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 (Interim Version)

Due date of deliverable:	M15
Actual Submission date:	

Deliverable ID:	**WP4/D4.3a**
Deliverable Title:	Programmable Filters – FML Specification (Interim Version)
Responsible partner:	AAU
Contributors:	John Soldatos (AIT) Nikos Kefalakis (AIT) Nektarios Leontiadis (AIT) Lo¨c Schmidt, Nathalie Mitton (INRIA)
Estimated Indicative Person Months:	14

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

Revision:	0.1
Dissemination Level:	PU

Document Information

Document Name:	**Programmable Filters – FML Specification (Interim Version)**
Document ID:	**WP4/D4.3a**
Revision:	**0.1**
Revision Date:	**06 March 2009**
Author:	**AAU**
Security:	**PU**

Approvals

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

Reviewers

Name	Organization	Date	Comments	Visa
Nathalie Mitton	INRIA			

Document history

Revision	Date	Modification	Authors
0.1	20 Mar 09	First draft	Nikos Kefalakis
0.2	23 Mar 09	Edited Programmable Filters Specification (creating business logic)	Nikos Kefalakis
0.3	24 Mar 09	Augmented Executive Summary and Introduction	Ramiro Samano Robles
0.4	26 Mar 09	Added Available Tools and the Complete Example	Nikos Kefalakis
0.5	01 Apr 09	Conclusion, augmented Introduction	Nektarios Leontiadis, John Soldatos
0.6	14 Apr 09	Section 3,Section 4, augmented Section 2	Nikos Kefalakis, John Soldatos
0.7	15 Apr 09	Added Section 10	Loïc Schmidt, Nathalie Mitton

0.8	29 Apr 09	Minor Corrections	Nikos Kefalakis

Content

Section 1 Executive Summary

This deliverable provides the specification of two of the main characteristics and contributions of the ASPIRE middleware platform: the programmable filters and their associated filtering markup language (FML). The filtering functionality within an RFID (Radio Frequency Identification) system plays an important role in the architecture since it is used to select and refine the raw tag data collected by the interrogator(s) which contains relevant information to upper layer applications while rejecting those unwanted or noisy data. The filtering and collection (F&C) module is, therefore, an important interface between the RFID interrogator or reader and the business world.

Parts of this deliverable provide explicit examples on how to use the widely accepted extensible markup language concept (XML), which is borrowed from the Internet literature, in order to illustrate how to program filters with business meaning in the context of the ASPIRE RFID middleware platform. By using such a flexible and high-level markup language approach, the envisioned filters can easily cope with business events as well as being easy to understand both for software integrators and for the engine that interprets its semantics. Furthermore, the filters can be easily reused for other applications, thus providing the ASPIRE platform with a great deal of programmability and modularity. Future integrators can make use of existing filters and templates in order to generate improved or application specific filtering functionalities.

We recall here that ASPIRE is developing an innovative royalty free middleware platform. This middleware platform is a primary target of the open source software (OSS) "AspireRFID" project, which has been successfully established in the scope of the OW2 community (http://wiki.aspire.ow2.org/). The programmable nature of the "AspireRFID" project asks for versatility and reusability in terms of the filtering that will be supported. Such a reusability and programmability concept is fully exploited by the proposed filtering markup language, which is easy to understand for anyone with a basic training on markup languages for Internet applications.

At the end of this deliverable the reader will be able to understand the semantics and philosophy of the filtering markup language and to relate them to the main components of the ASPIRE architecture, namely the Application Level Event generator (ALE), the Business Event Generator (BEG) and the Information Sharing Repository (EPCIS). Details are given later in this document, while the definition of the ASPIRE architecture for reference purposes can be found in previous documents of the project such as in D2.3b. Also note that this document is the interim version of the deliverable, thus being the preamble to the final version of the specifications to be released by month 24 (M24) of the project.

Section 2 Introduction

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.

Filtering has therefore 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

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), 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 workpackage 3, e.g. D3.3)

The filtering functionality in and 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). 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 filter 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.

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 to understand the operation and structure of the language format and the functionalities addressed in the filtering definition. Also note that this document is the interim version of the deliverable, thus being the preamble to the final version of the specifications to be released by month 24 (M24) of the project.

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 in the Aspire middleware architecture.
- Section 4 discusses the concept and the operation of the programmable filters from a high level of 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 will be completed in Deliverable 4.3b and
- Section 11 draws a conclusion 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 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 a problem like the one described above is the concept of reusable filters. By setting specific filters for the RFID middleware using XML 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 RFID developer expert and to set it up to serve the specific company without much effort.

The above solution concept derives from the ability to break off into distinct business transactions a company's business processes which at its turn can be break up in distinct Transaction Events as shown in Figure 1 below.

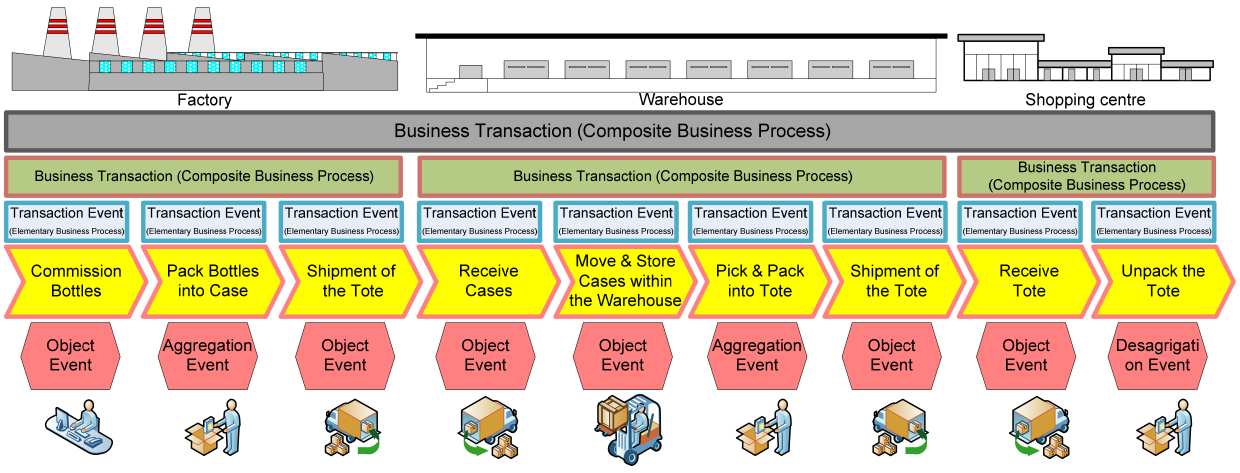


Figure 1 Wider Business Process/Transactions Example

The fact is that it is important to break 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.

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.

Figure 2 depicts the concept of decomposing a process into a number of business events. The latter events comply to the ASPIRE Information Sharing specifications for RFID events (with direct references to EPC-IS framework). We call Elementary Business Process, the process which can be directly decomposed into RFID business events (as shown in Figure 2).

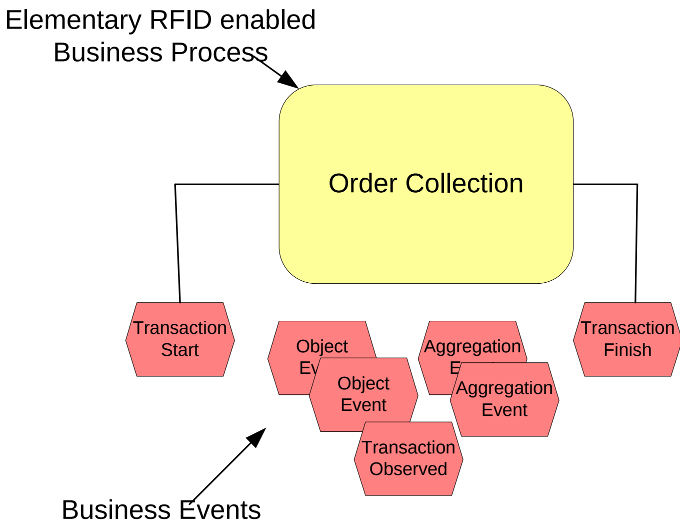


Figure 2 Description of Elementary RFID enabled Business Process

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.

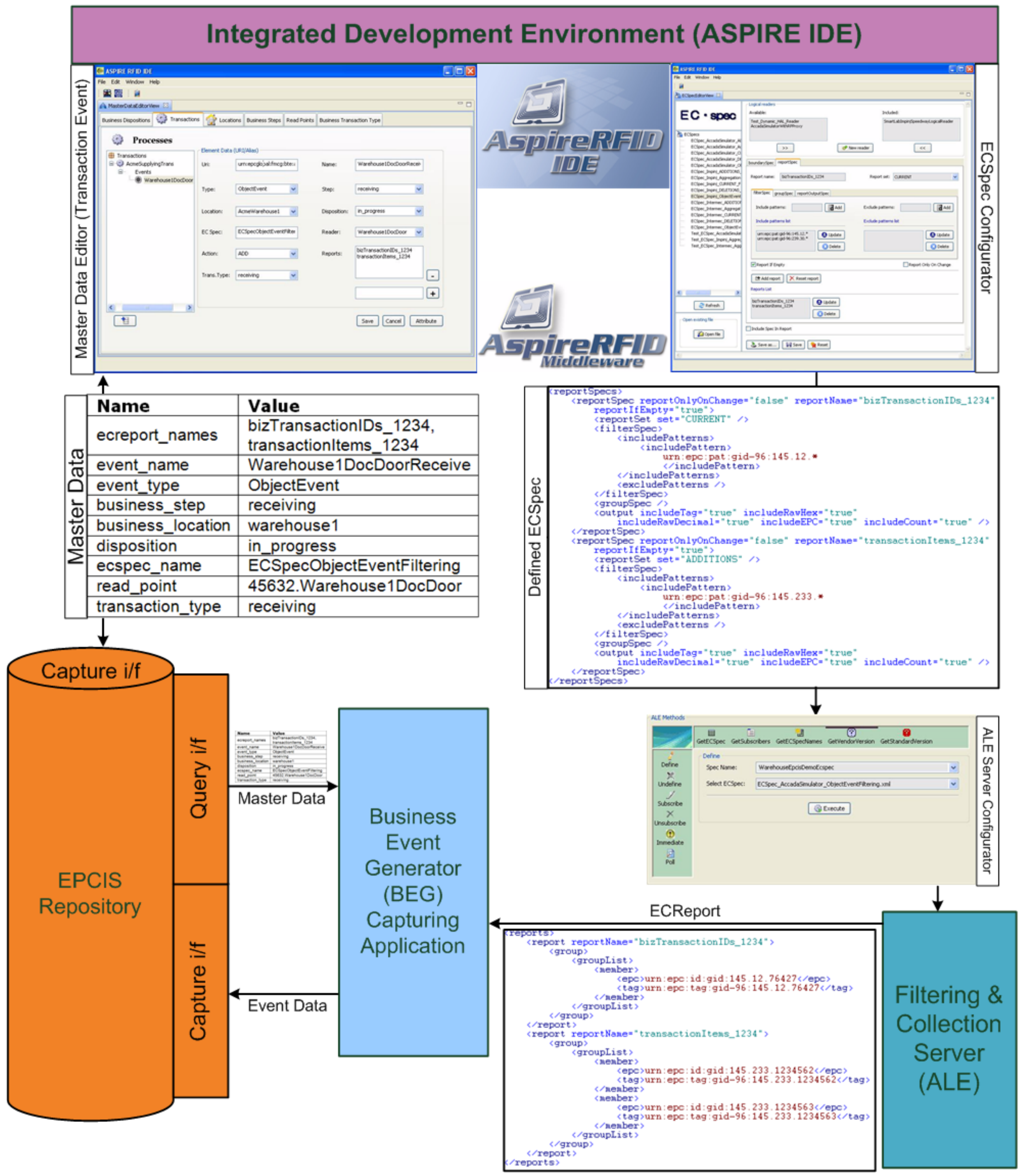
Section 4 Notion of Programmable Filters in the ASPIRE Architecture

Because ASPIRE is designed in such a way that it will be expandable, configurable and modular, the use of Programmable filters is necessary. Moreover ASPIRE IDE will include a programmability engine, which will be an integral component of the AspireRFID IDE. This engine will be able to process a fully fledged RFID solution described in a special purpose domain specific language which will be specified as part of future deliverables of the WP4 of the ASPIRE project.

The core "engine" described in Section 3 above for the AspireRFID middleware is comprised from three different components:

- The filtering and Collection module, which is responsible of the low level filtering.
- The Business Event Generator module, which is responsible for the High Level filtering providing Buisness context to the captured events.
- And the Information sharing repository, which is the repository which stores a company's Master Data and Business functions.

Figure 3 demonstrates a complete Aspire programmable filter solution with ASPIRE's existing tools. With the help of Master Data editor we can "describe" the company's business data, processes and the required Low Level input to create business events. All these are stored to the Information Sharing repository which is used from the Business Event Generator on demand to configure its behavior on creating the business events. With the help of ECSpec editor we can create the required ECSpecs that configure the Filtering and Collections layer behavior by using the ECSpec configurator. The Filtering and Collection module after being configured as required collects the raw readings from the attached RFID readers to it and produces the filtered ECReports which are feed to the Business Event Generator. A complete detailed example is described in Section 9.



Name	Value
ecreport_names	bizTransactionIDs_1234, transactionItems_1234
event_name	Warehouse1DocDoorReceive
event_type	ObjectEvent
business_step	receiving
business_location	warehouse1
disposition	in_progress
ecspec_name	ECSpecObjectEventFiltering
read_point	45632.Warehouse1DocDoor
transaction_type	receiving

```
<reportSpecs>
    <reportSpec reportOnlyOnChange="false" reportName="bizTransactionIDs_1234"
        reportIfEmpty="true">
        <reportSet set="CURRENT" />
        <filterSpec>
            <includePatterns>
                <includePattern>
                    urn:epc:pat:gid-96:145.12.*
                    </includePattern>
            </includePatterns>
            <excludePatterns />
        </filterSpec>
        <groupSpec />
        <output includeTag="true" includeRawHex="true"
            includeRawDecimal="true" includeEPC="true" includeCount="true" />
    </reportSpec>
    <reportSpec reportOnlyOnChange="false" reportName="transactionItems_1234"
        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>
</reportSpecs>
```

```
<reports>
    <report reportName="bizTransactionIDs_1234">
        <group>
            <groupList>
                <member>
                    <epc>urn:epc:id:gid:145.12.76427</epc>
                    <tag>urn:epc:tag:gid-96:145.12.76427</tag>
                </member>
            </groupList>
        </group>
    </report>
    <report reportName="transactionItems_1234">
        <group>
            <groupList>
                <member>
                    <epc>urn:epc:id:gid:145.233.1234562</epc>
                    <tag>urn:epc:tag:gid-96:145.233.1234562</tag>
                </member>
                <member>
                    <epc>urn:epc:id:gid:145.233.1234563</epc>
                    <tag>urn:epc:tag:gid-96:145.233.1234563</tag>
                </member>
            </groupList>
        </group>
    </report>
</reports>
```

Figure 3 Complete Programmable Filters ASPIRE solution

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

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 a logical reader name. It is converted into a business location in the Information Services module.

Further, 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…) that manipulate Information Services module data, in contrast, typically deal explicitly with historical data and hence are inherently persistent in nature.

Architecturally, the Filtering and Collection layer is concerned with dealing with the mechanics of data gathering, and of filtering down to meaningful events that are a suitable starting point for interpretation by business logic. Business layers, where 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 the receipt of 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, 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 is therefore a smaller part of the whole. [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 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 we need first to analyze the components that are used from each one separately.

Note that in this section we are going to define the specifications for only the required parts from the all ASPIRE's architecture.

6.2 Filtering and Collection module

6.2.1 Role

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.

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:

- 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 completes, 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. 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.

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-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]

6.3 Information Services module

6.3.1 *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.

6.3.1.1 EPCIS Event

An EPCISEvent is a generic base class for all event types which provides date and time fields. Below is given the EPCISEvent's XML schema. [2][6]

```
<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>
```

6.3.1.2 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.

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>
```

6.3.1.3 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>
```

6.3.1.4 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>
```

6.3.1.5 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>
```

6.3.2 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:

- **Add** which means that the entity in question has been created or added to.
- **Observe** which means that the entity in question has not been changed it neither has been created, added to, destroyed, or removed from.
- **Delete** which means that the entity in question has been removed from or destroyed altogether.

6.3.3 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.

6.3.3.1 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:

- bizLocation
- bizStep
- bizTransactionList
- bizTransaction which is comprised from the
    - Business Transaction and
    - Business Transaction Type
- disposition
- epcClass
- readPoint

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

6.3.3.1.1 BusinessTransaction

A BusinessTransaction 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 a pair of identifiers, as follows:

- 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 Business event generation module associates business-context information (Master Data) with event data. The data is stored in the Information Services module repository as Event Data and are mapping associated events with a company's master data.

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

The business event generation module uses the data defined at the BusinessTransaction's Attributes to translate the received ECReports from the Filtering and Collection module.

Section 7 Programmable Filters Specification (creating business logic)

7.1 Overview

The "glue" from the three modules described in Section 6 to create business logic at the AspireRFID middleware is the Business Event Generation module, which uses the predefined data provided from the two others to produce the required Event Data.

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 4 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 its turn process 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 from the F&C module and process 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.

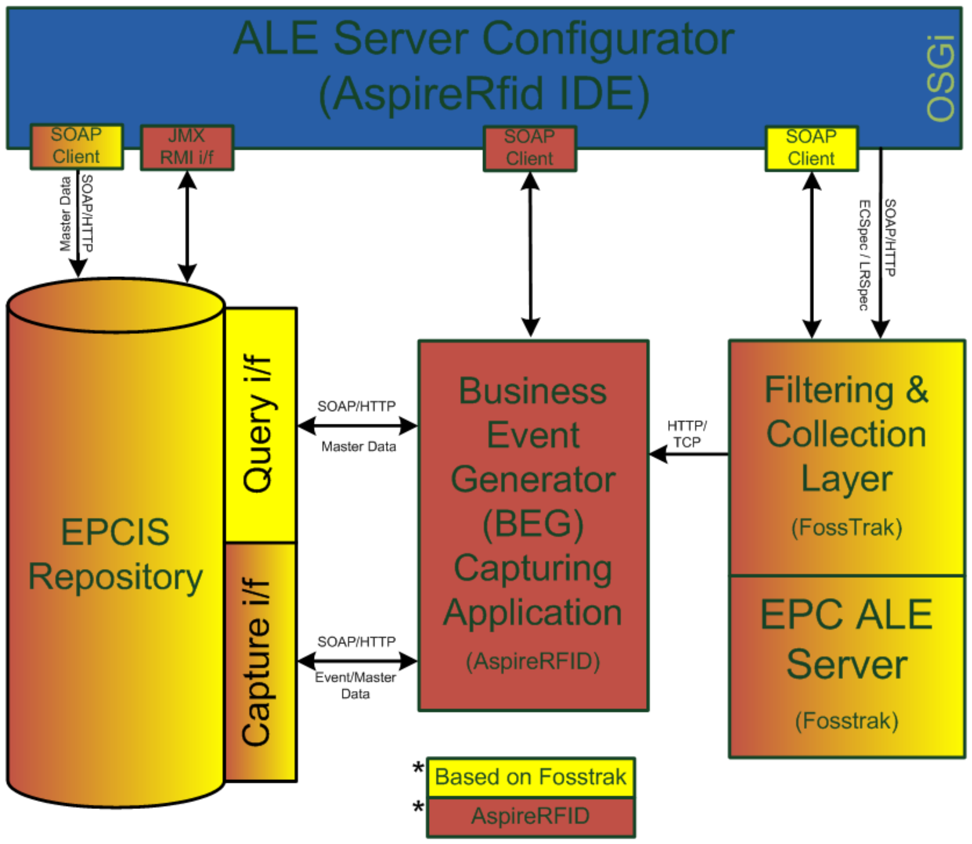


Figure 4 Creating Event data Sequence

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 ID of each report should be placed, taking in consideration the list of "*ECReportNames*" Attribute of the BusinessTransaction Master Data type.

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_456"
```

```
      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>
```

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_456"
      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>
```

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_456"
      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>
```

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. 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, the 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 ID of each report should be placed, taking in consideration the list of “*ECReportNames”* Attribute of the BusinessTransaction Master Data type.

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_768"
      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>
```

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_768"
      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>
```

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. If the specific one has been used then it is used. If it hasn't then 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 ID of each report should be placed, taking in consideration the list of "*ECReportNames*" Attribute of the BusinessTransaction Master Data type.

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_1234"
      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>
```

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_1234"
      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>
```

7.2.3.2 Processing the ECReport

As soon as the report is received from the Business Event Generator module first it is checked whether a "*bizTransactionID*" has been included. If it has been included then the specific one is used. If it hasn't then 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 ID of each report should be placed, taking in consideration the list of "*ECReportNames*" Attribute of the BusinessTransaction Master Data type.

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_736" 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>
```

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_739"
      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>
```

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_739"
      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>
```

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 5 below) enabling the visual management of all configuration files and meta-data that are required for the operation of an RFID solution.

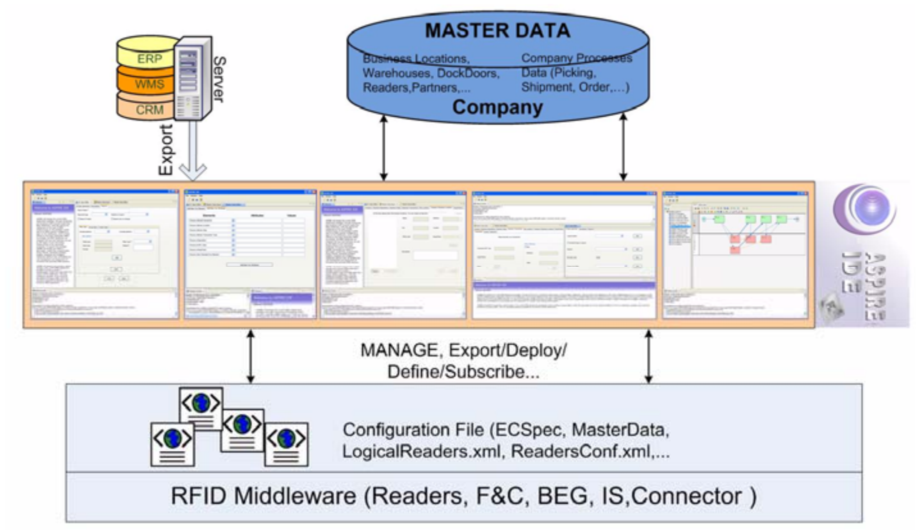

MASTER DATA
Business Locations, Warehouses, DockDoors, Readers,Partners,...
Company Processes Data (Picking, Shipment, Order,...)
Company
ERP
WMS
CRM
Server
Export
ASPIRE IDE
MANAGE, Export/Deploy/ Define/Subscribe...
Configuration File (ECSpec, MasterData, LogicalReaders.xml, ReadersConf.xml,...
RFID Middleware (Readers, F&C, BEG, IS,Connector)

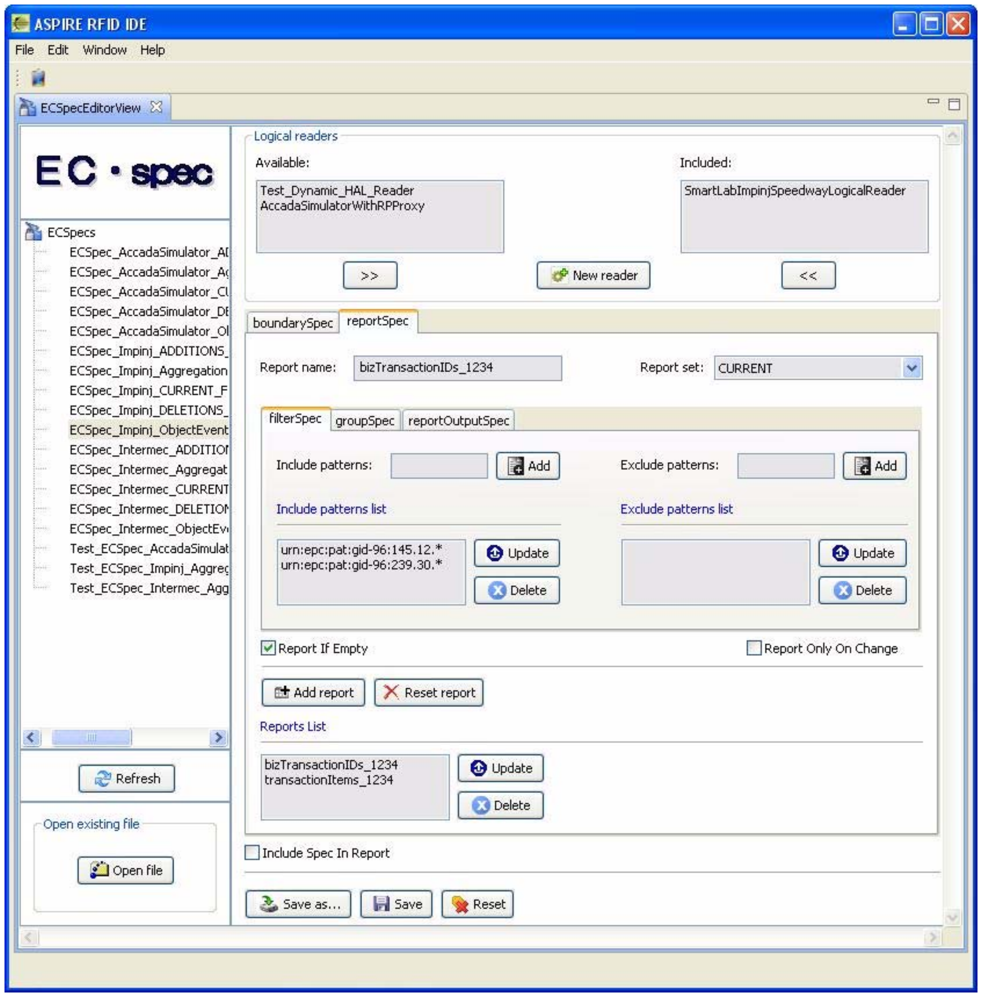
Figure 5 Programmability Tooling

AspireRFID IDE has been designed as an Eclipse RCP (Rich Client Platform) application that runs over Equinox OSGI server. It uses the command API 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 at Figure 6 below.



ECSpecs
ECSpec_AccadaSimulator_AD
ECSpec_AccadaSimulator_Ag
ECSpec_AccadaSimulator_CU
ECSpec_AccadaSimulator_DE
ECSpec_AccadaSimulator_Ol
ECSpec_Impinj_ADDITIONS_
ECSpec_Impinj_Aggregation
ECSpec_Impinj_CURRENT_F
ECSpec_Impinj_DELETIONS_
ECSpec_Impinj_ObjectEvent
ECSpec_Intermec_ADDITION
ECSpec_Intermec_Aggregat
ECSpec_Intermec_CURRENT
ECSpec_Intermec_DELETION
ECSpec_Intermec_ObjectEv
Test_ECSpec_AccadaSimulat
Test_ECSpec_Impinj_Aggreg
Test_ECSpec_Intermec_Agg

Available:
Test_Dynamic_HAL_Reader
AccadaSimulatorWithRPProxy

Included:
SmartLabImpinjSpeedwayLogicalReader

Report name: bizTransactionIDs_1234
Report set: CURRENT

Include patterns:
Exclude patterns:

Include patterns list
urn:epc:pat:gid-96:145.12.*
urn:epc:pat:gid-96:239.30.*

Exclude patterns list

[✓] Report If Empty
[ ] Report Only On Change

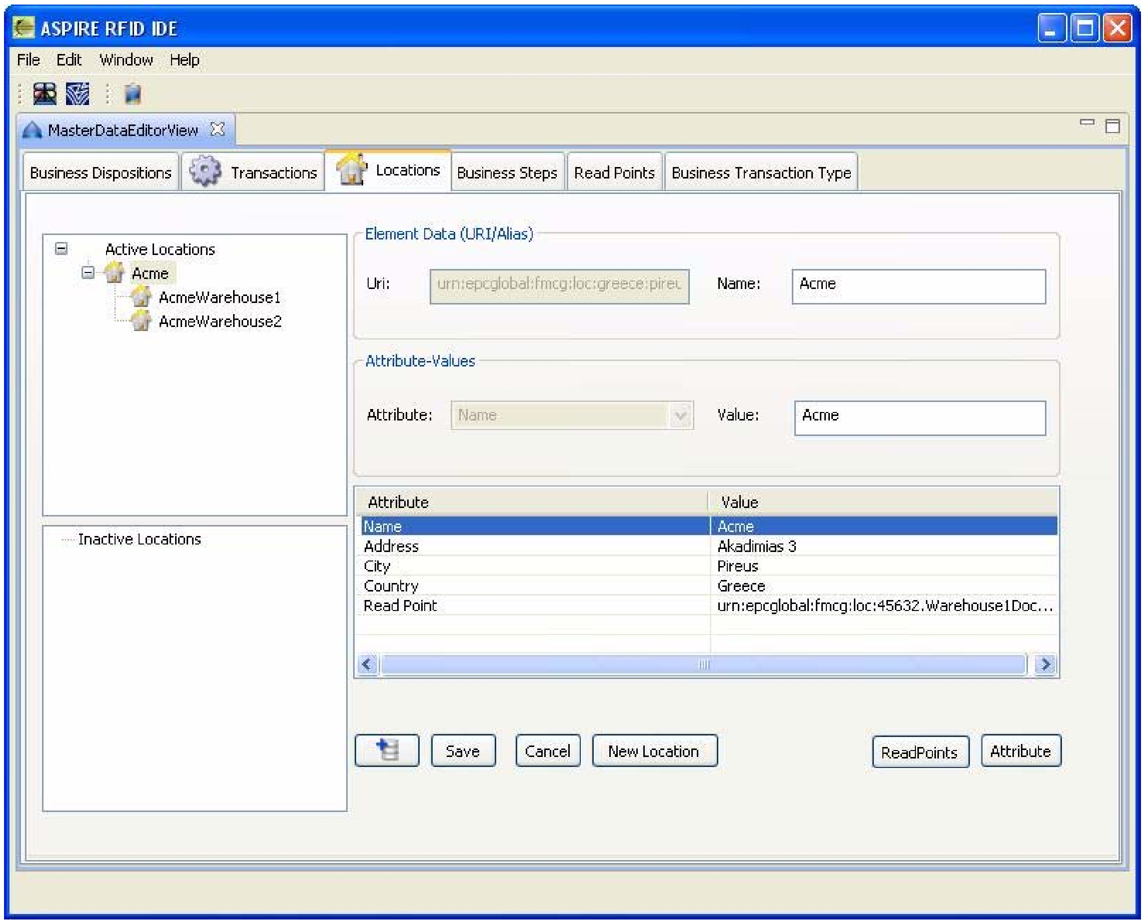
Reports List
bizTransactionIDs_1234
transactionItems_1234

[ ] Include Spec In Report

Figure 6 ECSpec Editor View

8.3 Master Data Editor

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. In Figure 7 below the Master Data editor tab for defining business location is shown.



Uri: urn:epcglobal:fmcg:loc:greece:pireu
Name: Acme

Attribute: Name
Value: Acme

Attribute	Value
Name	Acme
Address	Akadimias 3
City	Pireus
Country	Greece
Read Point	urn:epcglobal:fmcg:loc:45632.Warehouse1Doc...

Figure 7 Master Data Editor (Business Location)

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. ACME at regular basis places orders to the Microchip Manufacturer for specific CPUs. ACME owns a Central building with three Warehouses. The first warehouse named Warehouse1 has 2 Sections named Section1 and Section2. Section1 has an entrance point where the delivered goods arrive.

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

9.2 Solution Requirements

An RFID Portal should be placed to ACME's Warehouse1 Section1 entrance point which will be called ReadPoint1. The RFID portal will be equipped with one Reader WarehouseRfidReader1. The received goods should get equipped with preprogrammed RFID tags from their "Manufacturer". The received goods should be accompanied with a preprogrammed RFID enabled delivery document. And finally AspireRFID middleware (Figure 8 below) should be configured for the specific scenario.

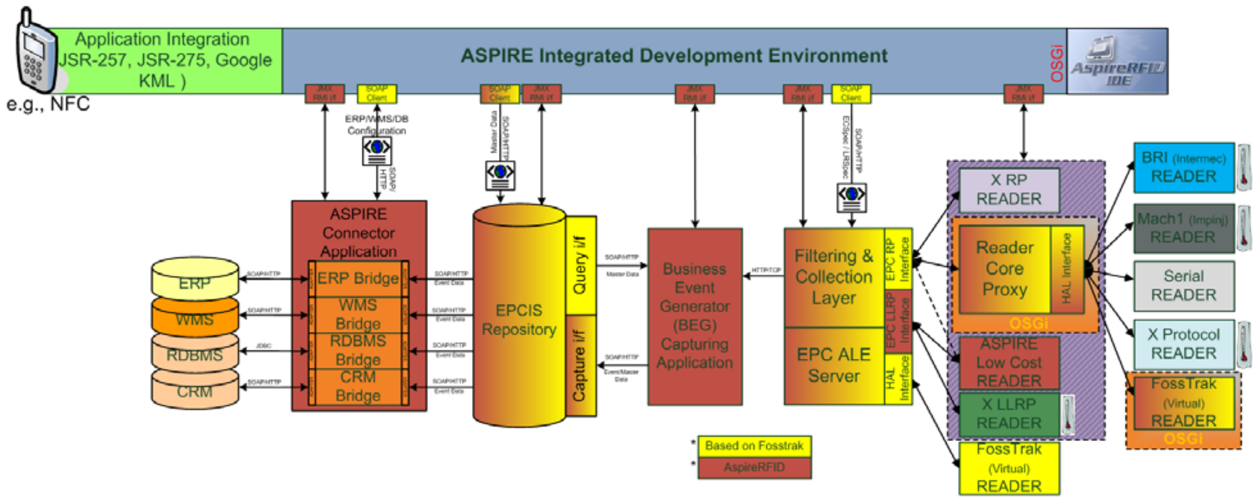


Figure 8 AspireRFID Architecture

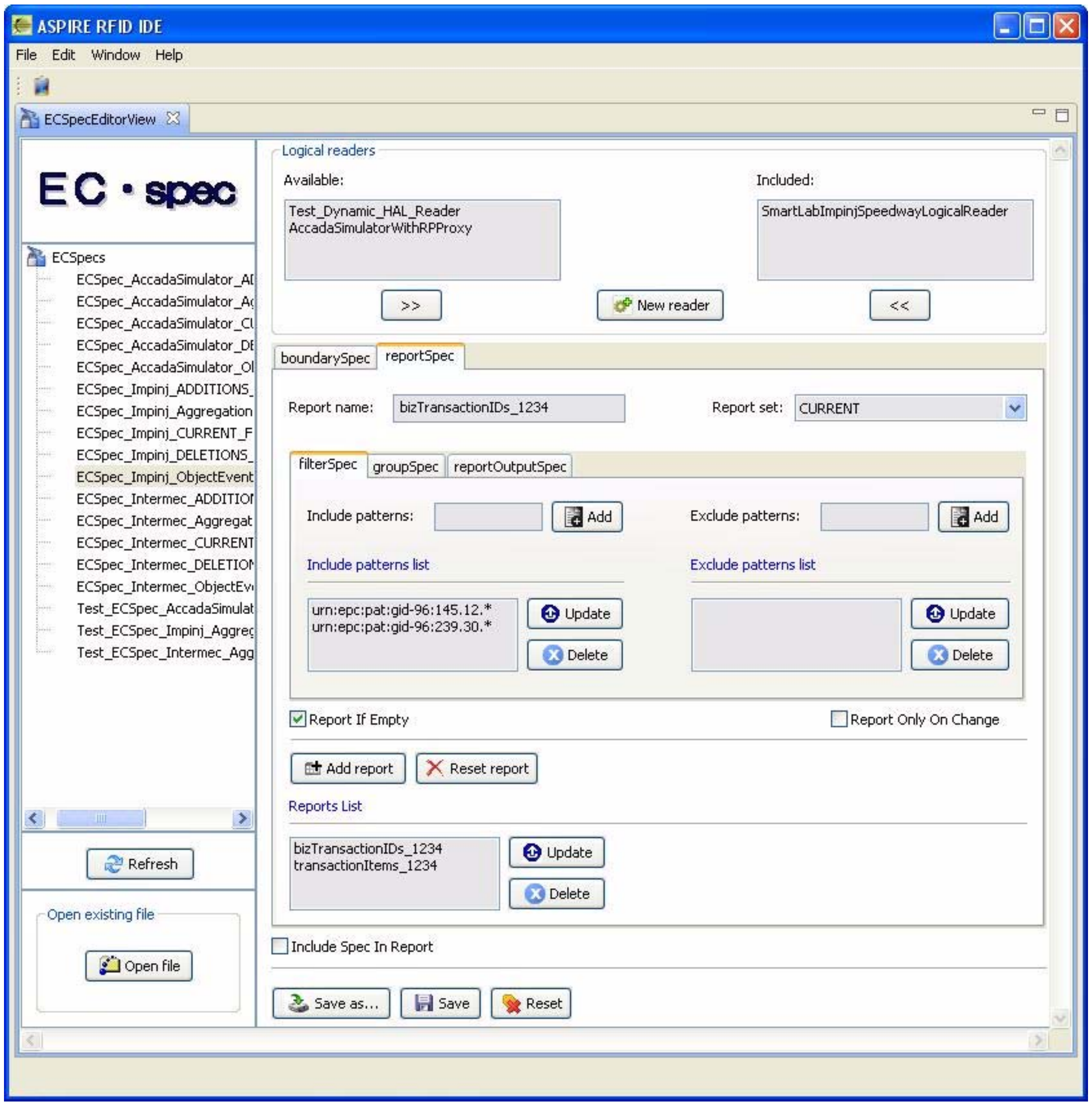
9.3 Setting up the Filtering and collection Module

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 3). 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 9 below



Available: Test_Dynamic_HAL_Reader, AccadaSimulatorWithRPProxy

Included: SmartLabImpinjSpeedwayLogicalReader

Report name: bizTransactionIDs_1234

Report set: CURRENT

Include patterns:

Exclude patterns:

Include patterns list:
urn:epc:pat:gid-96:145.12.*
urn:epc:pat:gid-96:239.30.*

Exclude patterns list:

[✓] Report If Empty

[ ] Report Only On Change

Reports List:
bizTransactionIDs_1234
transactionItems_1234

[ ] Include Spec In Report

Figure 9 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="bizTransactionIDs_1234"
                  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="transactionItems_1234"
                  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>
```

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

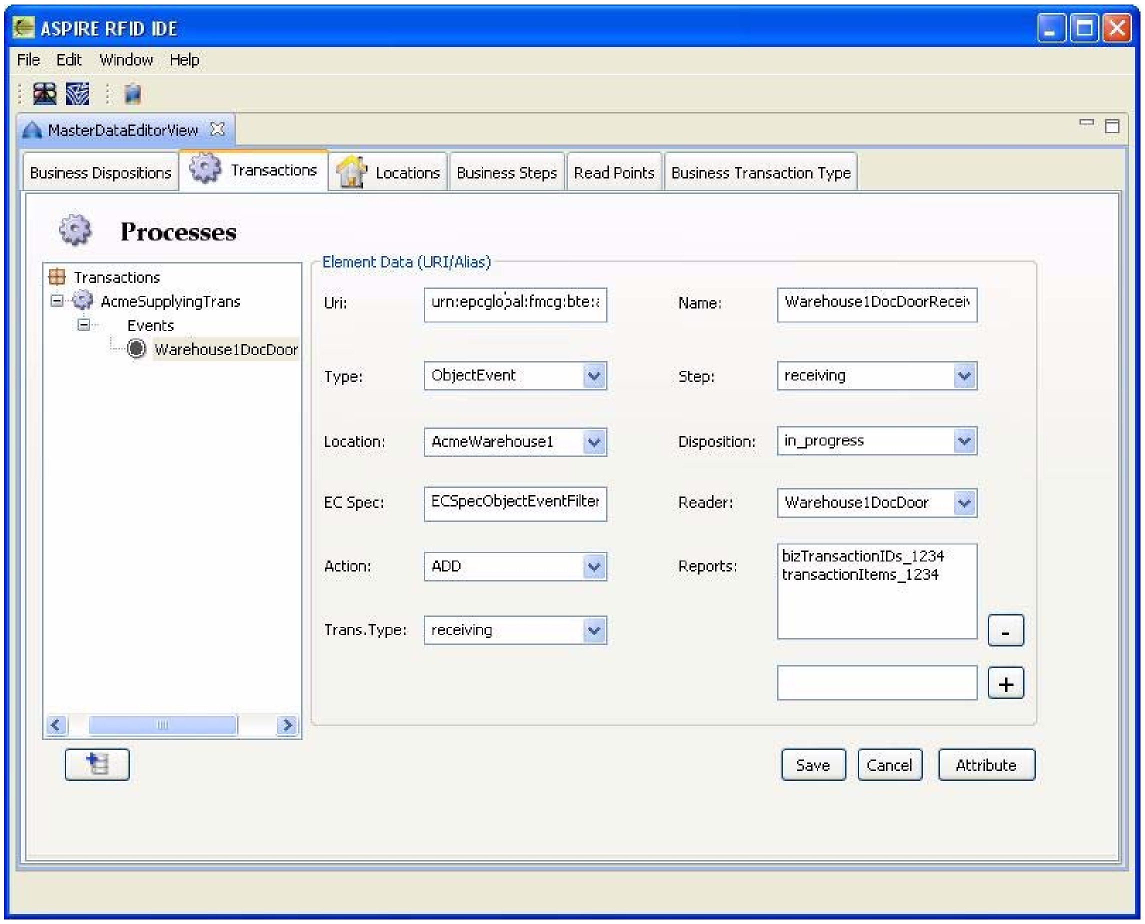
9.4 Setting up the Information Services Module

The Business Event Generator (Figure 3) 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_1234,transactionItems_1234
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:ecspec_name	ECSpecObjectEventFiltering
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 at Figure 10 below.



Uri:	urn:epcglobal:fmcg:bte:a	Name:	Warehouse1DocDoorRecei
Type:	ObjectEvent	Step:	receiving
Location:	AcmeWarehouse1	Disposition:	in_progress
EC Spec:	ECSpecObjectEventFilter	Reader:	Warehouse1DocDoor
Action:	ADD	Reports:	bizTransactionIDs_1234 transactionItems_1234
Trans.Type:	receiving		

Figure 10 Master Data Editor (Specifying a Transaction Event)

9.5 Setting up the Business event generation module

The Business Event Generation module (Figure 3) 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 above.

9.6 Process description

ACME gives an order with a specific deliveryID to the Microchip Manufacturer. With the previous action AspireRfid Connector subscribes to the AspireRfid EPCIS Repository to retrieve events concerning the specific deliveryID.

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 Deliverable 4.3b.

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 11 shows a projection of nodes by types (triangle, square or round).

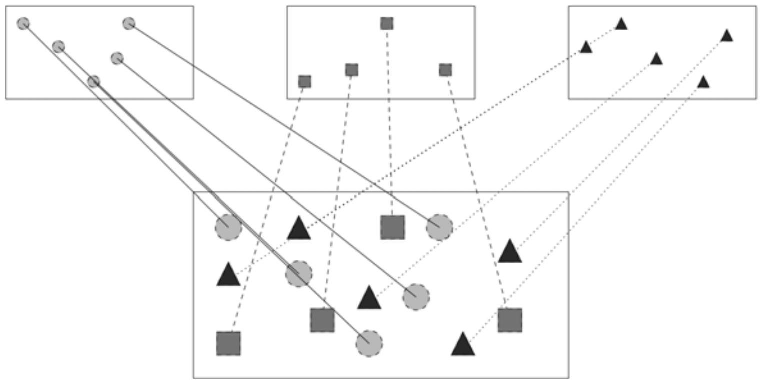


Figure 11 Projection into layers

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 in order 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 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, initial research studies in using DHTs in order to enhance the filtering functionalities of an RFID system have also been provided.

This document is the interim version of the deliverable, thus being the preamble to the final version of the specifications to be released by month 24 (M24) of the project. It is therefore expected that by the time of releasing the second version of this deliverable, mature filtering templates and policies that have been

Section 12 List of Acronyms

ALE	Application Level Event
API	Application Product Interface
ASPIRE	Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications
BEG	Business Event Generator
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
HAL	Hardware Abstraction Layer
HF	High Frequency
HTTP	HiperText Transfer Protocol
IDE	Integrated Development Environment
IT	Information Technology
iPOJO	injected POJO
JMX	Java Management Extensions
LLRP	Low Level Reader Protocol
OBR	OSGi Bundle Repository
OSGI	Open Service Gateway Initiative
OSS	Open Source Software
POJO	Plain Old Java Object
RFID	Radio Frequency Identification
RP	Reader Protocol
SME	Small and Medium Enterprise
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCO	Total Cost of Ownership
TCP	Transfer Control Protocol
UHF	Ultra High Frequency
UML	Universal Markup Language
WADL	Wired Application Description Language
WMS	Warehouse Management System
WP	Work Package
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.