider ASPIRE tooling architecture. In practice, this means that RFID integrators can define, configure, manage and deploy such filters within the wider ASPIRE IDE, which serves as a single entry point to the definition, design and deployment of integrated RFID solutions. Note that the deliverable provides filtering examples using the FML language. The examples illustrate how to program filters with business meaning in the context of the ASPIRE RFID middleware platform. In particular it is shown how the envisioned filters can cope with business events while being easy to understand by non-specialized RFID solution integrators.

It should be noted that the filtering mechanisms are part of the open source software (OSS) project "AspireRFID" project, which has been successfully established in the scope of the OW2 community (please see http://wiki.aspire.ow2.org/). Hence, the filtering functionality described in this deliverable is an integral part of the AspireRfid project, as a set of middleware libraries that interconnect the generation of Application Level Events with the Information Sharing Repository. These libraries comprise the Business Event Generator (BEG) module of the ASPIRE architecture, as described in Deliverable D2.3b of the project [21].

Section 2 Introduction

2.1 RFID middleware: motives and functionalities

Recent advances in microelectronics and radiofrequency transceivers have paved the way to more efficient and smaller RFID transponders ([19], [20]). As the direct result of this, the cost of passive tags based on backscattering has been dramatically reduced [20], and hence new applications at the item level such as asset tracking and inventory management became possible [19].

Unlike conventional RFID applications such as toll payment and access control, these new applications at the item level require of more specialized and complex middleware tools [9]. Additionally, RFID standards have just recently been updated to cope with these new item-level applications. As a result, many of the current RFID middleware solutions still have to be designed on a per-case basis, which considerably increases the total cost of ownership (TCO).

Due to these reasons, RFID middleware is gradually becoming a cornerstone for non-trivial RFID deployments. This is particularly true in the scope of complex heterogeneous environments comprising multiple readers, application instances, legacy IT (Information technology) systems, as well as sophisticated business processes and semantics. In these environments (e.g., in factories, warehouses, and distribution centers) many distributed readers and antennas capture RFID data, which must be conveyed to a variety of applications (such enterprise resource planning (ERP) systems, warehouse management systems (WMS), corporate databases, process management systems). In such settings, middleware platforms are indispensable for three main reasons:

1. The need to filter out duplicated reads and excess information in order to avoid pushing information that is not needed to the upstream applications, while at the same time optimizing network resources.

2. The need to interface and deal with readers, tags and devices in a heterogeneous multi-vendor environment without resorting to custom integration logic.

3. The need to pass and route RFID data streams to different applications and databases.

2.2 ASPIRE objectives and middleware architecture

In order to fill the gap in the development of middleware for RFID applications, ASPIRE consortium aims at developing an open-source, programmable, light-weight and scalable middleware platform. Furthermore, in order to deliver a solution that best matches current market trends, ASPIRE consortium has carried out a complete study on the requirements of end users via information days, workshops, online questionnaires and training sessions. Among the main results,

the consortium found about two main points: 1) the lack of knowledge of SMEs about the potential implications of this new technology; and 2) SMEs still perceive RFID as an expensive solution as compared to legacy systems such as optical bar scanners [22]. In addition, as a way to follow up and advance the state-of-the art of middleware platforms, ASPIRE partners have carried out a complete evaluation of existing middleware platforms, either open source or proprietary [23]. The outcome of this research served the partners of the project to have an idea of the needs and gaps in this area, and at the same time to identify open-source software modules or innovative architectures that could be reused or further upgraded.

As the result of this process, ASPIRE has defined a middleware architecture that is fully compliant with the EPC suite of standards [8] (see Figure 1). The EPC set of standards is mainly concerned with the processing of data in centralized RFID architectures [10]. This means that this set of standards assumes that a central node or core is in charge of the coordination, configuration, collection and other functions of the platform. In Figure 1 we can observe the architecture framework proposed by EPC. The modules are divided into three general functionalities: Identification, capturing and exchange. These functionalities more or less resemble those of the OSI (Open System Interconnection) model. For example the identification modules are concerned with the format and mapping rules to handle identification strings and other parameters required by upper layer protocols. The capture functionality covers physical, medium access control and link layer parameter definition (e.g. the tag protocol). Going up in the architecture reference model in Figure 1, the capture functionality also includes reader protocols that define the rules, cycles, and report formats for reader or interrogators to pass information to the middleware platform. The main core of the middleware, namely the filtering and collection module, is then defined by the ALE (Application Level Event) standard, and further complemented for the communication with upper business layers by the EPC Information Services (EPCIS) standard.

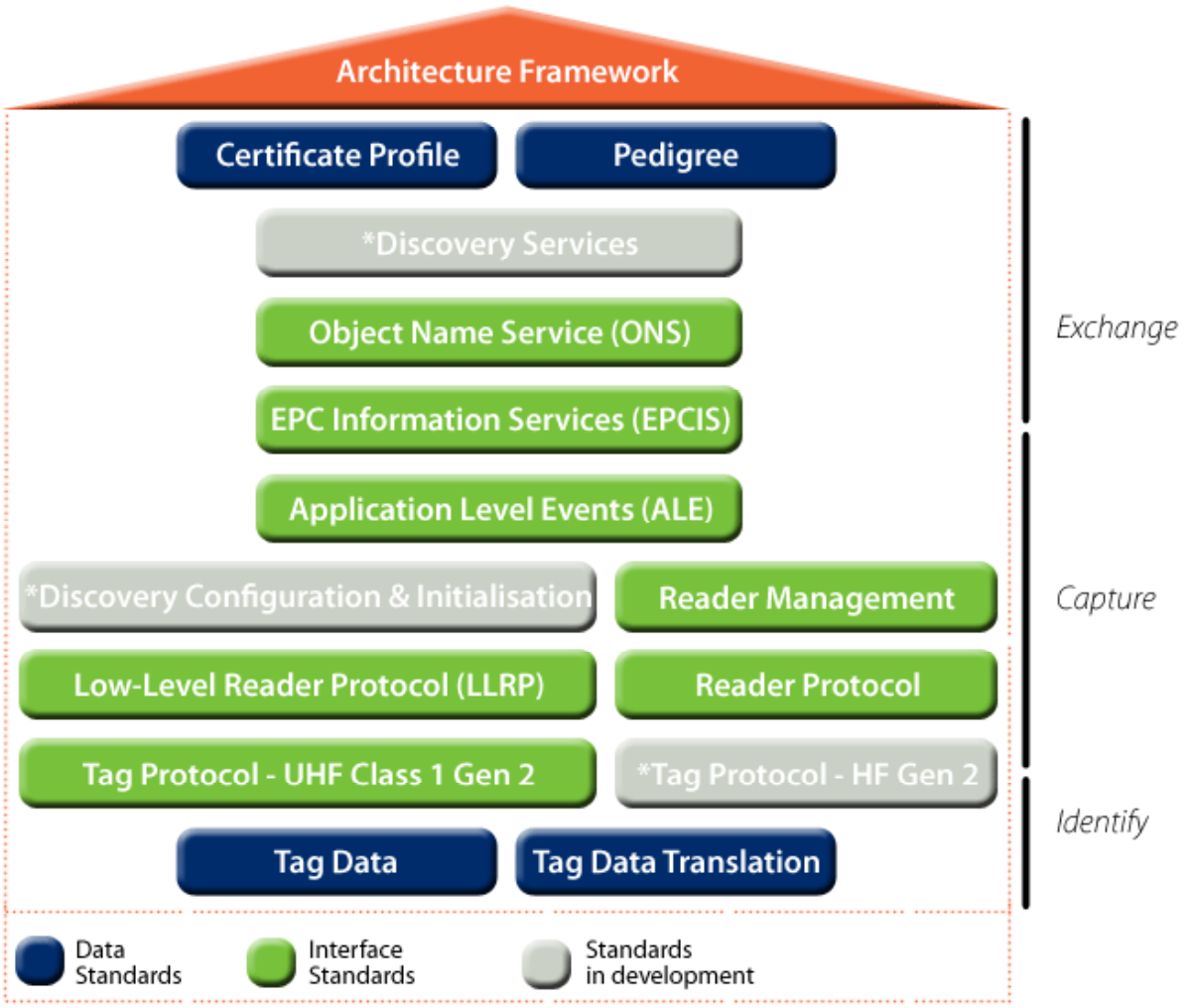

Figure 1: EPCglobal Architecture Framework

The ASPIRE architecture, which is fully described in deliverable D2.3 b [21] and which is depicted for the sake of clarity in Figure 2, implements the set of EPC standards and complements it by a set of added value features. From right to left, Figure 2 exposes the main components of the architecture. First, a heterogeneous landscape of readers from different providers is displayed. Note that readers that are not under EPC or ASPIRE standards are connected to the platform via a HAL or hardware abstraction layer, which basically converts proprietary into ASPIRE semantics, and vice versa. Readers that deploy EPC reader protocols are directly connected via RP or LLRP to the Filtering and collection (F&C) server, which in turn implements the ALE interface standard to upper layers. The F&C server filters unwanted data and forwards refined streams to different subscribers. In ASPIRE the main subscriber is the BEG (Business Event Generator), which interconnects the F&C and the EPCIS modules and which constitutes and added value solution provided by ASPIRE. The EPCIS module is finally connected to particular end user applications via, for example, web-services. A key element in the ASPIRE architecture is the Integrated Development Environment, which allows a rapid and efficient management of the

ASPIRE middleware platform. The management solution is not part of EPC standards and hence it also constitutes an added value solution of ASPIRE.

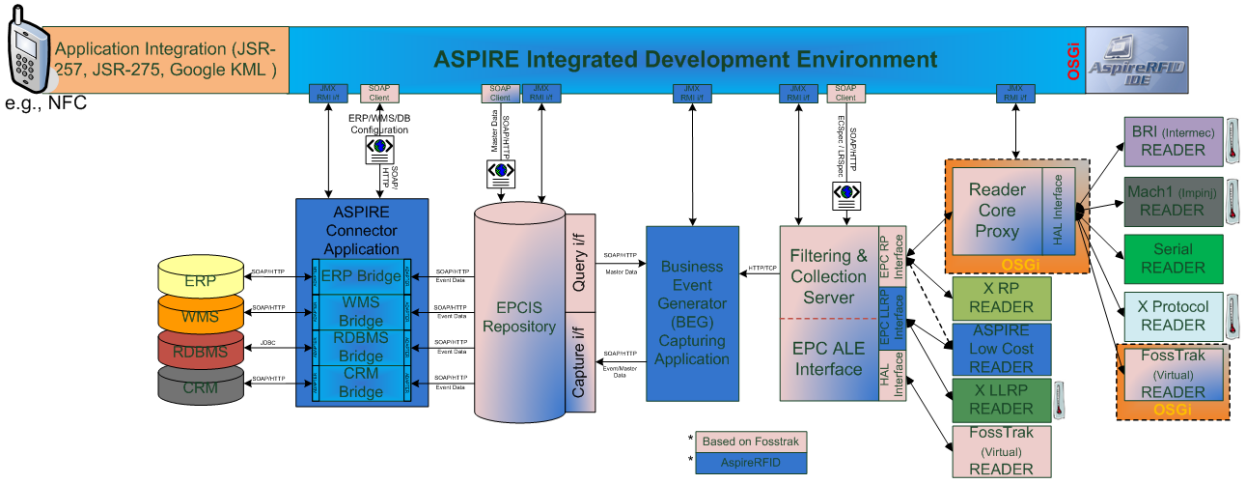

Figure 2 ASPIRE Architecture for Programmability, Configurability and End-to-End Infrastructure Management

2.3 ASPIRE innovative filtering solution

As observed in previous subsections, filtering has a prominent position in the RFID middleware blocks. Legacy middleware products concentrate on

- Low-level filtering (Tags, Tag Data)
- Aggregation of readings
- Provision of basic low-level application events

Advancing on this legacy features, ASPIRE introduces a new approach to RFID middleware through a two-tier filtering:

- Conventional filtering (e.g., EPC-ALE paradigm)
    - Open Source Tools (Stored/Save, Edit, Delete Filters) compliant to ALE specifications

- Filtering of business events (i.e. based on the paradigm of BEG module)
    - Combination of filtered data with business metadata according to declared/configured processes

    - Specifications for mapping sensor reading events into business events

- Filtering of many types of sensors other than RFID, like ZigBee (IEEE 802.15) and HF sensors.

At the time of writing the DoW (Description of Work) [24], low-level filtering functions had not been standardized. Hence we extended the scope of programmable filters to business events with the Business Event Generator (BEG) enabling programmable translation of basic filtered application events to

high level. Despite this, low level filtering functionalities, which will release the middleware platform from some of its filtering functionalities and tag traffic to be processed, have started to be investigated in the ASPIRE project (see deliverables within work-package 3, e.g. D3.3 [25])

2.4 The ASPIRE FML (Filtering Markup Language)

The filtering functionality in an RFID platform is not only used to get rid of extra information that is not relevant for upper layers, but it represents the connection between the low level RFID world, and the business and application level semantics, therefore being a critical point for middleware integrators and developers. To provide a clear consensus for open source contributors around this important interface, a straightforward solution is to use a high level programming language oriented to describe business semantics and to isolate them from the low level details of RFID platforms. Among such languages, one that has received special interest due to its flexibility and great acceptance between Internet and application developers is the extensible markup language (XML). XML is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C (world wide web consortium) and several other related specifications; all are fee-free open standards. As the name suggests, by using a set of markups, the language is able to be adapted to a variety of purposes, including the filtering functionality of an RFID system as described in this document. The filtering markup language proposed by ASPIRE not only helps in the programmability of the tool but it also provides modularity and the possibility of reusing filtering rules. In this way future developers can start building up new and interesting filtering policies from previously tested and mature solutions. Code reusability has been widely used in the software world, for example in the context of compilers for object oriented languages, and in the open source community itself. Reuse of filters with business meaning will allow SMEs and untrained persons on RFID to deploy new applications in short periods of time and at low cost.

2.5 Scope and organization of the document

This deliverable is dedicated to the specification and description of the filtering markup language, its semantics, the type of reusable filters that arise from its specification, the components of the ASPIRE architecture that are involved on the defined filtering functionalities, and useful usage examples that help the integrator understand the operation and structure of the language format and the functionalities addressed in the filtering definition. Also note that this document is the final version of the deliverable, thus being the culmination of the preliminary specifications released in month 16 (M16) of the project (D4.3a in [26]).

The present document is the report, whereas the implementation can be found at the ASPIRE's Wiki and Forge Pages. The executable final version of the implementation can be found at the AspireRFID forge page (http://forge.ow2.org/project/showfiles.php?group_id=324), the source code of the implementation can be found at the AspireRFID SVN

(http://forge.ow2.org/plugins/scmsvn/index.php?group_id=324). Directions on how to use the AspireRFID Information Sharing repository, Business Event Generation and F&C module can be found at the AspireRFID Wiki documentation page (http://wiki.aspire.ow2.org/xwiki/bin/view/Main/Documentation).

The rest of this deliverable is structured as follows:

- Section 3 discusses the role of the programmable filters and their impact on the Aspire middleware architecture.
- Section 4 discusses the concept and the operation of the programmable filters from a high level design perspective.
- Section 5 reviews the role of the Business Event Generator between the low level event generator and the high level Information Systems
- Section 6 dives deep into the Programmable Filters Specification presenting the three distinct components that assemble it.
- Section 7 discusses how these three components are combined to produce business information that can be utilized by the high level Information Systems.
- Section 8 explores the tools that have been built in the scope of Aspire to implement the components of the Programmable Filters Specification
- Section 9 provides a complete example that demonstrates the use of the aforementioned tools
- Section 10 gives an overview of low level filtering (filtering deported to the reader device) which is the completed version of the specification previously provided in D4.3a and finally
- Section 11 draws the main conclusions on the matters raised in this deliverable.

Section 3 Concept of Reusable filters

It is relatively straightforward for a trained person or an expert RFID programmer to develop a new or adapt an existing RFID middleware platform according to specific requirements and with the support of specific functions. However, having such an expert RFID developer available for expanding the system and adding new functions/features is difficult and expensive, particularly for small and medium enterprises (SMEs).

A possible solution for this problem is the concept of reusable filters. By setting specific filters for the RFID middleware using XML language (Extensible Markup Language) and by using a suitable "engine" which would be able to interpret XML semantics, one would be able to describe to that "engine" the requirements and processes of an RFID infrastructure without the need of an expert RFID developer. Thus, we would be able to set up this engine to serve specific company operations without a lot of extra-effort. These filters can then be reused in different application domains and by different end-users, as long as the targeted processes are more or less similar. This reusability feature provides the filtering module and in general the middleware platform with a high degree of modularity, flexibility and programmability, all features that differentiate ASPIRE from previous approaches.

Reusing code or pieces of code is, however, not new in the software world. Compilers for object oriented programming languages exploit the concept of reusing already compiled pieces of code as a means to save memory and improve efficiency. Open source software communities themselves are also based on a kind software reuse feature. In our case, reusable filters further alleviate one of the main problems of initial middleware deployments that had to be designed on a per-case basis. Filters can now be translated to many other application scenarios with minimum effort and reduced cost. The reusability concept directly derives from the ability to break off into distinct business transactions a company's business processes, which in turn can be broken up into distinct Transaction Events. Figure 3 below illustrates the example of the supply of consumer items (in this case bottles) all the way from the moment they are shipped at the factory, going through the warehouse premises, up to the shopping centre. This entire process is called Business Transaction or Composite Business Process (see Figure 3). This extended business transaction can be broken up into other three business transactions, each for a different physical location. The transaction in the factory in Figure 3 can be further divided into three transaction events or elementary business processes: Commission of bottles, Pack Bottles into case and Shipment of the Tote. Note that the nature of the first and third events is similar, hence they will be called Object events, whereas the second one will be called Aggregation event. A similar partition can be done in the transaction events at the Warehouse and at the Shopping centre. Note that at the warehouse, reception of the cases, shipment of the tote, as well as moving and storing cases in the warehouse can be regarded as Object events. On the contrary, Picking and Packing is classified as aggregation event. Finally,

at the shopping centre, reception and unpacking of the tote can be regarded as object and disaggregation events, respectively.

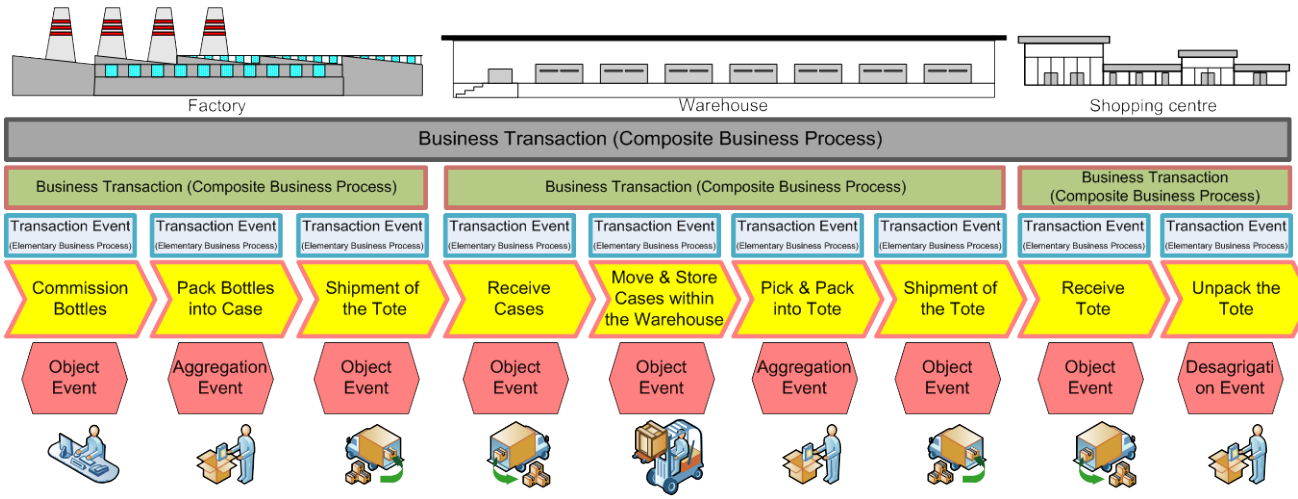

Figure 3 Wide Business Process/Transactions Example

The key point in the example of Figure 3 is that it is important to break up each use case into a series of discrete business steps corresponding to various business events so as to be able to reuse each one of them to describe a different scenario. This also depends on the designer's ability to abstract in a proper way all the events of a business process in a common format that resembles almost exactly other processes in different applications domains.

Fixed lists of identifiers with standardized meanings for concepts like business step and disposition must be defined, along with rules for population of user-created identifiers like read point, business location, business transaction and business transaction type. All these information elements will be stored and managed as pieces of Master Data within an appropriate database schema. Master data is the name given to the set of properties or additional parameters that are used to interpreting an event of a business process (see definition later in this document).

Figure 4 depicts another example of the concept of decomposing a process into a number of business events. The latter events comply with the ASPIRE Information Sharing specifications for RFID events (with direct references to EPC-IS framework). We call Elementary Business Process to the process that can be directly decomposed into RFID business events (as shown in Figure 4).

The business events shown in Figure 4 are transaction start, object event (such as receive are ship totes in the previous example), transaction observed, aggregation event (such as picking and packing bottles in the previous example) and transaction finish. Note that these business events completely describe the given business process.

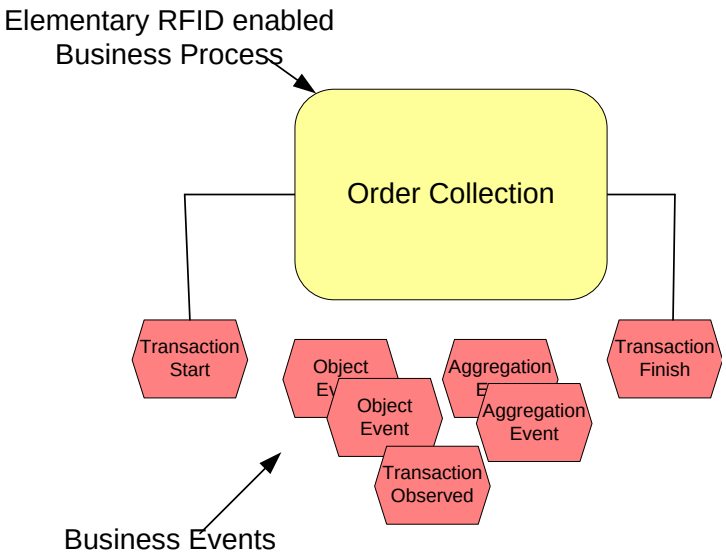

Figure 4 Description of Elementary RFID enabled Business Process

These “Business Events” are stored at the Business’s Master Data which except the business step, disposition, read point, business location, business transaction and business transaction type. It should also define the required input needed from the underlying Filtering and collection layer so as to create the current RFID events. Details about the Master Data and Master Data Editor (MDE) are given in section 6 and section 8, respectively.

Section 4 Notion of Programmable Filters in the ASPIRE Architecture

Because ASPIRE is designed in such a way that it will be expandable, configurable and modular, the use of Programmable filters arises as a natural feature. Moreover ASPIRE IDE (Integrated Development Environment) will include a programmability engine that will be able to process a fully fledged RFID solution described in a special purpose domain specific language. This language will be specified as part of future deliverables of WP4 in the ASPIRE project.

As mentioned in Section 3, the reusable filters and their proposed filtering markup language (FML) need a "core" engine that interprets its semantics and that executes the rules, policies or commands given in their parameter data fields. In the AspireRFID middleware, this core "engine" consists of three different components of the architecture (see also Figure 2 in section 2.2 to see the overall ASPIRE architecture):

- The filtering and Collection (F&C) module, which is responsible of the low level filtering. We recall that low level filtering simply deals with getting rid of duplicated tag readings.
- The Business Event Generator (BEG) module, which is responsible for the High Level filtering providing Business context to the captured events.
- And the Information sharing (IS) repository, which is the repository which stores a company's Master Data and Business functions.

Figure 5 demonstrates a complete Aspire programmable filter solution with ASPIRE's existing tools. With the help of a Master Data editor (explained in detail in subsection 8.3) we can "describe" the company's business data, processes and the required Low Level input to create business events. Note that the Master Data Editor (top left corner of Figure 3) is under the scope of the ASPIRE IDE. Also shown in Figure 3 is an example of Master Data parameter values: ecreport_names, event_name, business_step, business_location, ecspecname, read_point and transaction_type. All these data are stored in the Information Sharing repository (bottom left of Figure 3, tagged EPCIS repository) which is used from the Business Event Generator on demand to configure its behavior on creating the business events. Note in Figure 3 that the BEG is interconnected to the EPCIS via two interfaces, one for capture and another for query. The query interface is the one used by the BEG module to request Master Data on demand from the EPCIS. With the help of ECSpec editor (top right of Figure 3) we can create the required ECSpecs that configure the Filtering and Collections layer behavior by using the ECSpec configurator (also called ALE server configurator in Figure 3). Also note that in Figure 3, the box tagged as Defined ECspec contains a code in XML format. The Filtering and Collection module (low right part of Figure 3) after being configured as required collects the raw readings from the attached RFID readers to it and produces the filtered ECReports which are fed back to the Business Event Generator. Event data is the reported back from the BEG to the EPCIS repository via the capture interface (see again bottom left part of Figure 3). A complete detailed example is described in Section 9.

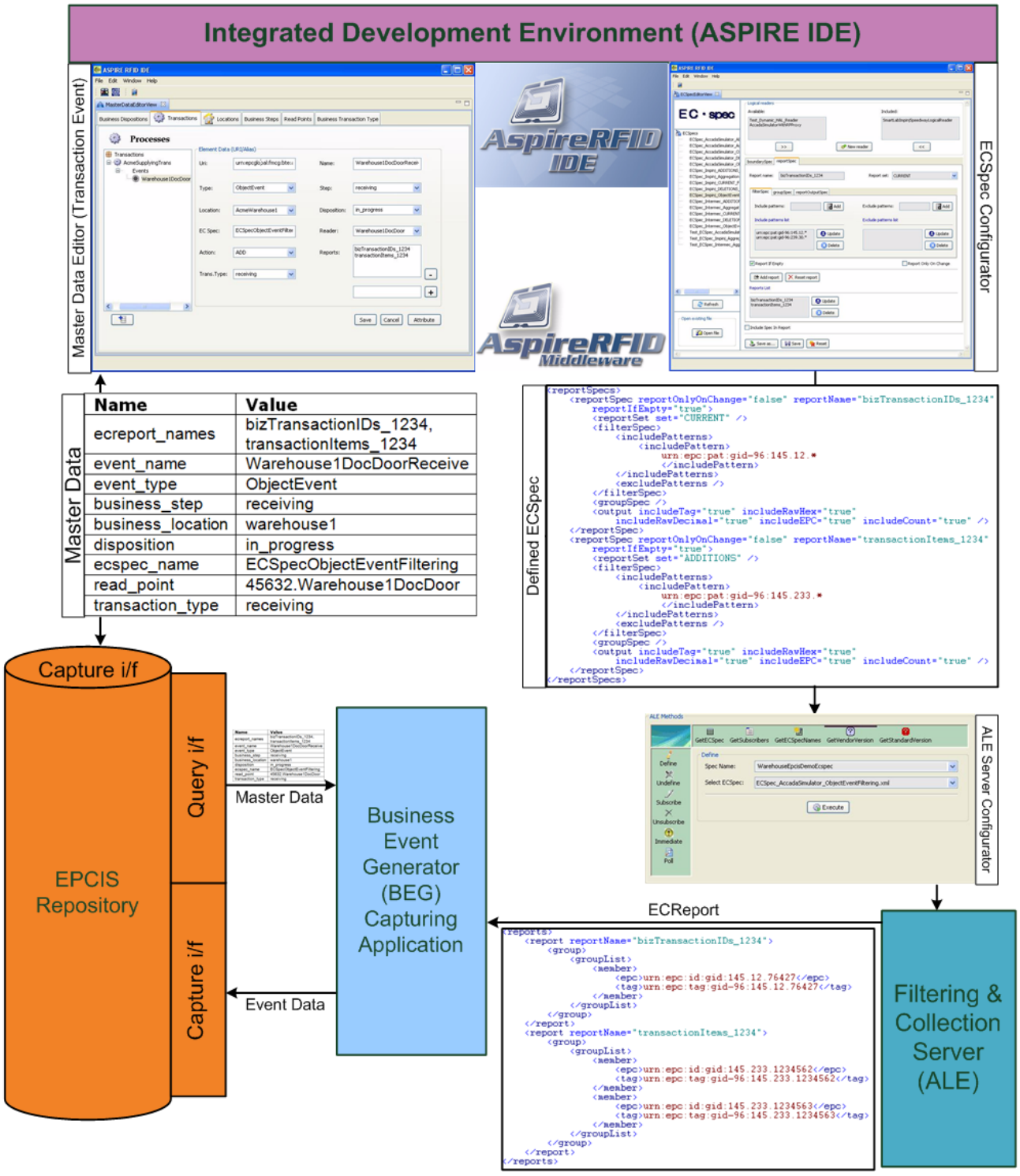

Figure 5 Complete Programmable Filters ASPIRE solution

Section 5 Business Event Generation: Connecting F&C and Information Services modules through Business Filters

Since the programmable filters described in this document are related to specific components of the ASPIRE architecture, it is useful then before proceeding to more technical specifications to understand, in simple words, their aim and how they interact with each other.

The Information Services module specification defines a data language for representing visibility information, namely events having four dimensions of "what", "when", "where" and "why".

Primarily, the Filtering and Collection module answers 'What', 'Where' and 'When'. Information Services module adds the 'Why' (i.e., the business context). For example, the 'Where' in Filtering and Collection module usually means a logical reader name (e.g. reader X). This is later converted into a business location in the Information Services module (e.g. Warehouse X).

Furthermore, the Filtering and Collection module interface is exclusively oriented towards real-time processing of Information Services data, with no persistent storage of these data required by the interface. Business applications (e.g. ERP, WMS, etc.) that manipulate Information Services module data, in contrast, typically deal explicitly with historical data and hence are inherently persistent in nature. More details of the F&C server can be found in deliverable D3.3 [25]

Architecturally, the Filtering and Collection layer is first concerned with the mechanics of data gathering, and then with filtering down such data to meaningful events that are a suitable starting point for interpretation by business logic. Business layers, where the Information Services module comes into play, are concerned with business process and recording events that can serve as the basis for a wide variety of enterprise-level information processing tasks.

Visibility information at the Information Services module level is often used to record what took place in an operational business process that involves the handling of physical assets, such as receiving goods through an entry door of a warehouse. The module responsible for supervising such a process and generating Information Services data is the Business Event Generation module.

The "glue" for the two modules described above, i.e. the information Service and the Filtering and collection modules, and the one that collects the produced F&C data and adds the "why" notion is the Business Event Generator module. To the extent that the Business Event Generation module interacts with EPC data and/or RFID tags in the course of carrying out its function, it uses ALE as the way to read those EPC data and/or RFID tags and the Information Sharing to store these "events".

In most of the cases the Business Event Generation module is also responsible for a complex orchestration of RFID devices, material handling equipment, and

human tasks that are involved in carrying out a business process. That is why Filtering and Collection is used specifically to interact with the RFID devices, and therefore it addresses a reduced scope. [4]

Section 6 Programmable Filters Specification (the different components)

6.1 Overview

To create programmable filters that contain a complete business logic throughout the ASPIRE's middleware we need to combine three different specifications together. These specifications apply to three different modules of the ASPIRE middleware architecture (for details of the architecture see deliverable D2.3b [21] and the introductory section of this document) which are:

- the Filtering and collection module,
- the Business Event generation module and
- the Information Services repository module.

So to analyze the combination of the specifications of those three modules first we need to study the components that are used from each one separately and then specify the way in which they interact.

Note that in this section we are going to define the specifications for only the required parts from the all ASPIRE's architecture. For detailed specifications of each module we refer the reader to other deliverables of the consortium, for example D3.3 [25] for the specifications of data collection, filtering and application events.

6.2 Filtering and Collection module

6.2.1 Role

The filtering and collection module (F&C) carries out processing to reduce the volume of EPC data, transforming raw tag reads into streams of events more suitable for application logic than raw tag reads. From the practical point of view, the F&C module interacts with upper layer clients that are subscribed to its services by defining report cycles and report data formats. These report data formats define the type of data to be exchanged, while the cycles define the way in which these data will be exchanged or retrieved and how often. The main specifications that define these procedures are the ECSpecs, and within it, the ECReportSpec. These are briefly described in the following subsections.

6.2.2 ECSpecs

An ECSpec is a complex type that describes an event cycle and one or more reports that are to be generated from it. Current tags or tags that have been added or deleted can be retrieved with respect to the last event cycle or combinations of all. [3]

An ECSpec Contains (see Figure 6):

- An unordered list of Logical Readers called "*logicalReaders*" whose reader cycles are to be included in the event cycle and are used to acquire tags.

- A specification of how the boundaries of event cycles are to be determined called "*boundarySpec*". In brief, it specifies the starting and stopping conditions for event cycles.
- An unordered list of Report Specifications, each describing a report to be generated from this event cycle and to be included in the output from each event cycle called "reportSpecs".

For defining filters the most important part of an ECSpec is the ECReportSpec.

6.2.2.1 ECReportSpec

An ECReportSpec specifies one report to be included in the list of reports that results from executing an event cycle. An ECSpec contains a list of one or more ECReportSpec instances. When an event cycle is completed, an ECReports instance is generated, unless suppressed. An ECReports instance contains one or more ECReport instances, each corresponding to an ECReportSpec instance in the ECSpec that governed the event cycle (see Figure 6). The ECReportSetSpec is an enumerated type denoting what set of Tags is to be considered for filtering and output: all Tags read in the current event cycle, additions from the previous event cycle, or deletions from the previous event cycle (see Figure 6).

An ECReportSetSpec contains one or more ECFilterSpec which specifies the Tags to be included in the final report. The ECFilterSpec implements a flexible filtering scheme based on two pattern lists. Each list contains zero or more URI (Universal Resource Identifier)-formatted EPC patterns. Each EPC pattern denotes a single EPC, a range of EPCs, or some other set of EPCs.

An EPC is included in the final report if

a) the EPC does not match any pattern in the excludePatterns list, and
b) the EPC does match at least one pattern in the includePatterns list.

The (b) test is omitted if the includePatterns list is empty. [3]

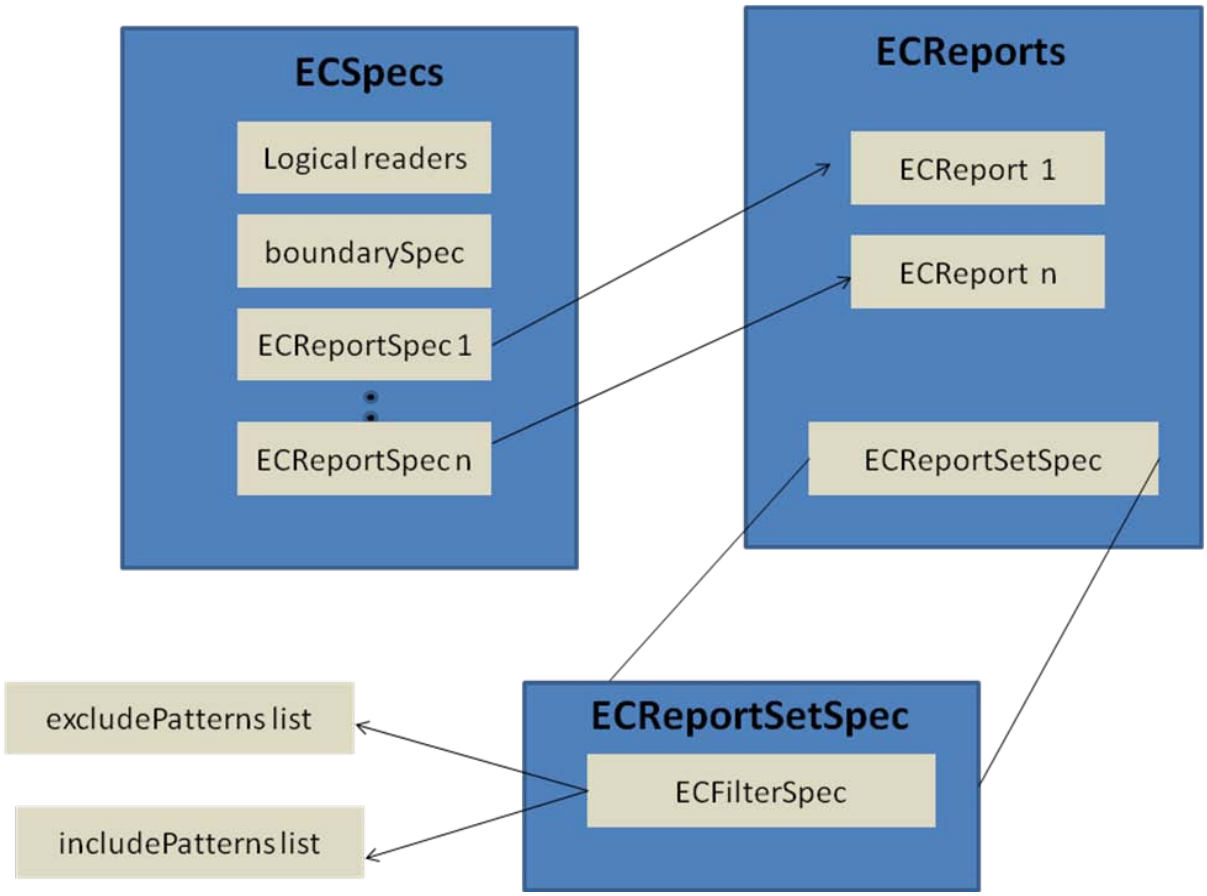

Figure 6 ECSpecs and related fields

6.3 Information Services module

6.3.1 Role

The ASPIRE Information Sharing repository is responsible for receiving application-agnostic RFID data from the filtering & collection middleware through the Business Event Generation (BEG) application and store the translated RFID data in corresponding business events. These events carry the business context as well (e.g., they refer to particular companies, business locations, business processes etc.). Moreover it makes business events and master data available and accessible to other upstream applications through the query interface.

Generally, the ASPIRE information sharing repository is dealing with two kinds of data:

- RFID event data i.e. data arising in the course of carrying out business processes. These data change very frequently, at the time scales where business processes are carried out.
- Master/company data, i.e. additional data that provide the necessary context for interpreting the event data. These are data associated with the company, its business locations, its read points, as well as with the business steps comprising the business processes that this company carries out.

At a glance Information Services of the ASPIRE Information Sharing middleware consists of three parts: a capture interface that provides web services for storing data, a repository that provides persistence, and query interface that provides web services that retrieves the business events/master data from the repository.

6.3.2 Event Data

Event data arises in the course of carrying out business processes. Event data grows in quantity as more business is transacted, and refers to things that happen at specific moments in time.

6.3.2.1 EPCIS Event

An EPCISEvent is a generic base class for all event types which provides date and time fields (see strings “eventTime” and “recordTime” in Figure 7). Below is given the EPCISEvent’s XML schema [2][6] and the different events are described in the following subsections (see also Figure 8 for a schematic representation of the different types of events.)

```
<xsd:complexType name="EPCISEventType" abstract="true">
      <xsd:sequence>
            <xsd:element name="eventTime" type="xsd:dateTime" />
            <xsd:element name="recordTime" type="xsd:dateTime"
minOccurs="0" />
…
      </xsd:sequence>
      <xsd:anyAttribute processContents=”lax” />
</xsd:complextype>
```

Figure 7 EPCISEvent’s XML schema

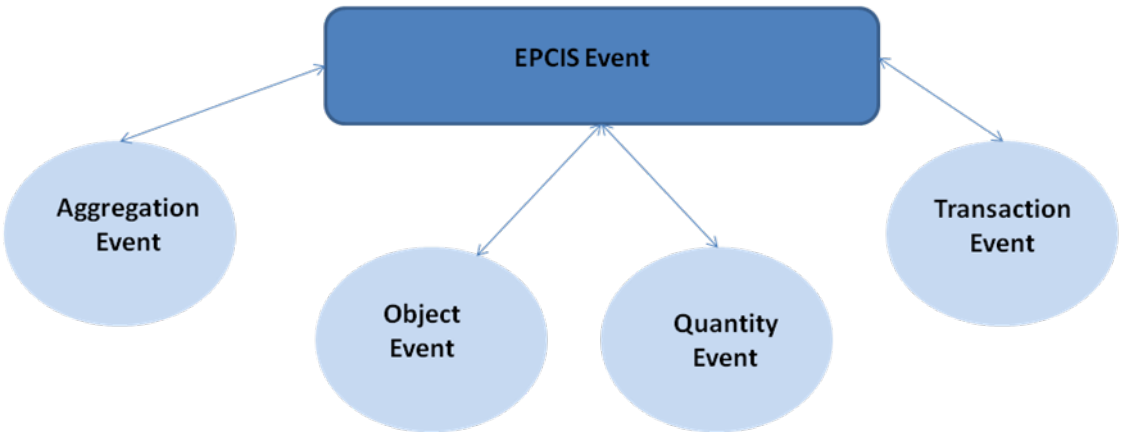

Figure 8 EPCIS Events

6.3.2.2 Aggregation Event

An AggregationEvent describes events related to objects that have been physically aggregated. In such an event, there is a set of contained objects that have been aggregated within a containing entity which identifies the physical aggregation itself. The contained objects are called children and the containing entity is called parent.

Because an AggregationEvent indicates aggregations among physical objects, the children are identified by EPCs. However, the parent entity is identified by an arbitrary URI (which may or may not be an EPC) because the parent is not necessarily a physical object that is separate from the aggregation itself. Below is given the AggregationEvent's XML schema. [2][6]

```
<xsd:complexType name="AggregationEventType">
      <xsd:complexContent>
            <xsd:extension base="epcis:EPCISEventType">
                  <xsd:sequence>
                        <xsd:element name="parentID"
                        type="epcis:ParentIDType"
                              minOccurs="0" />
                        <xsd:element name="childEPCs"
                        type="epcis:EPCListType" />
                        <xsd:element name="action" type="epcis:ActionType"
                        />
                        <xsd:element name="bizStep"
                        type="epcis:BusinessStepIDType"
                              minOccurs="0" />
                        <xsd:element name="disposition"
                        type="epcis:DispositionIDType"
                              minOccurs="0" />
                        <xsd:element name="readPoint"
                        type="epcis:ReadPointType"
                              minOccurs="0" />
                        <xsd:element name="bizLocation"
                        type="epcis:BusinessLocationType"
                              minOccurs="0" />
                        <xsd:element name="bizTransactionList"
                        type="epcis:BusinessTransactionListType"
                              minOccurs="0" />
                        …
                  </xsd:sequence>
                  <xsd:anyAttribute processContents="lax" />
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>
```

Figure 9 AggregationEvent's XML schema

6.3.2.3 Object Event

An ObjectEvent captures information about an event pertaining to one or more physical objects identified by EPCs.

Logically, an ObjectEvent pertains to a single object identified by an EPC. However, you can specify more than one EPC in an epcList when the remaining ObjectEvent data applies to all the EPCs in the list.

In an ObjectEvent, no relationship among the EPCs is implied by their appearance in the same ObjectEvent other than the coincidence of them all being captured with identical information. By contrast, an AggregationEvent or TransactionEvent conveys an implicit association among the EPCs in the event. Below is given the ObjectEvent's XML schema. [2][6]

```
<xsd:complexType name="ObjectEventType">
<xsd:complexContent>
      <xsd:extension base="epcis:EPCISEventType">
            <xsd:sequence>
                  <xsd:element name="epcList" type="epcis:EPCListType" />
                  <xsd:element name="action" type="epcis:ActionType" />
                  <xsd:element name="bizStep"
                  type="epcis:BusinessStepIDType"
                        minOccurs="0" />
                  <xsd:element name="disposition"
                  type="epcis:DispositionIDType"
                        minOccurs="0" />
                  <xsd:element name="readPoint" type="epcis:ReadPointType"
                        minOccurs="0" />
                  <xsd:element name="bizLocation"
                  type="epcis:BusinessLocationType"
                        minOccurs="0" />
                  <xsd:element name="bizTransactionList"
                  type="epcis:BusinessTransactionListType"
                        minOccurs="0" />
                  …
            </xsd:sequence>
            <xsd:anyAttribute processContents="lax" />
      </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

Figure 10 ObjectEvent's XML schema

6.3.2.4 Quantity Event

A QuantityEvent is an event that happens to a specified number of objects all having the same type, but where the individual instances are not identified. Quantity Events can serve as a bridge between RFID systems and legacy inventory systems that do not identify individual items. Below is given the QuantityEvent's XML schema. [2][6]

```
<xsd:complexType name="QuantityEventType">
      <xsd:complexContent>
            <xsd:extension base="epcis:EPCISEventType">
                  <xsd:sequence>
                        <xsd:element name="epcClass"
                        type="epcis:EPCClassType" />
                        <xsd:element name="quantity" type="xsd:int" />
                        <xsd:element name="bizStep"
                        type="epcis:BusinessStepIDType"
```

```
                        minOccurs="0" />
                    <xsd:element name="disposition"
                    type="epcis:DispositionIDType"
                        minOccurs="0" />
                    <xsd:element name="readPoint"
                    type="epcis:ReadPointType"
                        minOccurs="0" />
                    <xsd:element name="bizLocation"
                    type="epcis:BusinessLocationType"
                        minOccurs="0" />
                    <xsd:element minOccurs="0"
                    name="bizTransactionList"
                        type="epcis:BusinessTransactionListType" />
                    …
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax" />
            </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

Figure 11 QuantityEvent's XML schema

6.3.2.5 Transaction Event

A TransactionEvent describes the association or disassociation of physical objects to a business transaction. While other event types have an optional bizTransactionList field that can be used to provide context for an event, the TransactionEvent is used to declare in an unequivocal way that certain EPCs have been associated or disassociated with one or more business transactions as part of the event. Below is given the TransactionEvent's XML schema. [2][6]

```
<xsd:complexType name="TransactionEventType">
      <xsd:complexContent>
            <xsd:extension base="epcis:EPCISEventType">
                  <xsd:sequence>
                        <xsd:element name="bizTransactionList"
                        type="epcis:BusinessTransactionListType" />
                        <xsd:element name="parentID"
                        type="epcis:ParentIDType"
                            minOccurs="0" />
                        <xsd:element name="epcList"
                        type="epcis:EPCListType" />
                        <xsd:element name="action" type="epcis:ActionType"
                        />
                        <xsd:element name="bizStep"
                        type="epcis:BusinessStepIDType"
                            minOccurs="0" />
                        <xsd:element name="disposition"
                        type="epcis:DispositionIDType"
                            minOccurs="0" />
                        <xsd:element name="readPoint"
                        type="epcis:ReadPointType"
                            minOccurs="0" />
                        <xsd:element name="bizLocation"
                        type="epcis:BusinessLocationType"
                            minOccurs="0" />
```

```
                    …
                </xsd:sequence>
                <xsd:anyAttribute processContents="lax" />
            </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

Figure 12 TransactionEvent's XML schema

6.3.3 Actions Types

The Action type says how an event relates to the lifecycle of the entity being described. The Action type has three possible values (see Figure 13):

- **Add** which means that the entity in question has been created or added to.
- **Observe** which means that the entity in question has neither been changed, nor has it been created, added to, destroyed or removed from.
- **Delete** which means that the entity in question has been removed from or destroyed altogether.

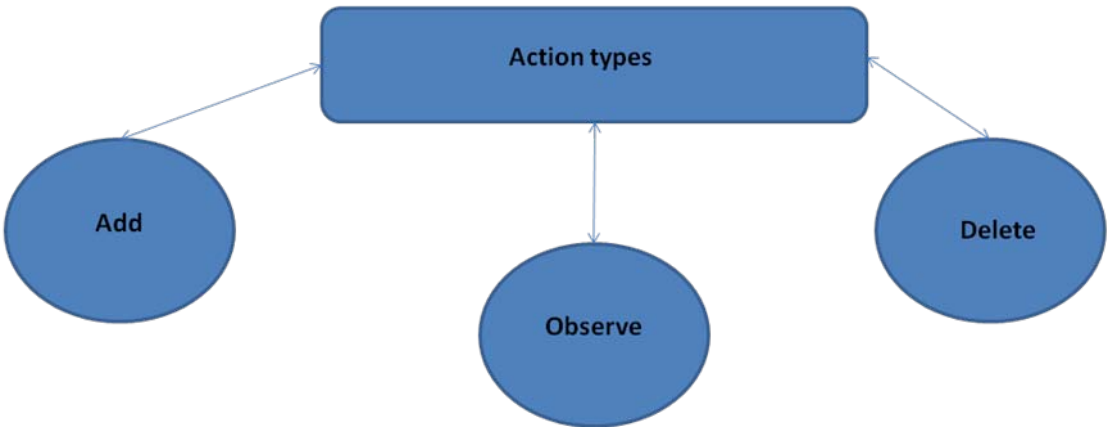

Figure 13 Action types

6.3.4 Master Data

Master data is additional data that provides the necessary context for interpreting event data. It is available for filtering a query through the EPCIS Query Interface, and available as part of a report through the Reporting Service.

Master data does not grow merely because more business is transacted. It is not typically tied to specific moments in time and provides interpretation for elements of event data.

Master data is business-context information that is associated with event data by the Business Event Generation module reporting service and by the data exchange service.

6.3.4.1 Master Data Types

A master data *type* is a definition for master data *entries* of that type. Each master data type defines a set of *attributes* and their data types.

A master data entry is a concrete instance of a master data type. You can create as many entries as you need of each master data type. Each entry has the same set of attributes defined for its master data type, but where the master data type defines the data type for each attribute an entry's attributes contain the real business-context information associated with a business's operations. [2][6]

The available master data types are (see also figure below):

- bizLocation
- bizStep
- bizTransactionList
- bizTransaction which is composed of
  - Business Transaction and
  - Business Transaction Type
- disposition
- epcClass
- readPoint

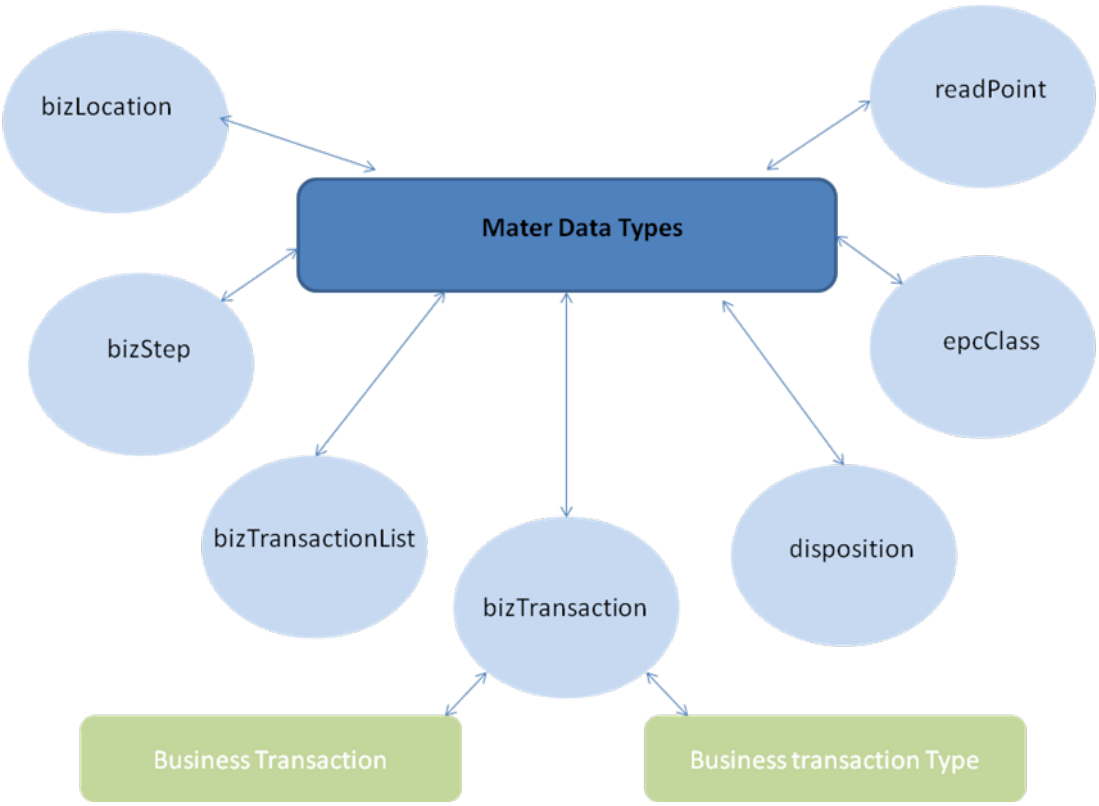

Figure 14 Master Data types

The master data type that mostly concerns us to define the required programmable filter is the "*BusinessTransaction*" type which is next analyzed.

6.3.4.1.1 BusinessTransaction

A BusinessTransaction field identifies a particular business transaction. Transaction information may be included in EPCIS events to record an event's participation in particular business transactions. [6]

A business transaction is described in Information Services module by a structured type consisting of the following pair of identifiers:

- BusinessTransactionTypeID
- BusinessTransactionID

BusinessTransactionID is a vocabulary whose elements denote specific business transactions. In Table 1 below the BusinessTransactionID's attributes are shown. The attribute that are most significant to create an EPCIS event are:

- The "*ECReportNames*" one which stores a list of the incoming ECReport names, to the Business Event Generator module, which concerns the event to be created.
- The "*EventType*" which denotes the type of the event (Aggregation Event, Object Event, Quantity Event or Transaction Event)
- And the "Action" which denotes how an event relates to the lifecycle of the entity being described.

Attribute Name	Attribute URI
ECReportNames	urn:epcglobal:epcis:mda:ecreport_names
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 1 Business Transaction ID Attributes

6.4 Business Event Generation module

The architecture introduces a Business Event Generator (BEG) module between the F&C and Information Sharing (e.g., EPC-IS) modules as shown in Figure 15. 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 business-context information (Master Data) with event data. The data is stored in the Information Service module repository.

6.4.1 Role

BEG module recognizes the occurrence of EPC-related business events, and delivers these as EPCIS data. It may coordinate multiple sources of data in the course of recognizing an individual EPCIS event. Sources of data include filtered, collected EPC data obtained through the Filtering & Collection Interface.

6.4.2 Functionality

In order for BEG to create the business events it needs the appropriate information from the repository. Hence, the EPCIS standard defines all the information that the EPCIS events encapsulate. This data necessary for the proper population of the EPCIS events are retrieved from the EPCIS repository, and more specifically the data defined at the BusinessTransaction's Attributes, using the query interface. This predefined information is entered and managed using the master data editor as shown in Figure 15 below (this process is indicated by the top arrow entering the EPCIS repository and indicates the definition of the elementary business processes). With the use of the MDE editor (paragraph 8.3) the end-user organizations can properly populate the vocabularies so as make the event generation possible. The MDE operates over the EPCIS layer also and could be considered an implementation of an accessing application according to the EPC Network Architecture terminology.

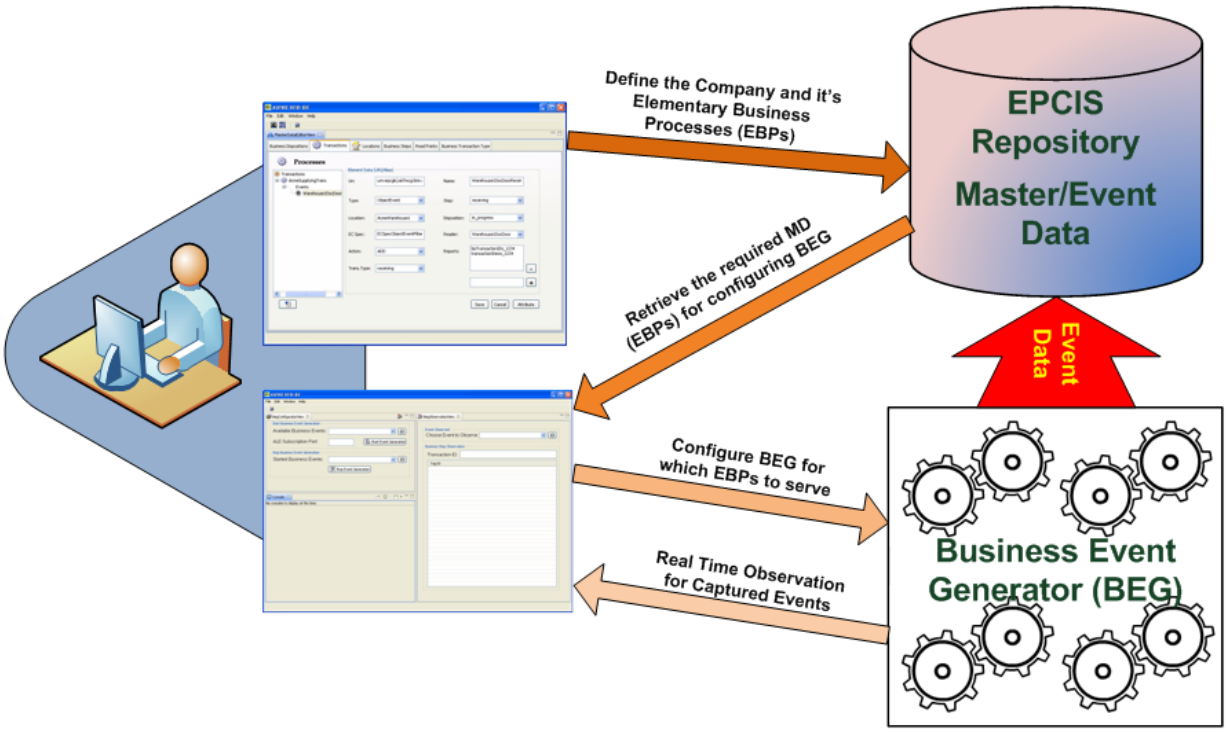

Figure 15 Role of the BEG Configurator and MDE tools

As already outlined BEG is a capturing application: BEG stores the data in the EPCIS repository as event data in an automated fashion, thereby adding business context to RFID readings in order to transform them to the target

business semantics as required by the particular business case associated with an overall RFID deployment. In order to create business context, BEG relies on a description of the target business processes, which should be described in the EPCIS Master Data. Each process is defined and described based on the composition of a discrete series of business steps. According to the EPCIS specification, each business step can be modeled by one EPCIS event of the specification. In particular, each business process has an identifier in the Business Transaction Vocabulary and is associated with a number of events. Each event is characterized as shown in Table 1 above by the type of the event, the event's action, the business location that the event takes place, the event's business step, the captured items disposition, the read point which the action took place and finally the required report names.

The attributes mentioned above constitute all of the information required in order to properly create an EPCIS event. Recall that based on such master data type, BEG is able to transform EC Reports from ALE into EPCIS events that can be stored in the EPCIS repository. Hence, each described event is associated with a list of Report names. The reports described in a specific ECSpec define all the required details about the Event Cycle and what kind of ECReports are to be delivered to the BEG. The BEG should be previously configured according to what specific report names is expecting from the F&C module which are defined in the business transaction vocabulary of the corresponding event[18].

In the context of Figure 15 the configuration process of the BEG module is indicated by two steps: first an arrow that goes from the EPCIS module to the editor and indicates the retrieval of the necessary information to configure the BEG, and second another arrow that goes from the editor to the BEG module and indicates the uploading of the configuration files to the BEG. Finally, real time capturing of events is displayed in the editor by obtaining real time information from the BEG and the BEG generates event data that are stored in the EPCIS module.

Section 7 Programmable Filters Specification (creating business logic)

7.1 Overview

While RFID readers detect and report tags along with a timestamp, it is not trivial to identify and populate properly business events based solely on a set of low-level RFID readings (i.e. tag streams). For example, there is a need to distinguish the identity of a containing entity from identities of the contained objects in the case of aggregations (e.g., a pallet carrying tagged objects that is in the range of an RFID reader). Most important, a business transaction as defined in the master data must be correlated with a specific transaction as is "monitored" by other applications. To address these needs we segment EPCs in classes. As an organization is assigned (from EPCglobal) and manages a pool of EPCs, it is possible to segment them into categories and reserve some of these categories for specific usage. Hence, a class of EPCs could be reserved for aggregating entities (e.g. pallets or boxes). The classification can then be used to populate the aggregation event. Another class may be reserved to denote a general category for all physical objects under detection, whereas another class can be used to signify documents that accompany business transactions (e.g., invoices). Also, a naming convention for the pattern of the report name is used. Specifically the implementation of BEG, understands a report name that starts with the string "bizTransactionIDs" as the name of the report carrying the EPC of the documents accompanying a process. Likewise, a report name starting with "parentObjects" is expected to carry the parent aggregating entity, while a report name starting with "transactionItems" is expected to carry the EPCs of the tagged items, etc. In the case only of quantity events, which do not have an EPC list and only the quantity of items affected by the event is of interest, this report is expected to contain the EPC class along with the number of items of the class.[18]

So as previously explained the "glue" from the three modules described in Section 6 to create business logic at the AspireRFID middleware is the BEG module, which uses the predefined data provided from the two others to produce the required EventData. Below the "event generation" notion is analyzed in more details and the relationship between the three middleware modules required for that will become clearer.

7.2 Combining ECSpecs & BizTransaction Attr to create Event Data

To create Event Data, some event fields are required and some are optional. Table 2 maps these associations. In addition, later on we are going to describe how we should set up our middleware to get these event fields and create the desired Event Data.

R = Required O = Optional	ObjectEvent	AggregationEvent	QuantityEvent	TransactionEvent
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 [2]

The sequence for creating the various Business Events shown in Figure 16 at the ASPIRE middleware is the following. The Filtering and Collection module receives the raw readings from the Logical Readers attached to it. The F&C module, in turn, processes the received readings taking in consideration the already predefined ECSpecs and delivers the produced ECReport to the Business Event Generation module. The Event Generation module receives the ECReport produced by the F&C module and processes them taking in consideration the BusinessTransaction attributes data from the already predefined Master Data of the company. Finally the Business Event Generation module sends to the Information Services module Capturing interface the produced event data where they are stored in a repository and are available for other applications, in our case the Connector application, to query through its Query Interface.

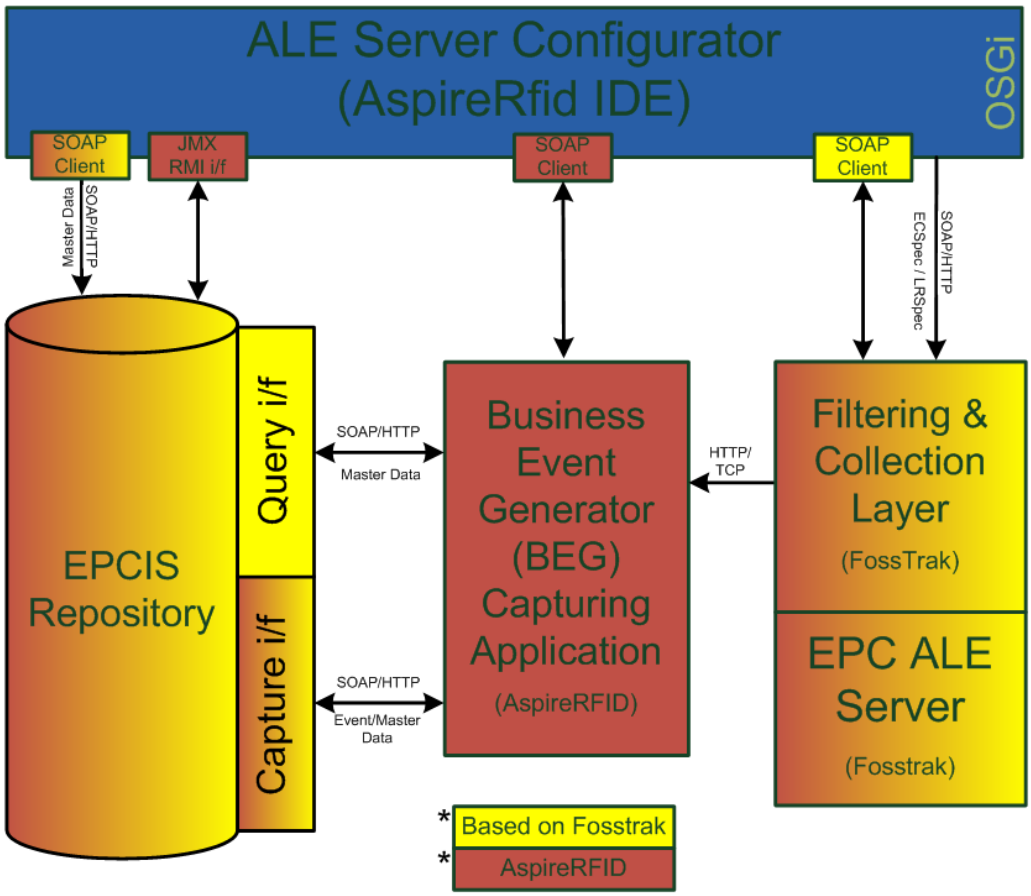

Figure 16 Creating Event data Sequence

7.2.1 Creating an Aggregation Event

To create an Aggregation Event the Business Event Generation module should receive an ECReport from the Filtering and Collection module comprised from three reports named:

- "*bizTransactionIDs_**"
- "*transactionItems_**"
- And "*parentObjects_**"

Where (*) the BusinessTransaction vocabulary URI.

7.2.1.1 Setting up the ECSpec

At the "*bizTransactionIDs*" report a filter should be set up in such a way that the F&C module would report, every time they are captured, only patterns of the included transaction ID Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.12.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="bizTransactionIDs_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
```

```
      <reportSet set="CURRENT" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.12.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 17 Example of a ReportSpec for the "*urn:epc:pat:gid-96:145.12." class**

At the "*transactionItems*" report a filter should be set up in a way that the F&C module would report, only the first time they are captured, only ID's belonging in the selected patterns of the included items Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.233.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="transactionItems_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="ADDITIONS" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.233.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 18 An example of ReportSpec for the "*urn:epc:pat:gid-96:145.233." class**

At the "*parentObjects*" report a filter should be set up in a way that the F&C module would report, every time they are captured, only patterns of the included parent Objects Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.56.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="parentObjects_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="CURRENT" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.56.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec>
```

```
            <pattern>
                  urn:epc:pat:gid-96:145.56.*
            </pattern>
      </groupSpec>
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 19 Example of a ReportSpec for the "urn:epc:pat:gid-96:145.56.*" class

7.2.1.2 Processing the ECReport

As soon as the report is received from the Business Event Generator module it is first checked whether a "*bizTransactionID*" is included or not. If it has then the specific one is used. If it has not then the last received one is used. Every "*transactionItem*" and "*parentObject*" is received and from now on it is bind with the specific "*bizTransactionID*".

After this it is checked if a "*parentObject*" is reported. If it has been reported, then it is used as the "*parentObject*" for every "*transactionItem*" received from now on. If it hasn't then the last received "*parentObject*" is used.

Finally, it is checked whether any "*transactionItems*" are reported. If they have then these "*transactionItems*" get as "*parentObject*" and "*bizTransactionID*" the last reported.

The rest of the information required to build the Aggregation Event is taken from the BusinessTransaction's attributes stored at the Information Services module repository.

7.2.2 Creating an Object Event

To create an Object Event the Business Event Generation module should receive an ECReport from the Filtering and Collection module comprised from two reports named:

- "*bizTransactionIDs_**"
- And "*transactionItems_**"

Where (*) the BusinessTransaction vocabulary URI.

7.2.2.1 Setting up the ECSpec

At the "*bizTransactionIDs*" report a filter should be set up in a way that the F&C module would report, every time they are captured, only patterns of the included transaction ID Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.12.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="bizTransactionIDs_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="CURRENT" />
      <filterSpec>
            <includePatterns>
```

```
                  <includePattern>
                        urn:epc:pat:gid-96:145.12.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 20 Example of a ReportSpec for the "urn:epc:pat:gid-96:145.12.*"

At the "*transactionItems*" report a filter should be set up in a way that the F&C module would report, only the first time they are captured, only ID's belonging in the selected patterns of the included items Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.233.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="transactionItems_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="ADDITIONS" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.233.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 21 Example of a ReportSpec for the "*urn:epc:pat:gid-96:145.233."class**

7.2.2.2 Processing the ECReport

As soon as the report is received from the Business Event Generator module it is first checked whether a "*bizTransactionID*" has been included or not. If the specific one has been found then it is used, otherwise the last one received is used. Every "*transactionItem*" received from now on is bind with the specific "*bizTransactionID*".

Finally, BEG checks whether any "*transactionItems*" are reported. If so, these "*transactionItems*" get as "*bizTransactionID*" the last reported.

The rest of the information required to build the Object Event is taken from the BusinessTransaction's attributes stored at the Information Services module repository.

7.2.3 Creating a Quantity Event

To create a Quantity Event the Business Event Generation module should receive an ECReport from the Filtering and Collection module comprised from two reports named:

- "*bizTransactionIDs_**"
- And "*transactionItems_**"

Where (*) the BusinessTransaction vocabulary URI.

7.2.3.1 Setting up the ECSpec

At the "*bizTransactionIDs*" report a filter should be set up in a way that the F&C module would report, every time they are captured, only patterns of the included transaction ID Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.12.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="bizTransactionIDs_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="CURRENT" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.12.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 22 Example of a ReportSpec for the "*urn:epc:pat:gid-96:145.12." class**

At the "*transactionItems*" report a filter should be set up in a way that the F&C module would report, only the first time they are captured, the count of the included patterns and in which Class they belong to. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.233.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="transactionItems_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="ADDITIONS" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.233.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec>
```

```
            <pattern>
                  urn:epc:pat:gid-96:145.233.*
            </pattern>
      </groupSpec>
      <output includeTag="false" includeRawHex="false"
            includeRawDecimal="false" includeEPC="false"
            includeCount="true" />
</reportSpec>
```

Figure 23 Example of a ReportSpec for the "*urn:epc:pat:gid-96:145.233." class**

7.2.3.2 Processing the ECReport

As soon as the report is received from the Business Event Generator module it is first checked whether a "*bizTransactionID*" has been included or not. If it has been included then the specific one is used, otherwise the last one received is used. Every "*transactionItem*" received from now on is bind to the specific "*bizTransactionID*".

Finally BEG checks whether any "*transactionItems*" are reported. If so, the count of these items and their Class get as "*bizTransactionID*" the last reported.

The rest of the information required to build the Quantity Event is taken from the BusinessTransaction's attributes stored at the Information Services module repository.

7.2.4 Creating an Transaction Event

To create Transaction Event the Business Event Generation module should receive an ECReport from the Filtering and Collection module comprised from three reports named:

- "*bizTransactionParentIDs_**"
- "*bizTransactionIDs_**"
- And "*transactionItems_**"

Where (*) the BusinessTransaction vocabulary URI.

7.2.4.1 Setting up the ECSpec

At the "*bizTransactionParentIDs*" report a filter should be set up in a way that the F&C module would report, every time they are captured, only patterns of the included transaction ID Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.19.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
      reportName="bizTransactionParentIDs_urn:epcglobal:fmcg:bte:xxxxxxxxx"
reportIfEmpty="true">
      <reportSet set="CURRENT" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
```

```
                        urn:epc:pat:gid-96:145.19.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 24 Example of a ReportSpec for the "*urn:epc:pat:gid-96:145.19." class**

At the "*bizTransactionIDs*" report a filter should be set up in a way that the F&C module would report, only the first time they are captured, only patterns of the included transaction ID Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.12.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="bizTransactionIDs_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="ADDITIONS" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.12.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 25 Example of a ReportSpec for the "*urn:epc:pat:gid-96:145.12." class**

At the "*transactionItems*" report a filter should be set up in a way that the F&C module would report, only the first time they are captured, only ID's belonging in the selected patterns of the included items Classes. An example of such a ReportSpec for the "*urn:epc:pat:gid-96:145.233.**" class is shown below.

```
<reportSpec reportOnlyOnChange="false"
reportName="transactionItems_urn:epcglobal:fmcg:bte:xxxxxxxxx"
      reportIfEmpty="true">
      <reportSet set="ADDITIONS" />
      <filterSpec>
            <includePatterns>
                  <includePattern>
                        urn:epc:pat:gid-96:145.233.*
                  </includePattern>
            </includePatterns>
            <excludePatterns />
      </filterSpec>
      <groupSpec />
      <output includeTag="true" includeRawHex="true"
      includeRawDecimal="true" includeEPC="true" includeCount="true" />
</reportSpec>
```

Figure 26 Example of a ReportSpec for the "*urn:epc:pat:gid-96:145.233." class**

7.2.4.2 Processing the ECReport

As soon as the report is received from the Business Event Generator module, it is first checked whether a "*bizTransactionParentID*" has been included or not. If it has been included then the specific one is used. If it hasn't then the last one received is used. Every "*bizTransactionID*" and "*transactionItem*" received from now on are bind with the specific "*bizTransactionParentID*".

Finally, it is checked whether any "*bizTransactionIDs*" and/or "*transactionItems*" are reported. If they are these "*bizTransactionIDs*" and/or "*transactionItems*" get as "*bizTransactionParentID*" the last reported.

The rest of the information required to build the Transaction Event is taken from the BusinessTransaction's attributes stored at the Information Services module repository.

Section 8 Available Tools for Defining Business Filters

8.1 Overview

As far as ease of development is concerned, the ASPIRE architecture specifies the existence of an IDE (Integrated Development Environment), which is conveniently called AspireRFID IDE (see Figure 27 below) enabling the visual management of all configuration files and meta-data that are required for the operation of an RFID solution.

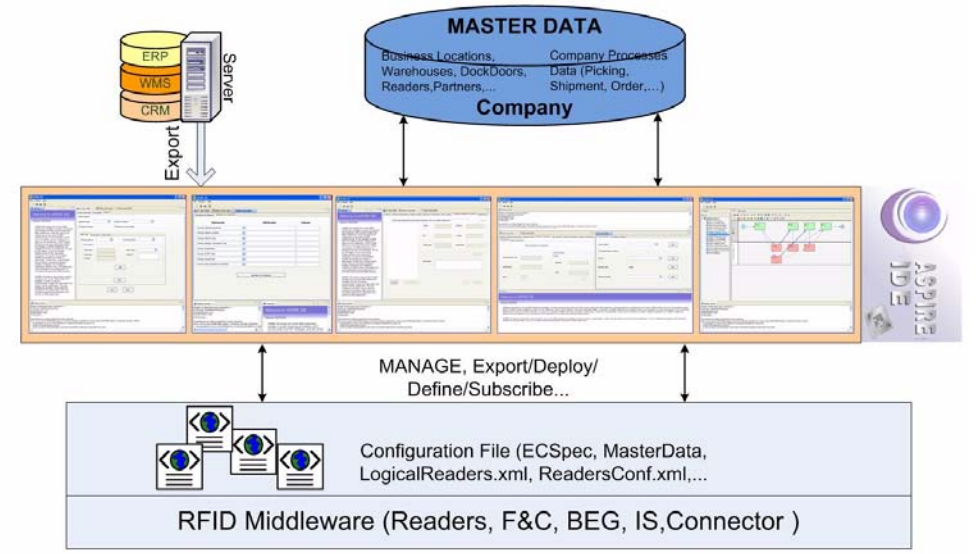

Figure 27 Programmability Tooling

AspireRFID IDE has been designed as an Eclipse RCP (Rich Client Platform) application that runs over Equinox OSGI server. It uses the command API (Application Program Interface) to define menus, pop-up menu items and toolbars so as to support easily plug-ins and provide more control. Every tool is an eclipse plug-in/bundle that is able to be installed or removed as needed. This way many editions of the AspireRFID IDE can be released depending on the functionalities required (as simple or as complicated depending on the demands) for the ASPIRE's RFID middleware blocks that will be used.

For specifying reprogrammable filters we use the ECSpec Editor and the Master Data Editor with the help of whom we are able to produce the required metadata to configure the Filtering and Collection module and the Business Event Generator module.

8.2 ECSpec Editor

ECSpec editor is a tool which is able to produce and edit EPC ALE V1.1 compliant ECSpec documents which are used to define the Filtering & Collection's module behavior concerning the kind of reports it will produce and which logical readers it will use to produce it. A sample view of the ECSpec editor is shown in Figure 28 below.

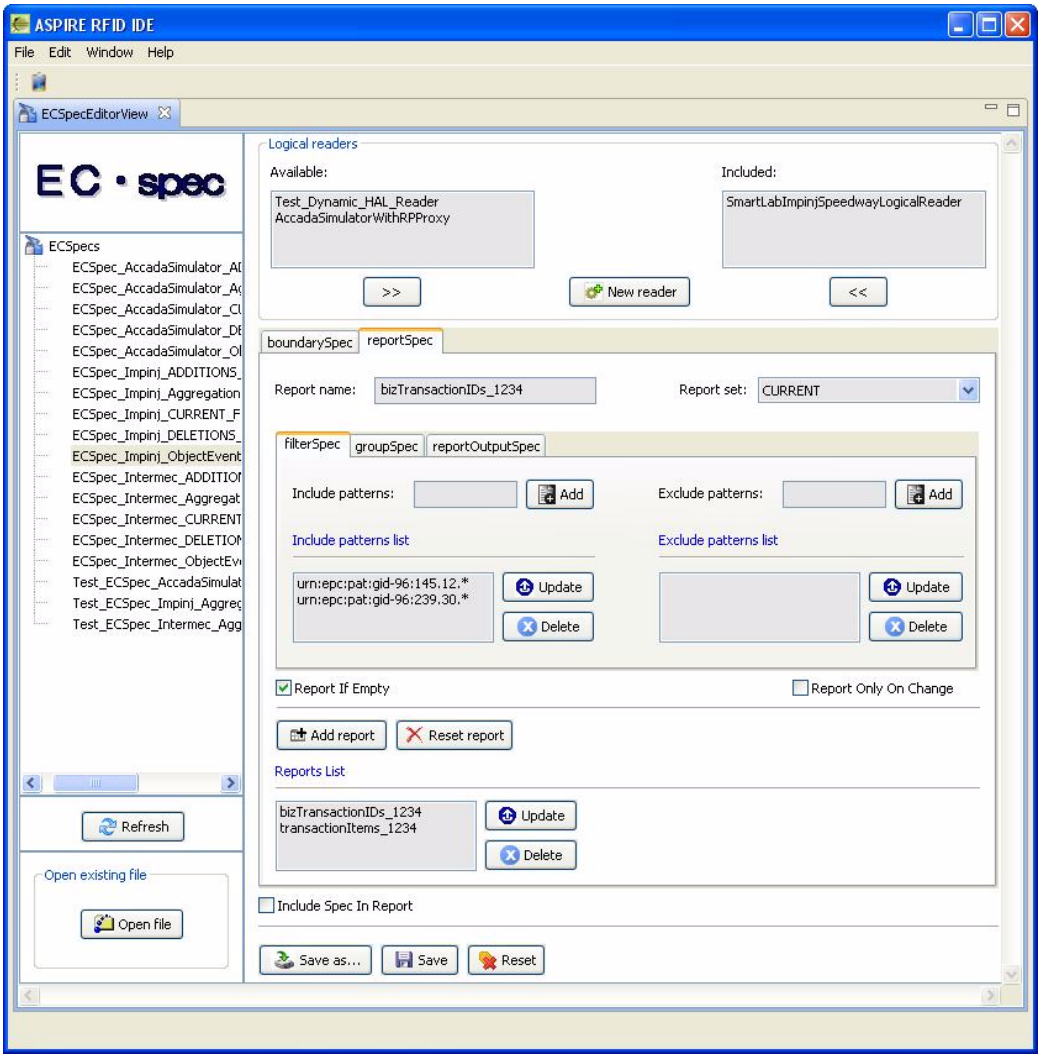

Figure 28 ECSpec Editor View

8.3 Master Data Editor (MDE)

The Master Data Editor (with support for Elementary Business Process Description) enables users and/or consultants to edit enterprise data (Master Data) including information about the company's location, its business locations, read points, as well as its business processes. The Master Data Editor (MDE) can be used for populating and managing vocabularies in the Master Data. Organizations can use this tool for the proper population of the Master Data without the need to know the associations of the Master Data vocabularies (Database Schema). All the relations among the RFID data and the restrictions are delegated to be the MDE's responsibility.

The MDE serves as an interface that is used to create the EPCIS repository master data and edit them as needed. Using the MDE users' can populate the EPCIS vocabularies and store all of the necessary information needed to provide business context to RFID data. The data are manipulated in an intuitive manner, and the user can view how they are related. Our MDE implementation uses the EPC EPCIS 1.0.1 specifications [6] query interface provided by Fosstrak [5] and

the AspireRFID Master Data capture interface [17] in order to access the EPCIS repository. Note that we have extended the Fosstrak project implementation, in a way that allows editing of the vocabularies which allows to the user to modify already existing master data. As already outlined MDE also complements BEG since MDE guides users to store all the data BEG needs in order to populate properly the fields of the various EPCIS events.

Using the MDE the user has the ability to define a process, and assign attributes that are descriptive to it and convey useful information about it. Additionally the user can define all of the events that compose the process. The events are treated as transactions and their EPC is stored in the business transaction vocabulary. In the attribute vocabulary for the transactions all of the related information for the events and the transactions is stored in the form of attribute-value pairs. The transactions are associated with the events through the attributes vocabulary. In Figure 29 we see that a user has created a process for supplying and has assigned certain information to it. This information (attributes) can be modified or removed as needed, or even new information can be added. As an example, in Figure 30, the user has associated an Object Event with this process and has provided all of the necessary information.

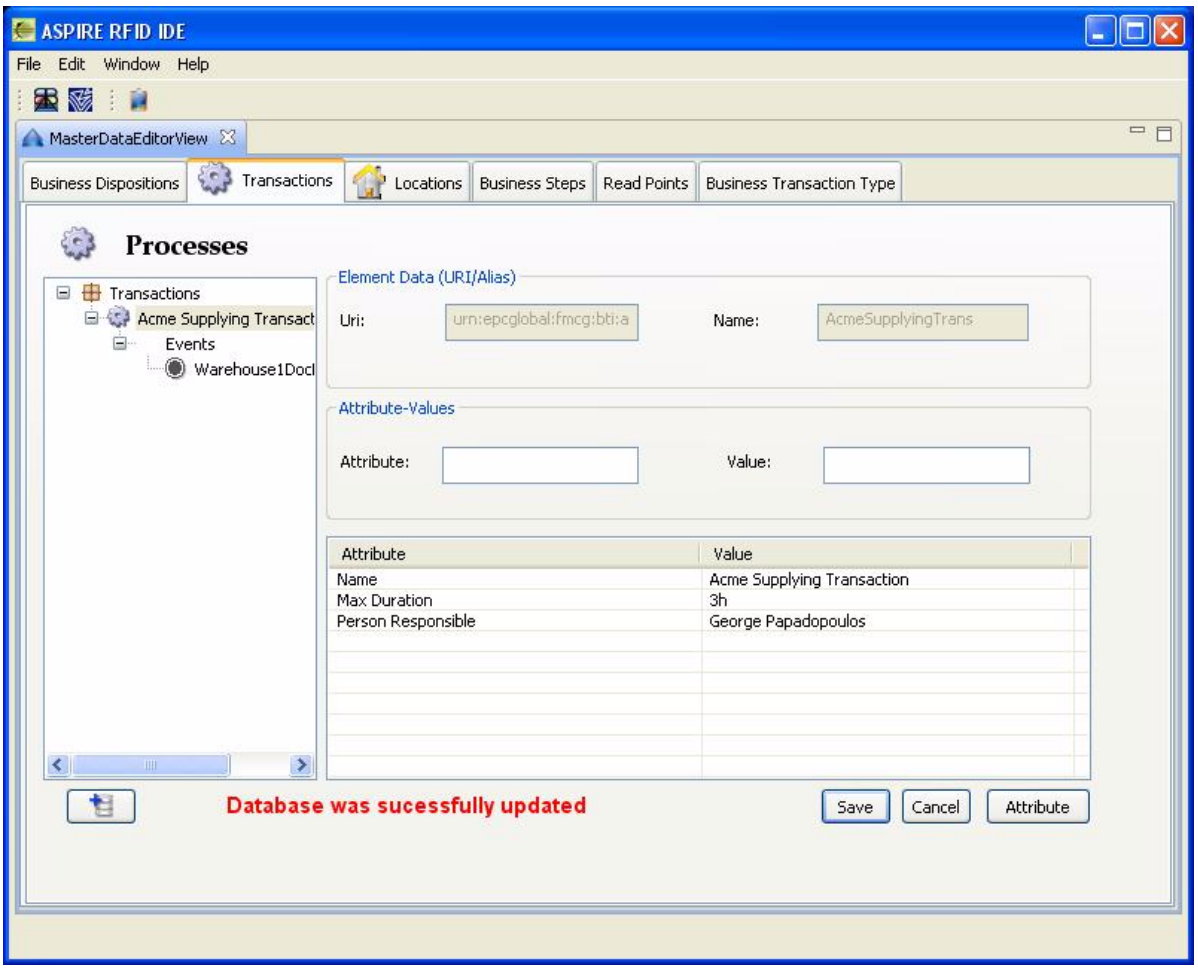

Figure 29 Definition of a transaction in the MDE

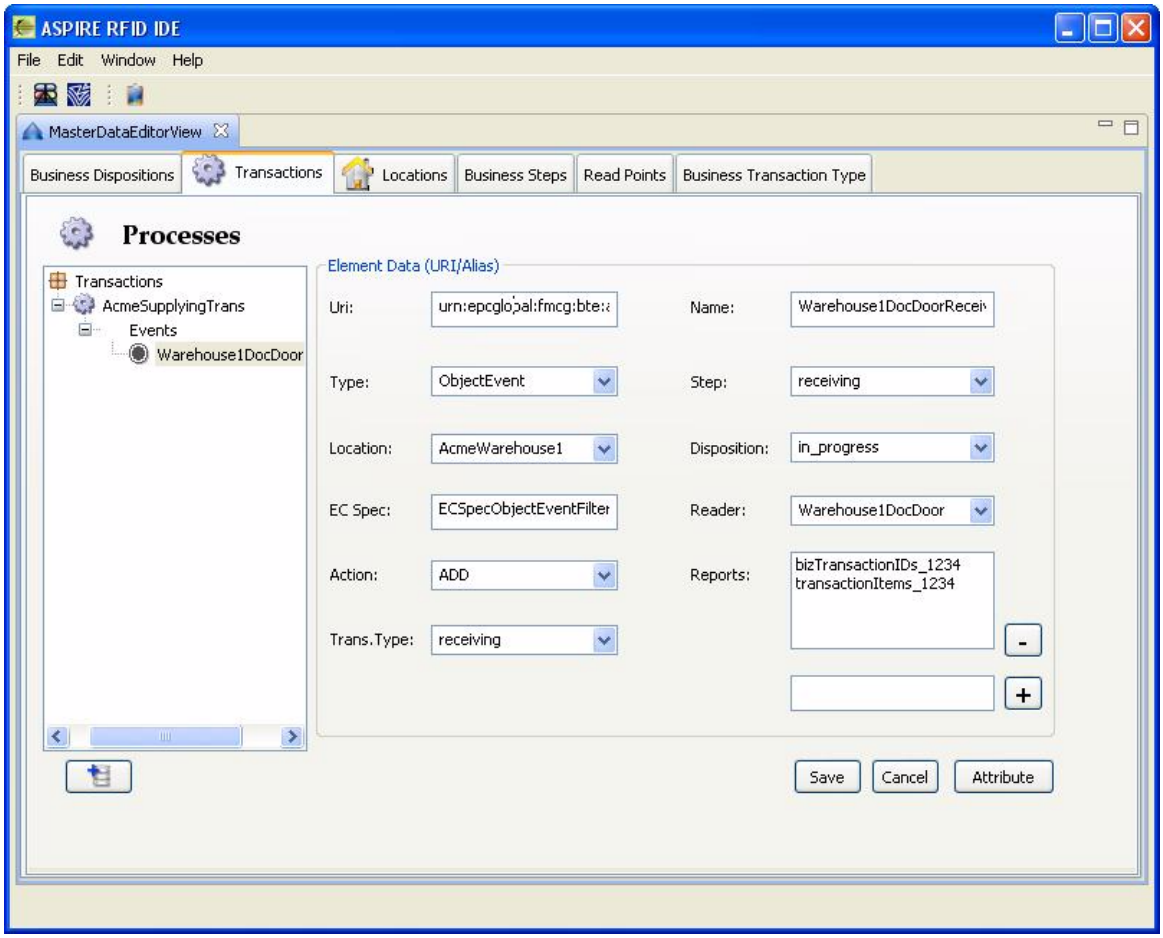

Figure 30 Process defined as a set of EPCIS events

A user has the ability to populate all the master data vocabularies using the corresponding tab, i.e. the Disposition vocabulary, the Business Step vocabulary, the Readers vocabulary, the Transaction Type vocabulary and the Business Location vocabulary. For all cases, the user can add a new element, associate attributes with it or edit an existing element, searching for it either among all of the existing elements or by the EPC. In the business location tag an organization can accurately describe all of the locations it owns in a hierarchical manner. For example has assuming that an organization has two warehouses, and two sections within the first warehouse, this can be reflected in the database and represented visually to the user which can add a location either as a single entity or as part of another location and associate attributes to it.

A new concept introduced in the MDE is the ability of the user to declare a location as no longer active. A user can mark a location as “deprecated” and this location and all of its sub locations will be defined as inactive as well in the database as we can see in Figure 31 below. The reason for such a feature is that as you have “described” your company to the system and work this way for some time a specific business location will be inevitably be bind with specific captured EPCIS Events. If later on the structure of the company enforce for a location to physically be removed, if we also delete it from the database, the

business location information of the already captured EPCIS events related with the specific location will be lost. So the notion of deprecation was created so as in spite the fact that the information is no longer used will not be lost. This is accomplished by adding an extra Boolean attribute at the business location, and this is allowed by the standard since it does not impose any restrictions on how to use the various attributes, which we call deprecated (urn:epcglobal:epcis:mda:deprecated). Also, a user can change the state and "un-deprecate" a location if necessary.[18]

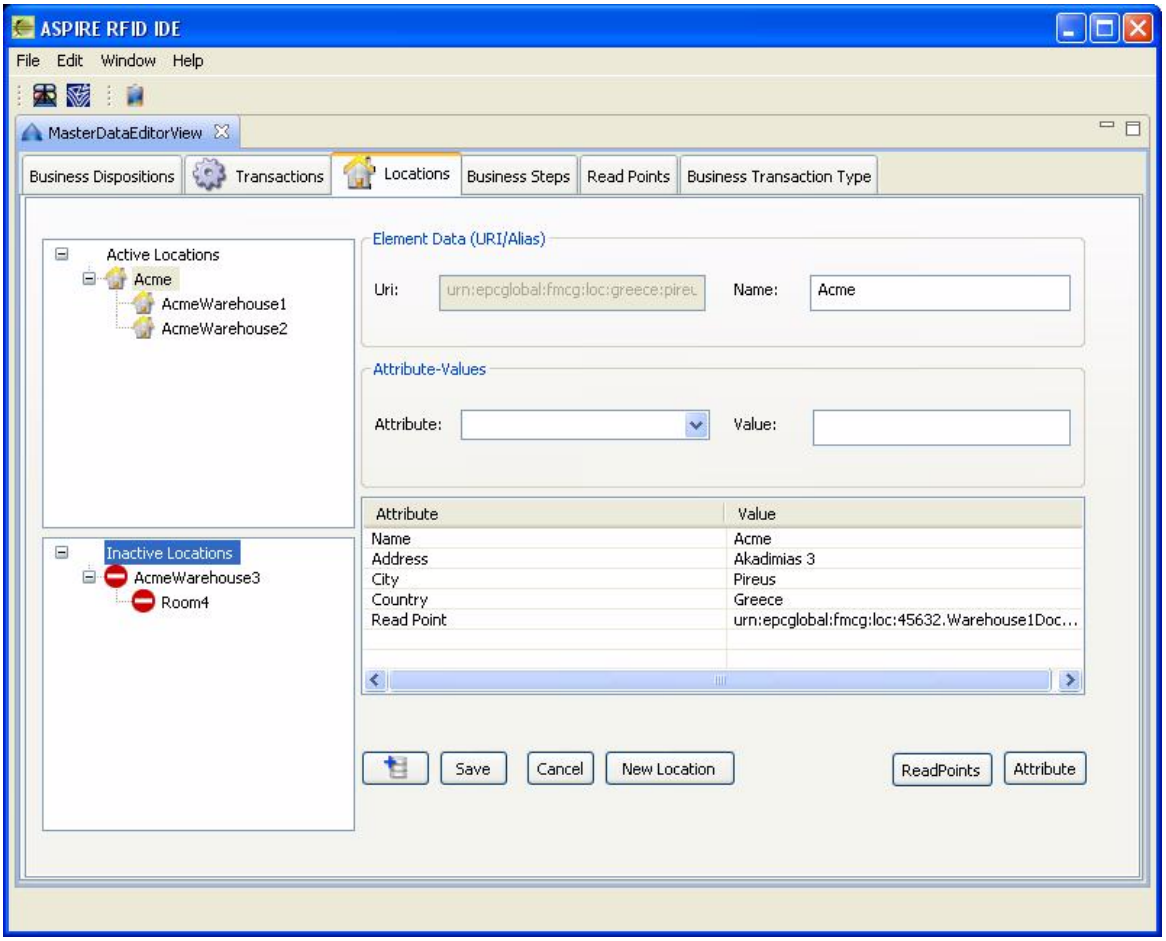

Figure 31 Master Data Editor (Business Location) - Company Acme owns 2 warehouses (Location AcmeWarehouse3 is no longer active)

Section 9 A complete example using Programmable filters

9.1 Describing the Problem

A Company Named "ACME", which is a Personal Computer Assembler, collaborates with a Microchip Manufacturer that provides it with the required CPUs (Central processing Units). 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 goods arrive.

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

9.2 Solution Requirements

An RFID Portal should be placed at 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 32 below) should be configured for the specific scenario.

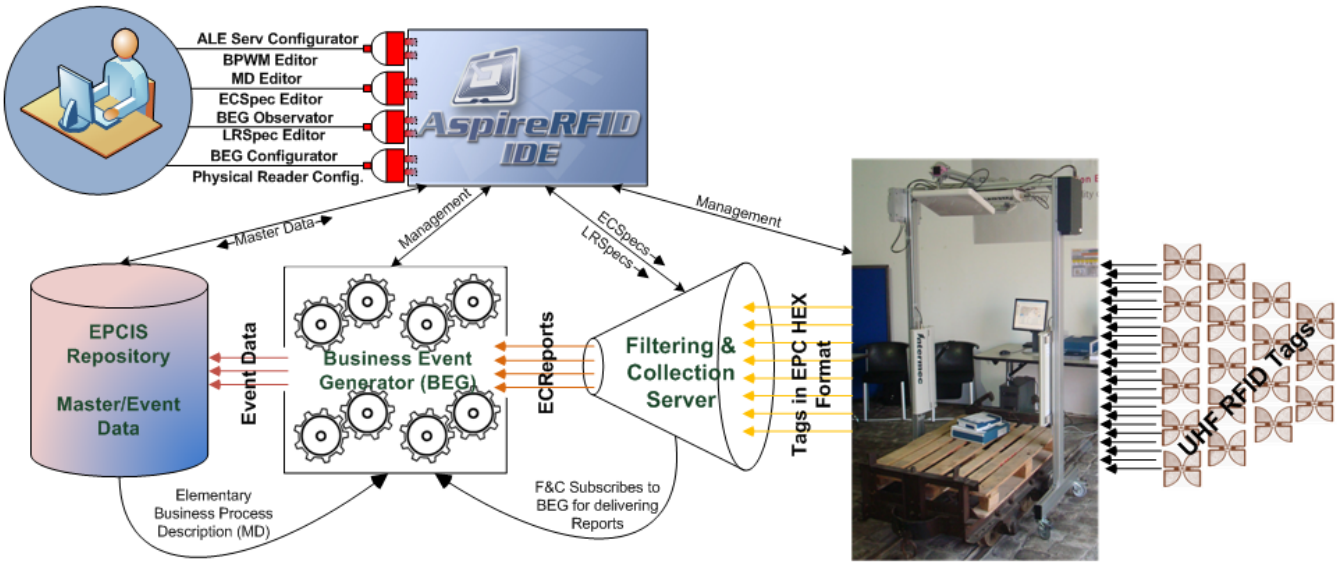

Figure 32 AspireRFID Architecture

9.3 Setting up the Filtering and collection Module

To Configure the Filtering and collection Module we should create an ECSpec 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 (Figure 5). 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" Classes are:
  - urn:epc:pat:gid-96:145.12.*
  - urn:epc:pat:gid-96:239.30.*

- and the "received items" Classes are:
  - urn:epc:pat:gid-96:145.233.*
  - urn:epc:pat:gid-96:1.3.*
  - urn:epc:pat:gid-96:1.4.*
  - urn:epc:pat:gid-96:145.255.*

By using the ECSpec editor as shown in Figure 33 below

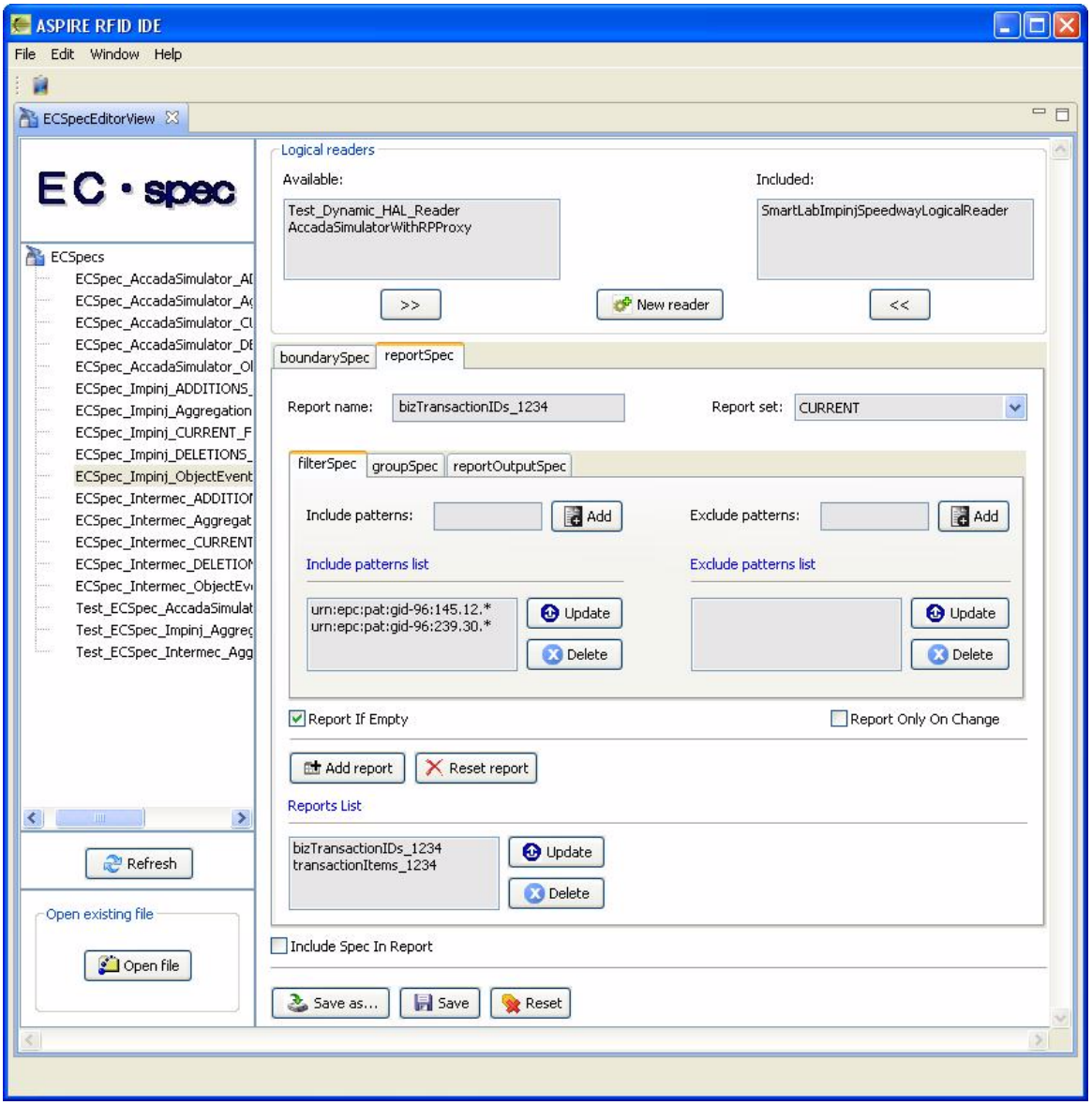

Figure 33 ECSpec Editor (Object Event)

We produce the ECSpec shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECSpec includeSpecInReports="false"
xmlns:ns2="urn:epcglobal:ale:xsd:1">
      <logicalReaders>
            <logicalReader>AccadaSimulatorWithRPProxy
            </logicalReader>
      </logicalReaders>
      <boundarySpec>
            <repeatPeriod unit="MS">4500</repeatPeriod>
            <duration unit="MS">4500</duration>
            <stableSetInterval unit="MS">0</stableSetInterval>
      </boundarySpec>
      <reportSpecs>
            <reportSpec reportOnlyOnChange="false"
reportName="urn:epcglobal:fmcg:bte:acmewarehouse1receive"
```

```
                  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>
            <reportSpec reportOnlyOnChange="false"
reportName="urn:epcglobal:fmcg:bte:acmewarehouse1receive"
                  reportIfEmpty="true">
                  <reportSet set="ADDITIONS" />
                  <filterSpec>
                        <includePatterns>
                              <includePattern>
                                    urn:epc:pat:gid-96:145.233.*
                              </includePattern>
                              <includePattern>
                                    urn:epc:pat:gid-96:1.3.*
                              </includePattern>
                              <includePattern>
                                    urn:epc:pat:gid-96:1.4.*
                              </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 />
</ns2:ECSpec>
```

Figure 34 Example of ECSpec

Which we will use to set up the Filtering & Collection module.

9.4 Setting up the Information Services Module

The Business Event Generator (Figure 5) 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 3 below from :

Business Transaction Attribute Name	Business Transaction Attribute Value
urn:epcglobal:epcis:mda:ecreport_names	bizTransactionIDs_ urn:epcglobal:fmcg:bte:acmewarehouse1receive, transactionItems_ urn:epcglobal:fmcg:bte:acmewarehouse1receive
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

Table 3 Master Data (Specifying a Transaction Event)

To set up the above info we will use the Master Data editor whose Business Transaction tab is shown in Figure 35 below.

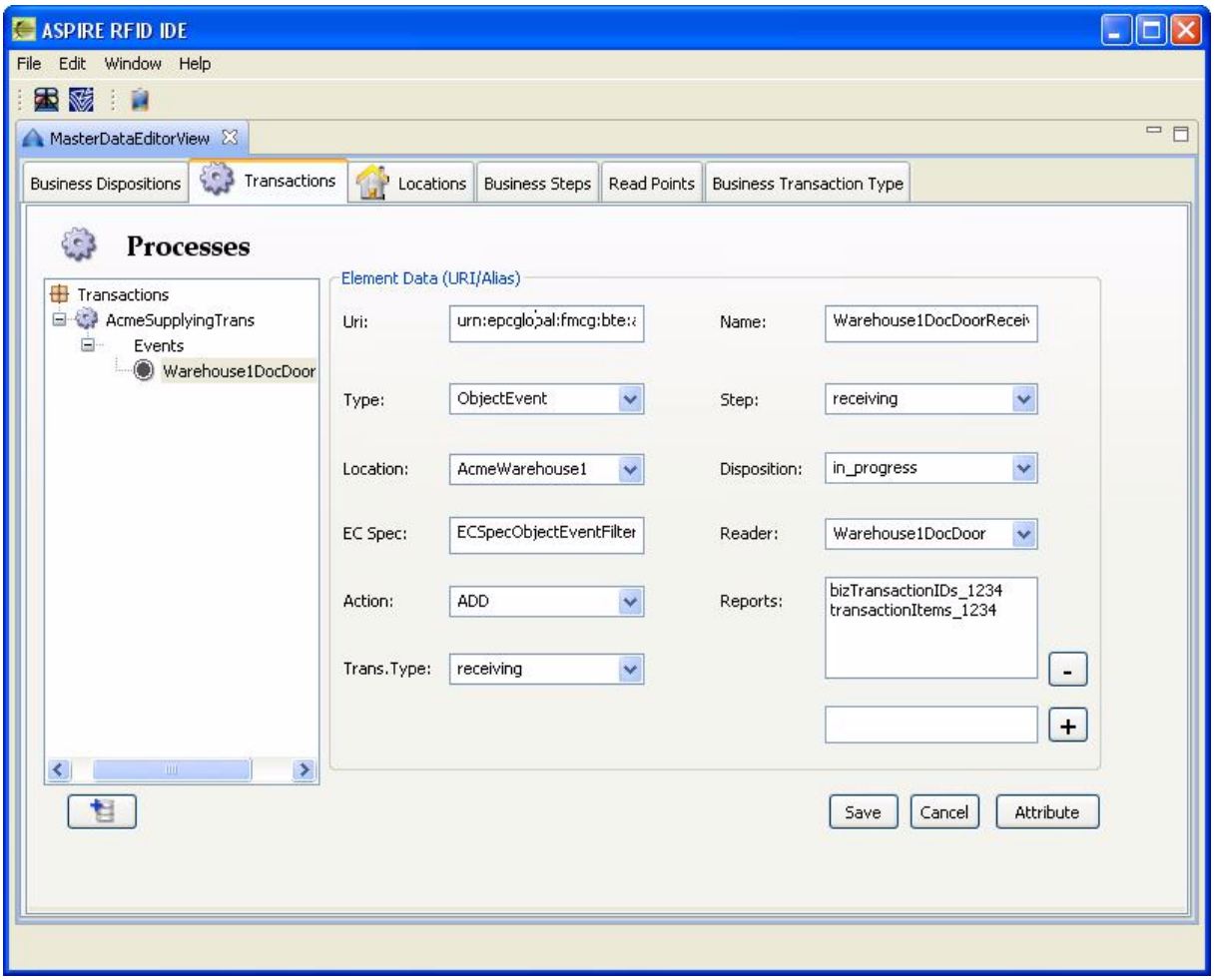

Figure 35 Master Data Editor (Specifying a Transaction Event)

9.5 Setting up the Business event generation module

The Business Event Generation module (Figure 5) should be set up to receive ECReports from the Filtering & Collection module (whose format is already defined from the ECSpec build above) and set it to serve the Transaction Event (urn:epcglobal:fmcg:bte:acmewarehouse1receive) defined with the master data editor illustrated above.

9.6 Process description

Figure 36 below depicts the whole process at a glance which is the following. 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.

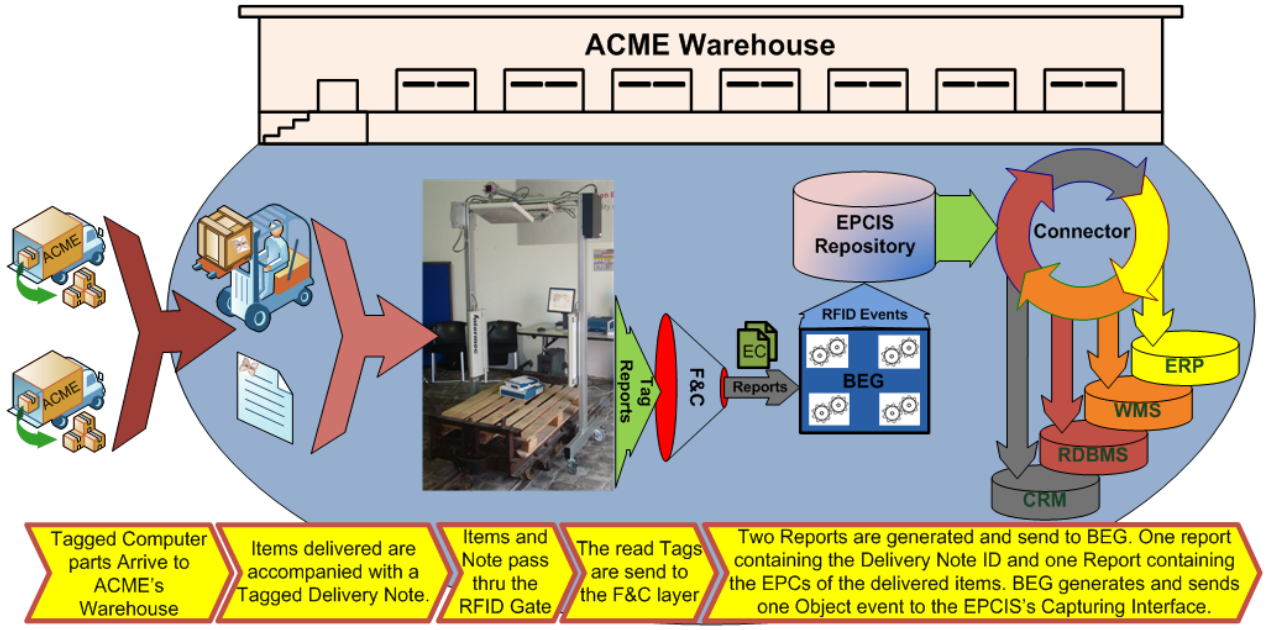

Figure 36 Acme computer parts Delivery

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.

Section 10 Filter using Distributed Hash Table (investigations)

In this section, we present a new way of RFID filtering based on the concept of DHT (distributed hash table), which is currently under investigations. Complete specifications of these filtering rules will be detailed in the deliverables of WP3.

This kind of filtering aims to offer a mechanism for querying only useful/concerned readers/databases. For example, let's consider the case of study of a distributor with several warehouses spread all over a country. Assume that warehouses are well organized (products of the same family are grouped, etc.), and that readers are spread over the warehouses.

With such a system, if we need for instance to draw an inventory of a special kind of products, we need to query every reader and/or database. Each of them needs to filter and aggregate data. We thus introduce a certain latency factor and load the network uselessly. For example, we ask to all nodes to scan all products but to report only trousers. Readers (or system) may perform a filter to count only trousers, no T-shirt, no sweat shirt, etc.

If, on the contrary, we assume that data are well organized and directed to a proper database, based on a DHT mechanism, (every reader/database is seen as a peer-to-peer node in a DHT based system), we can enhance performance.

The idea here is to assign one (or more) type to all nodes defining the kind of nodes they are (readers, databases, etc.), or/and the kind of products they are in charge. For example, databases near trousers in all warehouses are responsible for the type "trousers". When data concerning product kind A are read, they are directed towards databases responsible for A, based on a DHT direction. We can then group databases responsible for a special kind into a virtual layer, and allow querying only readers/databases responsible of a well defined type. Instead of asking to all nodes to filter products to report only trousers, we ask only to the ones that are responsible for trousers. Others are kept quiet. Note that a physical database may be responsible for several kinds of products and thus is mapped to several virtual layers. Figure 37 shows a projection of nodes by types (triangle, square or round).

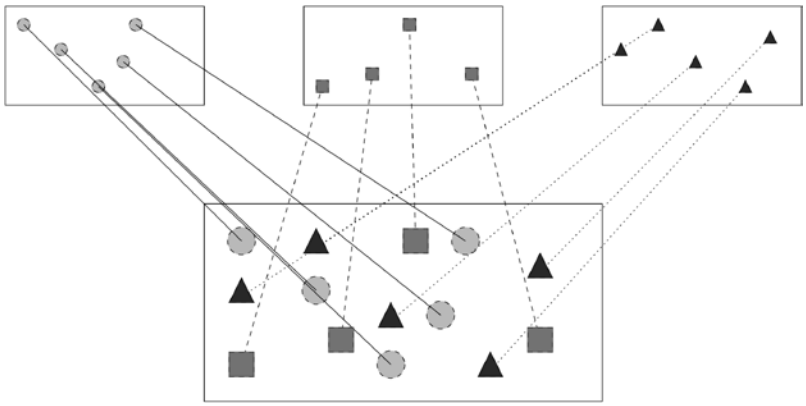

Figure 37 Projection into layers

It defines three layers composed by nodes of the same type. If the square nodes are readers/databases in different warehouses responsible for trousers for example, all we have to do is to interrogate the second layer (composed only with square nodes) with broadcast message to perform an inventory, anycast message to find at least one pair of trousers (if exists), or k-cast to know if the quantity is sufficient for a command.

Section 11 Conclusions

The generation of business events is an essential capability of an RFID middleware platform so as to be able to rapidly develop and deploy end user applications. This type of events is a combination of filtered but raw tag readings with added-on business-related information. This process can transform meaningless reading data to powerful and valuable information for intelligent applications and services in the upper layers.

As it has been extensively presented in this deliverable, the role of programmable filters is essential for providing a stable but customizable platform to build more complex services based on business events. In this direction, we defined in this deliverable the core element of this platform through the definition of the Programmable Filters Specification, its components and examples using the associated filtering mark-up language (FML). Moreover we have provided the proof of concept by implementing these components as part of the AspireRFID middleware which exists in the scope of the Aspire project.

The filtering markup language not only will help in the programmability of the tool but it will also provide modularity and the possibility of reusing filtering templates. In this way future developers can start building up new and interesting filtering policies from previously tested and mature solutions. This deliverable has presented several examples of how to use the reusable filters, the contents of their specification and the events currently supported. Additionally ASPIRE offers an Integrated Development Environment that allows easy development and definition of the filtering rules. Finally, research studies in using DHTs in order to enhance the filtering functionalities of an RFID system have also been provided.

This document is the final version of the deliverable, thus being the updated version of the specifications that were released by month 14 (M14) of the project.

Section 12 List of Acronyms

AAU	Aalborg University
AIT	Athens Information Technology
ALE	Application Level Event
API	Application Product Interface
ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generator
CTIF	Center for TelelnFranstrutur
DHT	Distributed Hash Table
DoW	Description of Work
EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Services
ERP	Enterprise Resource Planning
F&C	Filtering and Collection
FML	Filter Markup Language
FP	Framework Project
HAL	Hardware Abstraction Layer
HF	High Frequency
HTTP	HiperText Transfer Protocol
ICT	International Conference on Telecommunications
ID	Identification
IDE	Integrated Development Environment
INRIA	Institut National de Recherche en Informatique et en Automatique
IT	Information Technology
IT	Instituto de Telecomunicações
iPOJO	injected POJO
JMX	Java Management Extensions
LLRP	Low Level Reader Protocol
MDE	Master Data Editor
OBR	OSGi Bundle Repository
OSGI	Open Service Gateway Initiative
OSI	Open System Interconnection
OSS	Open Source Software
OW2	
PM	Person month
POJO	Plain Old Java Object
PU	Public
RF	Radio Frequency
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
SVN	Subversion
TCO	Total Cost of Ownership

TCP	Transfer Control Protocol
UHF	Ultra High Frequency
UML	Universal Markup Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WADL	Wired Application Description Language
WMS	Warehouse Management System
WP	Work Package
XML	Extensible Markup Language

Section 13 List of Figures

Section 14 List of Tables

Section 15 References and Bibliography

[1] Matthias Lampe, Christian Floerkemeier, “High-Level System Support for Automatic-Identification Applications”, In: Wolfgang Maass, Detlef Schoder, Florian Stahl, Kai Fischbach (Eds.): Proceedings of Workshop on Design of Smart Products, pp. 55-64, Furtwangen, Germany, March 2007.

[2] BEA WebLogic. Understanding the Event, Master Data, and Data Exchange Services. BEA WebLogic RFID Entersprise Server. [Online] October 12, 2006. http://e-docs.bea.com/rfid/enterprise_server/docs20/pdf.html.

[3] EPCglobal, “The Application Level Events (ALE) Specification, Version 1.1”, February. 2008, available at: http://www.epcglobalinc.org/standards/ale

[4] EPCglobal Inc™. Frequently Asked Questions - ALE 1.1. EPCglobal. [Online] http://www.epcglobalinc.org/standards/ale.

[5] FossTrak Project. FossTrak Project. [Online] http://www.fosstrak.org/index.html.

[6] EPC Information Services (EPCIS) Specification, Version 1.0.1, September 21, 2007 available at: http://www.epcglobalinc.org/standards/epcis/

[7] EPCglobal Inc™. The EPCglobal Architecture Framework Version 1.2. [Online] September 10, 2007. http://www.epcglobalinc.org/standards/architecture/.

[8] Application Level Events 1.1(ALE 1.1) Overview, Filtering & Collection WG, EPCglobal, March 5, 2008 , available at: http://www.epcglobalinc.org/standards/ale

[9] C.Floerkemeier, C. Roduner, and M. Lampe, RFID Application Development With the Accada Middleware Platform, IEEE Systems Journal, Vol. 1, No. 2, December 2007.

[10] C. Floerkemeier and S. Sarma, “An Overview of RFID System Interfaces and Reader Protocols”, 2008 IEEE International Conference on RFID, The Venetian, Las Vegas, Nevada, USA, April 16-17, 2008.

[11] Russell Scherwin and Jake Freivald, Reusable Adapters: The Foundation of Service-Oriented Architecture, 2005.

[12] 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".

[13] Architecture Review Committee, “The EPCglobal Architecture Framework,” EPCglobal, July 2005, available at: http://www.epcglobalinc.org.

[14] 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).

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

[16] 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.
[17] John Soldatos, Nikos Kefalakis, Nektarios Leontiadis, et. al., "Core ASPIRE Middleware Infrastructure", ASPIRE Project Public Deliverable D3.4a, Jun 2009, publicly available at: http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables
[18] Efstathios Mertikas, Nikos Kefalakis and John Soldatos, "Managing Master Data and Business Events in an RFID Network", Submitted to the Pervasive and Mobile Computing Journal (Elsevier), September 2009
[19] Jeremy Landt, The history of RFID, IEEE Potentials, October-November 2005.
[20] Ron Weinstein, RFID: A Technical Overview and Its application to the enterprise. IT Pro, published by the IT Computer Society.
[21] ASPIRE D2.3b. Architecture specifications for the innovation management framework, 2009.
[22] ASPIRE D2.2 (End-User) SME Requirements (IT infrastructure, traceability, Anti-counterfeiting, privacy)
[23] ASPIRE D2.1 Review on methods and tools for concurrent innovation engineering - Review of State-of-the-Art Middleware, Methods, Tools and Techniques
[24] ASPIRE, Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications, Annex I Description of work.
[25] ASPIRE D3.3 Data Collection, Filtering and Application Level Events
[26] ASPIRE D4.3a Programmable filters (FML) specification. Interim version