

# Bridging RFID Systems and Enterprise Applications through Virtualized Connectors

Nektarios Leontiadis, Nikos Kefalakis and John Soldatos  
Athens Information Technology, 0,8km Markopoulo Ave., P.O. Box 68, GR-19002 PEANIA, GREECE  
E-mail: {nele@ait.edu.gr, nkef@ait.edu.gr, jsol@ait.edu.gr}  
Phone: {+302106682755, +302106682759}  
Fax: {+302106682703}

## **Abstract**

*Despite the proliferating number of tools and techniques for building RFID applications, their integration with legacy enterprise applications (such as ERP and WMS systems) and corporate databases is still a very tedious task. Integration effort is therefore still a set-back to rapid and cost-effective RFID deployments in non-trivial enterprise environments. In this paper, we introduce a middleware component (conveniently called "Connector"), which abstracts the interfacing of RFID systems with enterprise applications. Our Connector component considers RFID deployments that adopt the EPC Global architecture, i.e. deployments populating RFID events into repositories compliant with the EPC-IS standards. Developers using the "connector" components are offered with handlers for the main interaction messages between EPC-IS repositories and enterprise applications. The paper ends-up presenting the implementation and use of the "Connector" middleware for the interfacing between a WMS system and an EPC-IS repository in the scope of a logistics application. Lessons learnt from this validating case study are also outlined.*

## **1. Introduction**

Applications utilizing Radio Frequency Identification (RFID) are increasingly being adopted in various industries like logistics and consumer services [1]. While the principles of the technology are simple, building applications of this technology in complex enterprise environments is still rather complex [2]. Indeed, non-trivial RFID applications are based on system-level software residing between the user application and the RFID hardware, which is called RFID middleware. RFID middleware is an intermediate software layer, allowing the dialog between heterogeneous applications. It aims to accomplish technical tasks for business applications connection and data exchange. RFID middleware becomes indispensable for complex RFID applications, especially in cases where data stemming from multiple RFID readers and read-points have to be routed to numerous enterprise applications (e.g., ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), WMS (Warehouse Management Systems)) [3]. In the later cases, middleware becomes a unique access point for caught data and it associates read items with an activity and a location [4].

Typical functionality of RFID middleware includes filtering out excessive information (e.g., duplicate reads of the same tags) stemming from the continuous operation of RFID readers. Filtering results in network bandwidth optimization, given that unneeded traffic is reduced. Also, middleware facilitates RFID application developers in interfacing to multiple RFID readers, tags and devices. Indeed, RFID middleware obviates the need to develop custom integration logic for each reader vendor, which is usually very resourceful (in terms of time and budget). Furthermore, it facilitates the routing of different RFID tag streams to different applications and data stores [8].

RFID middleware standards and related suites homogenize the access to RFID data via standardized interfaces for readers management (i.e. through decoupling physical and logical readers), data management (i.e. interfaces for formatting, aggregating and filtering data), invocation of application functions, as well as data collection and

treatment in both synchronous and asynchronous modes. The most prominent set of RFID middleware standards is the one produced by EPCglobal [5], which specifies standardized ways for interfacing to readers (e.g., the EPC-RP and EPC-LLRP standards), filtering and collecting RFID data (i.e. the EPC-ALE specification), as well as tagging, storing and sharing RFID data along with contextual information (i.e. the EPC-IS specification) [6]. EPCglobal and other families of standards consider also standard ways for interfacing to RFID tags. Nevertheless, RFID standards and state-of-the-art middleware suites (e.g., [7]) do not provide a standard way to integrate RFID systems with enterprise applications. In terms of the EPC (Electronic Product Code) architecture and standards, there is no standardized interface (i.e. message exchange) between EPC-IS and enterprise applications. As a result, integration between RFID systems and enterprise applications (such as CRM, ERP, WMS) is currently based on proprietary connectors, which are built as custom software/middleware components that take into account the peculiarities of the enterprise applications. In several cases enterprise applications vendors provide proprietary connectors to RFID systems, in an attempt to provide RFID-enabled enterprise products.

In this landscape, the development of custom software connectors tends to be a tedious and time-consuming task. This is a direct consequence of the fact that RFID integrators need to understand RFID enabled transactions, as well as the way these transactions can be mapped to enterprise applications. In addition, integrators must allocated effort in developing messaging for bridging RFID middleware systems with enterprise applications. As a result, middleware components and libraries that could act as standardized reusable middleware bridges between RFID systems and enterprise applications could greatly boost RFID integration efficiency [9].

Acknowledging this need, this paper specifies a generalized interface for conveying transactions between EPC-enabled RFID systems and enterprise applications. This interface capture the semantics of the business transactions in RFID-enabled enterprise systems, while at the same time “virtualizing” the bridging between the two. Moreover, it introduces a related middleware library, which is conveniently called “Connector”, as a reusable and extensible middleware bridge between the above systems. The “Connector” interface and related library have been designed and implemented according to the specification of RFID events and transactions within the EPC-IS standards [10]. In particular, the “Connector” library connects EPC enabled RFID systems with enterprise applications, using the EPC-IS semantics in order to convey transaction boundaries (i.e. when information from the RFID system must be conveyed to the enterprise application and vice versa), as well as the contents for the “Connector” interactions (e.g., objects within an order). Hence, the “Connector” enables legacy ICT systems to be part of an existing or new RFID installation that complies with EPC-IS standards. The extensibility and reusability of the “Connector” library makes it applicable across many different RFID applications, provided that the later comply with RFID standards. As a proof of concept, we present how the “Connector” is leveraged and used in the scope of a sample RFID application for warehouse management [11], which has been built by the authors.

The structure of the paper is as follows: Following this introductory paragraph, Section 2 outlines the main elements of RFID systems that follow the EPC Architectural framework. The aim of this section is to facilitate the understanding of the scope of the systems that could use the “Connector” (middleware) bridge. Then, section 3 elaborates on the “Connector” concept and positions it within the EPC architecture. Section 4 introduces the “Connector” virtualized interface, while Section 5 illustrates the proof-of-concept application. Finally, section 6 draws basic conclusions.

## 2. Overview of the EPC Architectural Framework and Related Standards

The EPC Architecture framework specifies the main middleware building blocks for an RFID system, along with information flows and interactions between them. The typical information flow within an RFID middleware system, as established in the EPC Architecture involves [9], [12]:

- Collecting RFID data from the physical readers, through reading the tagged items. At this level middleware implementations insulate higher layers from knowing what reader /models have been chosen. Moreover, they achieve virtualization of tags, which allows RFID applications to support different tag formats. The EPC-RP (Reader protocol) and EPC-LLRP (Low-Level Reader Protocol) ensure standardize access to RFID interrogators, whereas the EPC-TDT (Tag Data Translation) deals with the required independence from the peculiarities of the tag format.
- Filtering the RFID sensor streams according to application needs, and accordingly emitting application level events. At this level middleware implementations insulate the higher layers from the physical design choices on how tags are sensed and accumulated, and how the time boundaries of events are triggered. The most prominent EPC standard at this layer is EPC-ALE (Application Level Events).
- Mapping the filtered readings to business semantics as required by the target applications and business processes. This mapping can be performed by a Business Event Generator (BEG) component (as shown in **Figure 2** and illustrated in [14]). At this level middleware implementations insulate enterprise applications from understanding the details of how individual steps in a business process are carried out. Information at this level is stored in the Information Sharing repository, which is structured according to the EPC-IS (Information Sharing) standard.

Based on the above architecture, enterprise applications wishing to exchange information with the EPC compliant RFID system, must interact with the EPC-IS repository. EPC-IS persists (supply chain oriented [13]) events in a repository and accordingly shares these events with internal (i.e. intra-enterprise) and external (intra-enterprise) applications. The sharing is accomplished through interfaces for capture and query of event data. EPC-IS events are represented as records of activity happening in real world. These events provide information on “What” (i.e. the tagged objects read), “Where” (i.e. the read point/location of the reading), “When” (i.e. the timestamp of the reading), “Why” (i.e. the business step or context) and proliferate as more business is transacted. EPC-IS events are interpreted based on descriptive information (so called “master data”), which provide context for the events such as descriptions of locations, products and business transactions. Contrary to EPC-IS events which proliferate as more business is transacted/conducted, master data grow slowly as companies grow.

Hence, the EPC-IS specification offers a standardized way of describing information at the business level. EPC-IS is prescribed as a both industry and application agnostic framework, which can be appropriately extended based on industry-specific vocabularies, as well as user extensions. EPC-IS constitutes a first step towards a business level specification of RFID enabled transactions. EPC-IS focuses on individual transactions (or business steps), which can be combined to form a wide business process. Under this definition a business process is defined as a workflow combination of business events [14].

EPC-IS events can be classified as follows (as depicted in Figure 1):

- Object Events, which correspond to observations of a collection of EPCs during a specific business step at a specified Location & Time.

- Aggregation Events, which reflect a physical association of a set of EPCs with a parent EPC along with a business step at a Location & Time.
- Quantity Events, which correspond to statements about an object Class (not individual objects), including a quantity, a Location & Time.
- Transaction Events, which records objects associated with a wider business transaction.

Using these events, consultants, researchers and engineers can describe RFID enabled business processes. Business requirements and use cases can be resolved into a series of discrete business steps. Each one of these steps is modelled as an EPC-IS event, based on the above core EPC-IS event types or (in rare cases) new types, which could be defined when existing types are not sufficient to describing a business step. As illustrated in the following section the semantics of the EPC-IS events drive the specification and implementation of the “Connector” interface and middleware bridge.

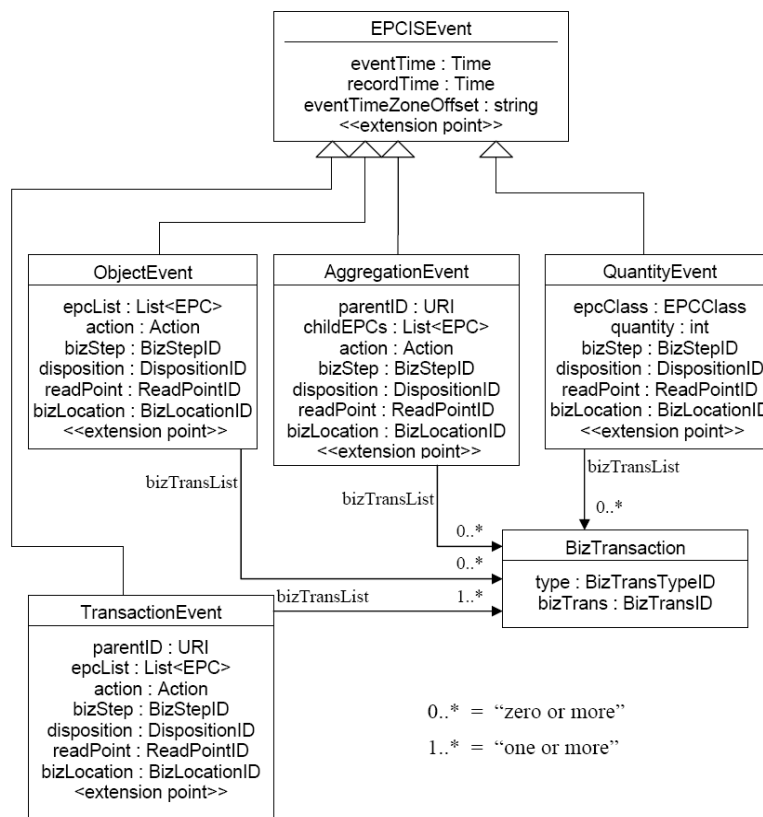


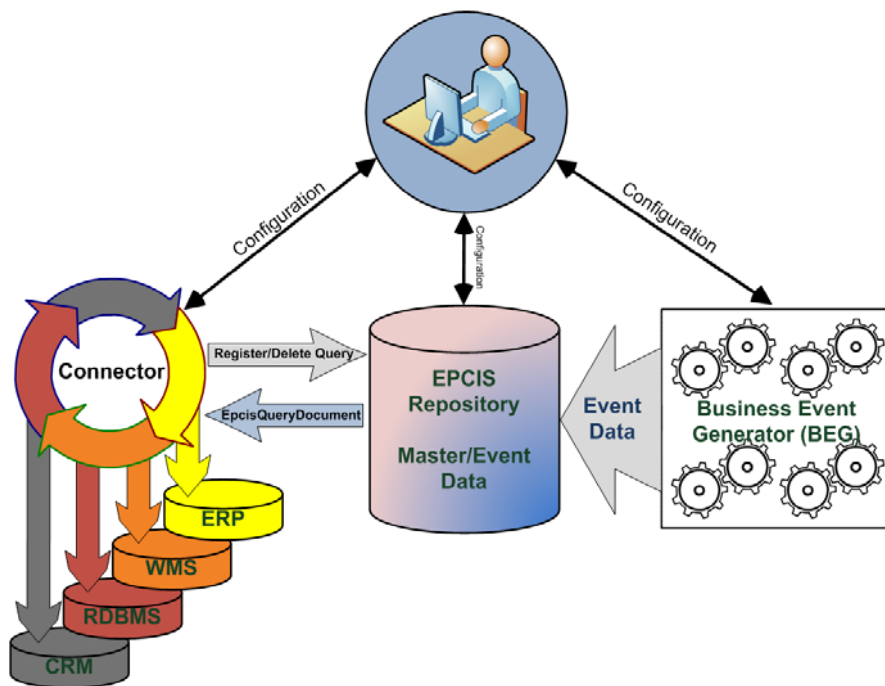
Figure 1: Core EPC-IS Event Types (according to [10])

### 3. The Connector Concept

RFID middleware components described in previous paragraphs provide a foundation for translating raw RFID streams to meaningful business events comprising business context such as where a tag was seen, at what time and in the scope of which process. Enterprises can then leverage these business events through their legacy IT systems (e.g., ERPs, WMS, corporate databases), which are used to support their business processes. As already outlined, EPC-IS defines a data model for events associated with the lifetime of uniquely identified objects. These events are industry and (enterprise) application agnostic. In this sense EPC-IS is a cross industry framework, which allows for industry specific vocabularies and extensions. Furthermore, the framework is a supplement to (and not a replacement for) existing enterprise information systems. Specifically, EPC-IS events are used to push/pull events to/from other enterprise

systems such as ERP (Enterprise Resource Planning Systems), WMS (Warehouse Management Systems) and corporate databases.

To this end, there is a clear need for interfacing the enterprise applications (including legacy applications), with the information sharing repositories, established and populated as part of the RFID deployment. In order to realize the interfacing between enterprise applications and the information sharing repository (EPC-IS), we introduce a specialized middleware component, which we call “Connector”. The “Connector” shown in **Error! Reference source not found.** represents a generic interface that can enable the actual large-scale deployment of a RFID infrastructure in an existing corporate environment.



**Figure 2: The concept of the Connector Component**

The main purpose of a “Connector” is to abstract the interface between the EPC information sharing repository and the enterprise information systems and provide interfaces to produce more industry or application specific messages. Hence, the “Connector” offers application programming interfaces (APIs) that enable proprietary enterprise information systems to exchange business information with a RFID middleware system. As a result, a “Connector” provides:

- **Support for services and events:** Composite applications can call out to existing functionality as a set of services, and to be notified when a particular event type (for example, “purchase order inserted,” “employee hired”) occurs within an existing application.
- **Service abstraction:** All services have some common properties, including error handling, syntax, and calling mechanisms. They also have common access mechanisms such as JCA (Java Connector Architecture), JDBC, ODBC (Object Database Connectivity), and Web services, ideally spanning different platforms. This makes the services more reusable, while also allowing them to share communications, load balancing, and other non-service-specific capabilities.
- **Functionality abstraction:** Individual services are driven by metadata about the transactions that the business needs to execute.

- **Process management:** Services embed processes, and process management tools call services. Hence, connectors support the grouping of several service invocations to processes.

In this way, a corporation having a legacy enterprise application or IT system can install and communicate with a RFID infrastructure without needing to change the IT infrastructure or significantly alter it. The effort needed to succeed towards the direction of incorporating the RFID infrastructure into the information loop is designed to be minimal and as it is going to be explained later on. The specific “Connector” introduced in this paper specifies a set of capabilities that every implementation should advertise to the actual consumers of data and information stemming from EPC compliant RFID middleware infrastructure (producers).

#### **4. Interface Specification of the Connector Component**

The “Connector” is intended to provide a means of transparency between an application and the EPC-IS repository. Typically, an EPC-IS repository, collects business events, and provides this information using either a push or a pull concept. The “Connector” is a two-tier component, namely the Connector Engine (CE) and the Connector Client (CC) that operate in the basis of a transaction. These two collaborating tiers enable the user application to receive EPC-IS events for specified operations called transactions, using the push or the pull model in a uniform way. Hence, an application can either subscribe to a specific type of operation and when these occur, the application can be notified by the connector, or an application can request information about past observations - defined by time boundaries - of the specific operation.

In the following sections we will describe the tiers that define the “Connector” interface specification, along with the messages that are used for the internal communication between the tiers and the external communication of the Connector with the EPC-IS and with the client application.

##### **4.1. The Connector Engine**

This tier that is part of the Connector component, has been designed to be "next-to" the EPC-IS component. In particular, the CE (Connector Engine) is responsible for the communication of the Connector with the EPC-IS repository through the EPC-IS Query and Capture Interfaces [10]. The CE tier interface specification provides also a Web Service (WS) interface to receive requests from the CC (Connector Client) tier regarding subscription and polling requests for specific transaction type. This interface is extensible and enables the definition of additional operations. The web service specification defines two operations, namely the:

- startObservingTransaction, and the
- stopObservingTransaction

Both of these operations take as an argument a **subscriptionParameters** type. The decision to use this construct in both operations permits us to handle in a uniform way poll or subscription requests.

A transaction, as described before, can be mapped to events that can be understood by enterprise applications. By invoking the **startObservingTransaction** operation, the legacy system will be notified for this event. If the observation is based on the push model (subscription) and depending on the defined intervals for checking if the event has occurred, the legacy system will be notified by the CC about this occurrence whenever the time triggers are met. If the observation is based on the pull model (poll),

the legacy system will be notified about the past occurrences - limited by the time constraints - immediately.

The invocation of the **stopObservingTransaction** is designed to be invoked by legacy applications that have previously used the **startObservingTransaction** operation to **subscribe** to specific transaction events. This operation handles the cancellation of the subscription on the EPC-IS side and the mandatory alteration of specific Master Data in the EPC-IS repository through a transaction delete operation. The **subscriptionParameters** construct is defined in XML in Table 1, while Table 2 provides a description for each of the fields of the Subscription Parameters.

```
<xs:complexType name="subscriptionParameters">
  <xs:sequence>
    <xs:element name="doPoll" type="xs:boolean"/>
    <xs:element minOccurs="0" name="initialTime" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="queryDayOfMonth" type="xs:string"/>
    <xs:element minOccurs="0" name="queryDayOfWeek" type="xs:string"/>
    <xs:element minOccurs="0" name="queryHour" type="xs:string"/>
    <xs:element minOccurs="0" name="queryMin" type="xs:string"/>
    <xs:element minOccurs="0" name="queryMonth" type="xs:string"/>
    <xs:element minOccurs="0" name="querySec" type="xs:string"/>
    <xs:element minOccurs="0" name="replyEndpoint" type="xs:string"/>
    <xs:element name="reportIfEmpty" type="xs:boolean"/>
    <xs:element minOccurs="0" name="subscriptionId" type="xs:string"/>
    <xs:element minOccurs="0" name="transactionId" type="xs:string"/>
    <xs:element minOccurs="0" name="transactionType" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

**Table 1: XML Schema Description of the Subscription Parameters**

Field name	XML Field type	Description
doPoll	boolean	This parameter handles that way a request will be processed. If false, then a new subscription will be registered within the EPC-IS with information that is provided with other elements. If true then the query will be executed only once and the results will be returned immediately. In any case the <b>replyEndpoint</b> will be used to send the result.
querySec, queryDayOfWeek, queryHour, queryMin, queryMonth, queryDayOfMonth	string	The category of <b>queryX</b> parameters define the time interval that the query will be executed within the EPC-IS repository if doPoll is false. At least one of these parameters should be defined in this case. These parameters take a comma separated list of integers that define the query schedule. For example, if querySec has the value <i>1,31</i> then the query will be executed on the 1st and 31st second of every minute. [1]
replyEndpoint	string	Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive.
reportIfEmpty	boolean	Indicates whether a query result will be send to the Connector client even if there is no matching event.

subscriptionId	string	Should be a universally unique identifier to identify a subscription.
transactionId	string	Indicates the events that we are interested in.
transactionType	string	Indicates the optional event type that we are interested in.
initialTime	dateTime	This parameter defines the time constraint after which all matching events will be returned. If doPoll is false, and the initialTime refers to the past, the old matching events will be returned within the first response message.

**Table 2: XML Schema Description of the Subscription Parameters**

#### 4.2. The Connector Client

This component has been designed to be located on the side of the (possible legacy) enterprise application and support its interactions with the RFID infrastructure. It is responsible for the following number of operations:

- Provide an interface to the legacy IT systems for receiving query (subscription or polling) requests through a provided application programming interface or through a web service.
- Submit the queries to the Connector Engine that it interacts with.
- Receive query responses from the CE. The responses may either be a response to a push (subscription) or pull (poll) request, and
- Pass the information to the legacy software.

```

<xs:complexType name="event">
  <xs:sequence>
    <xs:element minOccurs="0" name="action" type="xs:string"/>
    <xs:element minOccurs="0" name="bizLocationId" type="xs:string"/>
    <xs:element minOccurs="0" name="bizStepId" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="bizTransactionList"
      nillable="true" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="childEpcs"
      nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="dispositionId" type="xs:string"/>
    <xs:element minOccurs="0" name="epcClass" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="epcList" nillable="true"
      type="xs:string"/>
    <xs:element name="eventTime" type="xs:long"/>
    <xs:element minOccurs="0" name="parentId" type="xs:string"/>
    <xs:element name="quantity" type="xs:int"/>
    <xs:element minOccurs="0" name="readPointId" type="xs:string"/>
    <xs:element minOccurs="0" name="subscriptionId" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:complexType>

```

**Table 3: XML Schema Description of the Event Structure**

The component provides a Web Service (WS) to receive events from the CE based on subscribed or polled queries. This operation is called asynchronously from the CE component when event information is available and receives an Event object that encapsulates information provided by EPC-IS events and is defined in the EPC-IS

specification. At Table 3 we can see a definition of the Event structure in XML format. By encapsulating all the required information within one specific structure instead of four that the EPC-IS specification defines, enables enterprise applications to handle common events captured by the RFID infrastructure with a minimal development, testing and deployment effort.

An enterprise application wishing to interact with an RFID infrastructure that is connector-enabled would only need to attach to CC by implementing a small number of operations, which would handle the Events whenever they occur and by enabling the submission of queries through calls to the CC component.

### **5. Validating Use case**

In order to illustrate and validate the functionality of the introduced RFID middleware module, we hereby present a sample use case based on the fictitious PC assembler company named "Acme". Acme places orders for various computer parts as CPUs, memory modules, and hard drives to two other companies with which, it collaborates. The management would like to automate various business processes as the order of products, the shipping of assembled computers to customers, and the tracking of the various parts inside the company's premises.

To meet these goals, RFID readers should be installed in all the key places that the various processes are being carried out (e.g. the entry point of Acme, where the products ordered arrive) so that the readers will be able to detect the tagged items movements and the appropriate events be generated. Once the readers are in place the RFID middleware should be deployed and configured to support the business's Master Data and Elementary Business Transactions [15-16]. This means that the processes, have already been properly documented and accurately analyzed into a series of business steps, and each step has been mapped to the appropriate EPC-IS event. Note that the items are delivered with EPC tags already attached (so that the readers can detect them).

If we would rely on the standardized components of the EPCglobal architecture it should also be required to build a median application for the communication between the WMS and the EPC-IS repository. The effort required for it would be huge, because the WMS would need to understand EPC-IS messages and protocols to subscribe to events and to translate the EPC-IS events to WMS specific events. So in our case we instead utilized the Connector Engine. For the Connector Engine configuration only the connection End-Points (EPC-IS, Connector Client and Connector Engine) and the query call back timings are required described above. For that we are using a special software (management tool) called Connector Configurator, which facilitates the configuration of the middleware bridge.

As soon as an order is placed from the Warehouse Management System the Connector Client which is embedded to it sends a **Transaction Start** message as shown in Figure 3 with the specific order ID to the Connector Engine. The CE in its turn sends a **startObservingTransaction** message to the EPC-IS Repository which registers for the specific transaction ID for the preconfigured timings.

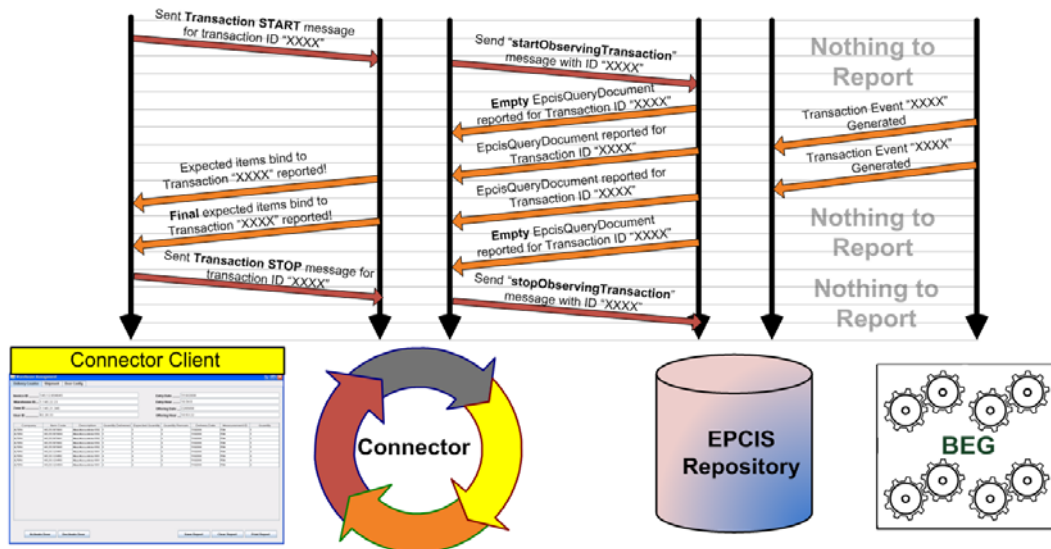


Figure 3: Connector Engine message flow

Given the above setup, as we can see in Figure 4, when the items arrive to the entry point of Acme the RFID reader there detects the EPC tag attached and delivers them to the Filtering and collection layer for filtering and “refinement”. The generated reports are then delivered to the Business Event Generator layer [14] which is responsible to add business semantics to them and generate the desired Event Data [16]. Till then the Connector Engine would receive empty reports from the EPC-IS’s QueryCallBack interface till the specific Transaction ID Event Data are captured from the EPC-IS repository. As soon as the data are captured and stored to the EPC-IS the next EPC-IS report to the Connector Engine will contain an **EPC-ISQueryDocument** with the specific transaction ID and the transaction Items bound to it. The Connector Engine in its turn sends these data to the Connector Client. This process continues till the WMS receive all the ordered items. Then a **Transaction Finish** is send back to the Connector Engine which in its turn sends a stopObservingTransaction which unregisters the specific transaction ID from the EPC-IS’s QueryCallBack interface and disassociates the transaction’s items from the Transaction ID (Transaction Delete).

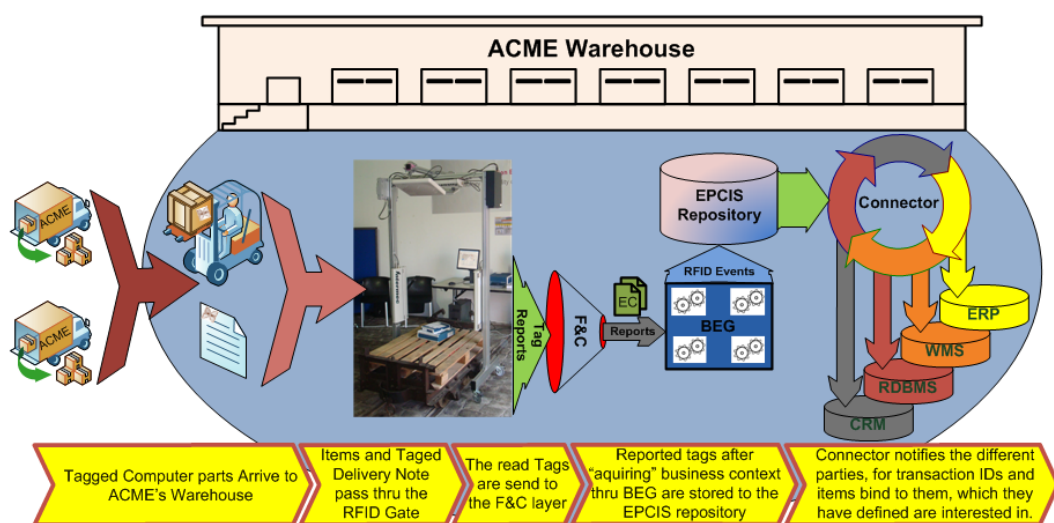


Figure 4: Computer Parts Delivery Scenario

Note that the above use case has been deployed and demonstrated (as a lab demonstration) in the scope of the ASPIRE project [18], which is co-funded by the European Commission. Likewise, the “Connector” middleware is part of the AspireRfid

Open Source RFID project [19], which aims at providing tools and techniques for facilitating RFID deployment [20].

## **6. Conclusions**

Integrating legacy business ICT applications with state of the art RFID infrastructures can significantly enhance the experience of doing business by minimizing the obsolete costs that manual processes impose. In this paper we have presented the concept of the Connector that enables the rapid deployment and integration of the RFID technology into the legacy applications. While this concept is concrete in its purpose it does not dictate the implementation specifics, thus boosting wider acceptance and technological longevity. The “Connector” middleware provides a number of benefits for developers, integrators and consultants in the scope of complex RFID deployments, including:

- Increased return on investment (ROI), as well as reduced total-cost-of-ownership (TCO) Long-term for integrated RFID enabled business processes, since “Connectors” implemented in one project, can be reused in other implementations.
- Reduced reliance on specific vendors and technologies, since “Connectors” minimize the use of proprietary middleware bridges and integration protocols.
- Better support for RFID standards (such as EPC). “Connector” implementation could incorporate any changes to baseline standards (e.g., EPC-IS), thus allowing users to emphasize on how standards affect the business side rather than on the low-level technicalities of the evolving specifications.
- Improved monitoring of RFID enabled business processes, since “Connectors” can convey consistent information about what message exchanges between RFID systems and enterprise applications.

Future work involves the development and instantiation of additional “connectors” for business scenarios other than logistics and WMS, as well as interfaces for managing and configuring “connectors” for different enterprise applications. As RFID applications proliferate, we envisage a higher penetration and exploitation of the “connector” concept for simplified enterprise applications integration in the RFID domain.

## **Acknowledgements**

Part of this work has been carried out in the scope of the ASPIRE project (<http://www.fp7-aspire.eu>), which is co-funded by the European Commission (EC) in the scope of the FP7 framework programme (contract no: FP7-215417). The authors acknowledge support from the EC, as well as help and contributions from all partners of the project.

## **References**

- [1] Nath, B.; Reynolds, F.; Want, R., ‘RFID Technology and Applications’, IEEE Pervasive Computing, Vol. 5, No. 1, Jan.-March 2006, pp. 22- 24.
- [2] S. Sarma, “Integrating RFID,” ACM Queue, vol. 2, no. 7, pp. 50–57, 2004.
- [3] Christian Floerkemeier, Christof Roduner, and Matthias Lampe, ‘RFID Application Development with the Accada Middleware Platform’, IEEE Systems Journal, Vol. 1, Issue 2, pp.82-94, December 2007.
- [4] Nikos Zarokostas, Panagiotis Dimitropoulos and John Soldatos, "RFID Middleware Design for enhancing traceability in the Supply Chain Management", in the Proc. of the 18th IEEE Personal Indoor and Mobile Radio Communications, Athens, Greece, September 3-7, 2007.
- [5] Sarma, S. (2005). A History of the EPC. In: Garfinkel, S.; Rosenberg, B. (eds.) (2005). RFID. 37–55, Addison-Wesley, Upper Saddle River.
- [6] Architecture Review Committee, “The EPCglobal Architecture Framework,” EPCglobal, July 2005, available at: <http://www.epcglobalinc.org>.

- [7] BEAWebLogic RFID Enterprise Server™, “Understanding the Event, Master Data, and Data Exchange Services”, Version 2.0, Revised: October 12, 2006.
- [8] C. Floerkemeier and M. Lampe, “RFID middleware design – addressing application requirements and RFID constraints,” in Proceedings of SOC’2005 (Smart Objects Conference), Grenoble, France, Oct. 2005, pp. 219–224.
- [9] Nikos Kefalakis, Nektarios Leontiadis, John Soldatos, and Didier Donsez, “Middleware Building Blocks for Architecting RFID Systems”, in Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, F. Granelli et al. (Eds.): MOBILIGHT 2009, LNICST 13, pp. 325–336, 2009.
- [10] “EPC Information Services (EPC-IS) Version 1.0, Specification,” EPCglobal, Apr. 2007, available at: <http://www.epcglobalinc.org>
- [11] James A. Tompkins, “Warehouse Management Handbook”, Tompkins Press; 2nd edition (October 1998), ISBN-10: 0965865916, ISBN-13: 978-0965865913.
- [12] Nikos Kefalakis, Nektarios Leontiadis, John Soldatos, Kiev Gama and Didier Donsez, “Supply Chain Management and NFC Picking Demonstrations using the AspireRfid Middleware Platform”, Demonstration in the scope of the ACM Middleware 2008 conference, Leuven, Belgium, December 1-5, 2008.
- [13] Agrawal, D.K. (2003), “Logistics and Supply Chain Management”, Macmillan India Ltd., Bombay.
- [14] Panos Dimitropoulos and John Soldatos, ‘RFID-enabled Fully Automated Warehouse Management: Adding the Business Context’, accepted for publication to the International Journal of Manufacturing Technology and Management (IJMTM), Special Issue on: "AIT-driven Manufacturing and Management" to appear 2009.
- [15] John Soldatos, Nikos Kefalakis, Nektarios Leontiadis, et. al., “Programmable Filters – FML Specification”, ASPIRE Project Public Deliverable D4.3a, April 2009, publicly available at: <http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables>
- [16] Nikos Kefalakis, John Soldatos, Sofoklis Efremidis, et. al., “Programmable RFID Solutions Specification (Interim Version)”, ASPIRE Project Public Deliverable D4.4a, Sept 2009, publicly available at: <http://wiki.aspire.ow2.org/xwiki/bin/view/Main.Documentation/Deliverables>
- [17] Vijayaraman, B.S.; Osyk, B.A. (2006). An empirical study of RFID implementation in the warehousing industry. The International Journal of Logistics Management, 17 (1), pp. 6-20.
- [18] The ASPIRE FP7 Project, <http://www.fp7-aspire.eu>,
- [19] The AspireRfid project, <http://forge.objectweb.org/projects/aspire/> (forge), <http://wiki.aspire.objectweb.org/xwiki/bin/view/Main/WebHome> (wiki)
- [20] John Soldatos, “AspireRfid Can Lower Deployment Costs”, RFID Journal, March 16th, 2009.