An Analysis of Physical Object Information Flow within Auto-ID Infrastructure

by

Tatsuya Inaba

BACHELOR OF ENGINEERING IN ELECTRICAL ENGINEERING
UNIVERSITY OF TOKYO, 1991

Submitted to the Center Engineering Systems Division in Partial Fulfillment of the Requirements for the Degree of

MASTER OF ENGINEERING IN LOGOSTICS

at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2004

Signature of Author.....
Engineering Systems Division
May 7, 2004

Certified by.....
George A. Kocur
Senior Lecturer of Department of Civil and Environment Engineering
Thesis Supervisor

Certified by.....
James B. Rice
Director, ISCM Program of Center for Transportation and Logistics
Thesis Supervisor

Accepted by.....
Yosef Sheffi
Professor, Engineering Systems Division
Professor, Civil and Environmental Engineering Department
Director, MIT Center for Transportation and Logistics

An Analysis of Physical Object Information Flow within Auto-ID Infrastructure

by

Tatsuya Inaba

Submitted to the Engineering Systems Division on May 7, 2004 in Partial Fulfillment of the Requirements for the Degree of

MASTER OF ENGINEERING IN LOGISTICS

ABSTRACT

The application of Radio Frequency Identification (RFID) has been studied for decades, and many field trials have been executed to evaluate the usability of RFID systems, the business case of RFID applications and so forth. One of the trial fields is its application to supply chain management (SCM) because the RFID technologies are thought to improve visibility of physical objects dramatically. Through this trial phase, benefits and feasibility of RFID have been confirmed, and as a result, major retailers, such as Wal-Mart, Target, and Metro, have decided to implement RFID. At the same time, these trials reveal the necessity of RFID standards. Among these newly developed RFID standards, Auto-ID standard, which was originally developed by Auto-ID Center, is a strong candidate to be a de-facto standard.

Auto-ID technologies consist of data standards and software architecture components. Data standards also consist of two components: Electronic Product Code (EPC) and Physical Markup Language (PML). On the other hand, software architecture components consist of four components: readers, Savant, EPC Information Server (EPC-IS), and Object Name Service (ONS). EPC-IS, which defines the interface of the servers that store physical object information, plays a key role in realizing business processes that the RFID technologies are expected to realize. In this thesis, we propose architecture of EPC-IS by defining the requirements for EPC-IS through generic business processes executed in Auto-ID infrastructure. The architecture we propose is not a monolithic message schema but three simple message schemas with vocabulary sets that are separately defined in dictionaries. By taking this structure, we achieve robust and scalable interface. We also evaluate our proposal by applying it to the problems found in the RFID trials and possible future business processes.

Thesis Supervisor: George A. Kocur
Title: Senior Lecturer of Civil and Environmental Engineering

Thesis Supervisor: James B. Rice
Title: Director of ISCM Program of Center for Transportation and Logistics

Acknowledgement

First and foremost, I would like to thank to my family to understand my will and support my study at MIT. Without their understanding and support I had not had the chance to study and mingle with my friends at MIT.

I also thank to my friends who cheer me up and give me valuable advices. With their support, I can keep high motivation to learn new things and write a thesis.

Regarding the thesis, I am very grateful for the support, advices and encouragement from my advisors George Kocur and Jim Rice. Their advices helped me to focus the thesis topic, construct the logic, and break the wall when I was stuck.

I also would like to appreciate all at Auto-ID Lab, especially Robin Koh. The discussion with him was very informative and helped me to develop the scope of my thesis. He sometimes pushed me to move forward and gave me advices and encouragement. He also gave me many contacts that are essential to my thesis. The visit to the Auto-ID Lab was one of the most valuable experiences I had through this thesis writing.

I would like to thank to the architects at EPCglobal, especially Ted Osinski. The advices from him helped me to develop the proposal I made in this thesis.

Last but not least, I would like to show my appreciation to all the staff and friends at MLOG Program. The experience we had, to discuss, to chat, to help each other and to eat and drink, made the life at MIT more valuable, fruitful, and enjoyable.

Table of Content

1. Introduction

The history of Radio Frequency Identification (RFID) is long, and studies and implementations of RFID go back to the mid-20th century [1]. Although the technology is categorized as RFID, the technology behind RFID applications at that time was different from that of today. Currently RFID technology premises the use of IC tag, which stores information about the object to which the IC tag is attached. This IC tag based RFID technology has been studied for decades, and based on these studies field trials have been executed to evaluate wide range of applications.

One of the trial fields of RFID has been its application to supply chain management (SCM) because this technology is thought to dramatically improve the visibility of physical objects and this increased visibility will bring companies in the supply chain many benefits, such as the reduction of inventory, reduction of inventory management costs, and realization of value added service.

Objectives of these trials are both technical and business verifications, which range from the feasibility of RFID infrastructure, such as the usability of IC tag and radio frequency signal reader, the accuracy of identification [2], and the estimation of RFID impact on the real business [3].

Through trial phase, benefits and feasibility of RFID infrastructure has been confirmed, and as a result, major retailers, such as Wal-Mart, Target, and Metro, have decide to deploy RFID implementation. At the same time these trials reveal the necessity of RFID standards. RFID standards are beneficial in many reasons. Major benefits are: 1) guarantee of interoperability, 2) acceleration of implementation, 3) reduction in cost of RFID components.

The RFID system is more beneficial when it is used among companies. To guarantee interoperability, standards are necessary. Without standards, IC tags of a manufacturer may not be detected by readers made by other manufacturers. This problem is not limited to the relation between IC tags and readers but occurs between other components as well, such as information handling between readers and systems, data stored in IC tags, and information exchange within the RFID system.

In order to deploy standards, standard organizations not only define standard specifications but also provide know-how by publishing implementation guidelines and organizing conferences to share success stories about implementation. With these efforts, companies can accelerate RFID system implementation with fewer troubles.

Reduction in cost of RFID components is a secondary benefit from RFID standard deployment. If a standard is defined and deployed to many companies, economy of scale works and, as a result, manufacturers of each component can achieve lower cost, and that benefits companies that implement RFID systems.

Among the RFID standards, the Auto-ID standard, which was originally developed by Auto-ID Center and is managed by EPCglobal, is a strong candidate to be a de-facto standard. Auto-ID Center was a federation of research universities that was originally founded in 1999 and is now

Auto-ID Lab. In Auto-ID Lab there are six universities: Massachusetts Institute of Technology (MIT), University of Cambridge, University of Adelaide, Keio University, Fudan University, and University of St. Gallen [4]. EPCglobal is a non-profit organization, and its goal is to realize global, multi-industry adoption and implementation of the EPCglobal Network [5].

The EPCglobal Network is by definition a set of technologies that enable immediate, automatic identification and sharing of information on items in the supply chain [6]. EPCglobal Network consists of software architecture components and data standards. Software architecture components also consist of four components: readers, Savant, EPC Information Server (EPC-IS), and Object Name Service (ONS). On the other hand, data standards consist of two components: Electronic Product Code (EPC) and Physical Markup Language (PML). In this thesis, we use the EPCglobal Network and Auto-ID technologies to mean the same thing.

Readers are devices for detecting tags when tags enter the read range. EPCglobal has published the Auto-ID Reader Protocol Specification. This specification defines interactions between readers and other Auto-ID compliant software components, such as Savant [7].

Savant is middleware software designed to control information flow between readers and EPC-IS. EPCglobal has published the Savant Specification. This specification defines the functions and interface of Savant. Data detected by readers is filtered, aggregated, counted and reduced by Savant. Savant also controls information flow between readers and the enterprise applications [8].

EPC-IS is an interface standard to define how to update and retrieve data of physical objects stored in servers. Types of data stored in the servers are product-level property data such as the size of physical objects, instance-level property data such as shelf life, and historical data such as the detected record of physical objects. EPCglobal has not published specifications for EPC-IS [9].

ONS provides global look-up service to the Auto-ID compliant software components. The function is similar to Domain Name Service (DNS) in the Internet. EPCglobal has published the Object Name Service Specification [10].

EPC is the identification schema for physical objects. EPCglobal has published EPC Tag Data Specification. In the specification, in addition to EPC, Filter Value and encoding and decoding rules are specified [11].

PML is an eXtensible Markup Language (XML) vocabulary set, which is used to describe all the information exchanged in the Auto-ID infrastructure. PML consists of two parts: PML Core and PML Extension. PML Core is specified in the Physical Markup Language Specification Core [12].

These are the components defined by EPCglobal. In order to fulfill the benefits of Auto-ID technologies, information about physical objects needs to be exchanged effectively within the Auto-ID infrastructure, and a key component to realize this information exchange is EPC-IS as an interface for servers that store physical object information. However, currently the specification of EPC-IS has not been published yet.

There are two reasons why the specification is not published. One reason is that Auto-ID trials

focus on real-time applications. If companies implement an application that needs historical data of physical objects, knowing how to use EPC-IS is necessary. The other reason is that Auto-ID implementation is still on a trial basis and the priority of interoperability within the Auto-ID infrastructure is still not so high. If companies need to exchange more information about physical objects, the necessity of a standard method of using EPC-IS becomes more crucial.

As we describe above, there is a gap between fundamental expectations towards the Auto-ID infrastructure and the current publication of standards. In this thesis, we would like to fill the gap by proposing a solution: architecture that can exchange information with EPC-IS.

This thesis is laid out as follows: Chapter 2 reviews the background of this thesis, such as Auto-ID standards. Chapter 3 explains objectives of the thesis and discusses assumptions used in the thesis. We also introduce the methodology employed in the thesis. Chapter 4 defines the requirements for the model we propose, and the following chapter models solutions. Chapter 6 evaluates the model with problems that are currently revealed and possible future scenarios and makes it more robust. Chapter 7 concludes our proposal and discusses further research questions that need to be addressed.

2. Background

Since we will propose architecture of EPC-IS in this thesis, components, functions and situation of the Auto-ID standard is necessary as background information. In addition to that, we review research that has been done about EPC-IS. At the same time, we introduce Business to Business (B2B) standards as a relevant technology in this chapter.

2.1. Auto-ID Standard

2.1.1. Abstract

The Auto-ID standard is a set of RFID standards which were originally researched and published by Auto-ID Center and are maintained by EPCglobal now. Auto-ID Center, which changed its name to Auto-ID Lab in November 2003, is a federation of six universities worldwide: MIT, University of Cambridge, University of Adelaide, Keio University, Fudan University, and University of St. Gallen. On the other hand, EPCglobal is a non-profit organization entrusted to drive global, multi-industry adoption and implementation of the EPCglobal Network and is a joint venture of Universal Code Council (UCC) and EAN International, which are also non-profit organizations to manage barcode, Electronic Data Interchange (EDI) standards, and B2B integration standards [13].

The EPCglobal Network is a set of technologies that enable immediate, automatic identification and sharing of information on items in the supply chain. EPCglobal Network, which is equivalent to Auto-ID technologies, consists of software architecture components and data standards. The software architecture components consist of four components: readers, Savant, EPC-IS, and ONS, and the data standards consist of two components: EPC and PML.

2.1.2. Readers

Readers are the devices which read and write the data stored in the Auto-ID compliant IC tags. Functions and interface of the readers are defined in two Auto-ID standards: Reader Protocol, which defines the upstream wire part of the readers and Radio Frequency Communication Protocol, which defines the downstream wireless part of the readers.

Reader Protocol defines three layers: Reader Layer, Message Layer, and Transportation Layer. Reader Layer is the layer that defines the content and format of the message exchanged between readers and Savant. Reader Layer defines commands sent from Savant and functions that readers should perform when commands are sent to readers, such as IC tag read trigger, filter, and send messages to IC tags.

Message Layer is the layer that defines how command messages defined in Reader Layer are formatted, transformed, and carried on a specific network transport, and Transportation Layer is the layer that defines transport facilities provided by the operating system or equivalent.

In the Reader Protocol specification, Message Layer and Transportation Layer are defined as Message Transportation Binding (MTB). Reader Protocol defines commands sent from Savant and MTB.

Regarding Radio Frequency Communication Protocol, there are three communication protocols defined in Auto-ID standards: 900 MHz Class 0 Radio Frequency (RF) Identification Tag Specification, 13.56 MHz ISM Band Class I Radio Frequency Identification Tag Interface Specification, and 860MHz - 930 MHz Class 1 Radio Frequency (RF) Identification Tag Radio Frequency & Logical Communication Interface Specification [14] [15] [16].

Each of the communication protocol specification defines radio signal frequency, radio signal power, shape of radio signal, message frame, command to IC tags, power of radio and so forth. It also refers to the functions of the IC tag as a corresponding device to communicate with.

2.1.3. Savant

Savant is a software system that is located between the enterprise application system and readers. The primary function of Savant is to reduce the vast amount of data read by readers; therefore, by this nature, Savant has functions to communicate with the enterprise applications and readers and to sort out data detected by readers. In addition, Savant has functions to communicate with ONS and EPC-IS as well. The specification of Savant is defined in Savant Specification.

Savant Specification comprises two parts: specification of Processing Modules, which defines functions which reside in Savant, and interface with other Auto-ID components. In Savant Specification version 1.0, only interface for the enterprise applications is defined.

Processing Modules define functions of Savant. Processing Modules are not only defined by EPCglobal but also by users. The combination of Processing Modules is arbitrary, and users can choose whatever they need to fulfill their business goals except for the Standard Processing Module. The Standard Processing Module, which defines minimal functions of Savant, is mandatory.

The interface specification in Savant Specification defines message structure, MTB, and the transportation protocol that Savant premises.

2.1.4. EPC-IS

EPC-IS is an interface that defines protocol to access to servers which store physical object data. The data is exchanged in PML format.

Servers for physical object information store two types of data: historical data and property data. Historical data is event records sent by Savant, and property data is a set of relevant

information about physical objects. Property data consists of the product-level property data and the instance-level property data. Examples of the product-level property data are the size and the manufacturer of the physical object, whereas examples of the instance-level property data are the expiration date and the purchase order document number for the physical object.

Specification of EPC-IS has not been published, but a couple of studies have been done for EPC-IS. We introduce them in the following section along with the history of EPC-IS.

2.1.5. ONS

ONS is a global lookup service that connects EPC with one or more Uniform Resource Identifiers (URIs) where more information about the physical object is stored. The location may be servers with EPC-IS, web sites, and other Internet resources. The specification of ONS is defined in ONS Specification.

ONS Specification defines the flow of lookup, method to convert EPC to URI format, and message format of the lookup query. ONS uses the same method that Domain Name Service (DNS) uses. In DNS, URIs and IP addresses are linked, whereas URIs and EPC are linked in ONS.

2.1.6. EPC

EPC is the fundamental identifier for physical objects. The specification of EPC is defined in EPC Tag Data Specification. EPC Tag Data Specification defines the format of EPC, and the encode/decode scheme with other industry identifiers, such as Serialized Global Trade Identification Number (SGTIN) and Serial Shipping Container Code (SSCC). All the encode/decode schemes are specified with the header value.

The number of the digits defined for an identifier is 96 digits, but, in order to meet industry requirements to collect the tag data effectively, EPCglobal defines Standard EPC Tag Data, which consists of EPC and optional Filter Value.

2.1.7. PML

PML is an eXtensible Markup Language (XML) vocabulary set, which is used to describe all the information exchanged in the Auto-ID infrastructure. PML consists of two parts: PML Core and PML Extension.

PML Core is for the common vocabulary that is commonly used in different industries and that describes basic attributes in the Auto-ID infrastructure, whereas PML Extension is for the industry specific vocabulary set. PML Core is defined in PML Core Specification by EPCglobal, but PML Extension may not be defined by EPCglobal but by other industry initiatives.

PML Core Specification defines the range of the specification, XML Schema for PML Core. PML Core schema describes EPC, date time stamp of event, EPC of the reader, and the related data, such as types of read command.

2.2. Research of EPC-IS

2.2.1. History of EPC-IS

The necessity of EPC-IS has been recognized since the initial stage of the Auto-ID technology development because one of the primary concepts of Auto-ID technologies is to store minimal data in the IC tag and to store other related data in the servers in the Auto-ID infrastructure. In order to realize this concept, PML has been developed as the common language to exchange information about physical objects, and a database to store the related information of physical objects has been proposed [17].

Initially this proposed database was called PML Server. However, as the study went on, researchers realized that related information might not be stored only in the PML Server and that it might be stored in the other enterprise systems, such as the enterprise resource planning (ERP) system. Based on this observation, researchers became focused on the interface of the database, and the interface is called EPC-IS. Until now, we have used the same term and studies have been done to better define the interface.

2.2.2. Research review on EPC-IS

Studies of EPC-IS have difficulty by nature because they are affected by industries to which EPC-IS is applied but to define the common use cases among different industries is not easy. The relatively lower necessity of EPC-IS standards also affects the research in this field. As written in Chapter 1, the publication of EPC-IS has been late because initial trials are focusing on applications which use real-time data, the data stored in the server can only be used with a proprietary method, and interoperability is not a high priority in the trials. Of course, the difficulty to define common use cases among industries is the main reason for the delay.

However, a couple of studies have been done by Cambridge University, one of the universities in the Auto-ID Lab. Approaches taken in the studies are to analyze characteristics of the data which is supposed to be stored in the servers in the Auto-ID infrastructure and also to define how to use the data [18].

The studies reveal that servers store two categories of data: the historical data and the property data. The property data is also divided into the product-level property data and the instance-level property data. This data type definition helps researchers to categorize use cases of each data type, and from this categorization they analyze efficient methods to retrieve data stored in the servers.

One fundamental assumption they make is that EPC-IS is different in different industries.

Based on this assumption, their focus is to provide technology components, such as query methods and database schemas for EPC related data, and to assess how proposed components are compatible with the existing industry standards.

2.2.3. Approach of this thesis

In this thesis, we take a different approach from the studies done by Cambridge University. We use the same data type category that they define: historical data and property data (product-level and instance-level). We also define use case as generic business processes, which are explained in Chapter 4. The difference is how to achieve compatibility with the existing industry standards. Our approach is to propose information flow architecture with compatibility in the component level, which is defined in a dictionary as explained in Chapter 5.

The benefit of this approach is that this information flow architecture can be used as a base model of new industry standards and that each industry can develop its standard by modifying it. The drawback of this approach is that it may be difficult to adopt the information flow architecture if the information flow architecture used in the industry is different from our model.

2.3. B2B standards

2.3.1. Introduction

Although EPC-IS is not used only between different companies, it shares many characteristics with B2B standards. Common components of B2B standards are messaging protocol, vocabulary, message schema, choreography, and registry/repository scheme [19]. Not all the industry standards define all the components, but they provide most of them by referring other standards. In this section, we introduce two industry B2B standards that we refer for this thesis.

2.3.2. ebXML

ebXML (Electronic Business using eXtensible Markup Language) is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes [20]. ebXML was started in 1999 as an initiative of OASIS and the United Nations/ECE agency CEFACT. It provides all the components listed in the introduction subsection. In addition to that, ebXML defines collaboration protocol agreements, which specify procedure to agree terms and condition on line.

One of the significant aspects of ebXML is the wide coverage of the components and industries. One of the drawbacks, on the other hand, is the speed of the standard defining process. This is inevitable for developing the comprehensive standard.

Another characteristic of the standard is to try to define each component separately and make it reusable. For example, business messages are not designed as a monolithic gigantic message but as a combination of a vocabulary set (ebXML calls it Core Component) and message schemas. The background technology to make ebXML define standard components separately is XML [21].

2.3.3. RosettaNet

RosettaNet is a non profit organization dedicated to creating and implementing open e-business process standards. Its focus is the high-tech industry starting from semiconductor, electronic component, information technology, and telecommunication industries. It was founded in 1999 in the United States and has eight affiliates around the world [22].

RosettaNet standard comprises messaging protocol, vocabulary (RosettaNet calls it dictionary.), message schema, and choreography. Same as ebXML, by defining message schema and dictionary separately, RosettaNet can reuse basic message structure and vocabulary and also maintain them separately. RosettaNet standard also uses XML [23].

2.3.4. Summary of B2B standard observation

From B2B industry standard observations, we conclude that it is useful to use XML to describe the interface and that to define components separately from message is a valid approach for reusable and robust interface architecture. In this thesis, we adopt these points.

3. Objective, Assumptions and Methodology

In this chapter, we describe the objective of this thesis and assumptions we make for the analysis. We also introduce the methodology employed in this thesis.

3.1. Objective

In order to fulfill the benefits aimed by the Auto-ID technologies, defining the information flow of the physical objects within the Auto-ID infrastructure is crucial. The key component to realize this information flow is EPC-IS, since EPC-IS is the common interface of the servers which store the property data and the historical data of physical objects distributed in the Auto-ID infrastructure.

However, currently specifications of EPC-IS have not been published yet, and, as a result, information flow itself and interoperability among Auto-ID components are limited and some of the business processes can not be implemented because of these constraints. Therefore, in order to fully utilize the physical object information stored in servers and realize new business processes, interface specifications are highly required.

The objective of this thesis is to propose physical object information flow architecture by defining generic businesses and a set of interface schemas that enable physical object information flow within the Auto-ID infrastructure.

3.2. Assumptions

Business requirements are necessary to start the analysis; however, some of the business requirements are contradictory. In order to make clear discussion and develop rational model, we discuss some of the business requirements that affect our analysis and make some assumptions. We identify four business requirements as issues that affect information flow within the Auto-ID infrastructure. They are: physical object type, supply chain range, server with EPC-IS existence, and existence of B2B connection. We will discuss these issues, show the possible dispositions, and make assumptions based on the discussion.

3.2.1. Physical object type

Physical object type is one of the most important business issues that need to be discussed. It is not appropriate to assume that a package of paper towels and a case of pharmaceuticals are treated in the same manner. It is conceivable that the paper towels may not be tracked by individual instance level, whereas the pharmaceuticals may require being tracked and traced by individual instance level. This is basically due to the monetary difference between paper towels and pharmaceuticals.

What is the consequence of the difference of the physical object value, then? Suppose a manufacturer ships the products to a retailer, does the manufacturer send all the individual EPCs to the retailer in the first place? The answer depends on the relation between companies, but presumably they do not exchange individual level information first. How about pharmaceuticals? The possibility to send EPC of individual items is much higher than paper towels, but it also depends on the relation.

However, if trouble, such as loss of a package, happens in shipping paper towels, how will the manufacturer and the retailer solve the problem? The retailer may request the manufacturer to send lower level shipping unit EPC, like package level EPC. In this paper towel example, they may not go down to instance level EPC, but there is a need to exchange granular level EPC. In addition, since container level is relative, we can suppose the lower level shipping unit EPC to be instance level EPC. From this observation, we assume that items need to be tracked by instance level in this thesis.

3.2.2. Supply chain range

Supply chain range is the issue of how we define supply chain starts and ends. Supply chain start can be defined from raw material, subassembly or work-in-process (WIP), or finished goods; and end can be defined as delivery to retail store, delivery to customer house and so forth. End may go beyond those and be defined considering scrapping or recycling.

However, except for combining and separating processes which make one or more products out of one or more components, business process can be defined as a collection of the physical object moves: raw material moves from the raw material supplier to the manufacturer, finished goods move from the manufacturer to the retailer, and scraps move from the customer to the wrecker.

Based on this observation, we decide to start with finished goods move from one company to another company for the requirement analysis in Chapter 4 and then model EPC-IS interface to meet the requirements. After that, we add complexity to the model in Chapter 6 by evaluating the model with the assembly process.

3.2.3. Server with EPC-IS existence

Existence of the server with EPC-IS is a fundamental assumption in this thesis, but size of the company may affect the existence of the server. It is reasonable to assume that big enterprises have servers, but it may not be true for small companies. If a small manufacturer does not have a server with EPC-IS but its trading partner requires the manufacturer to keep information of their products, what does the small manufacturer do? One possible scenario is to outsource the maintenance of physical object information to a third party. In this way, the company does not physically own a server but still keeps information of the products. Therefore, we assume the existence of servers with EPC-IS in any business entity in this thesis.

3.2.4. B2B connection

Before companies start using the Auto-ID technologies, they may have already established connections with other technologies, such as Electronic Data Interchange (EDI) or B2B Integration (B2Bi). In this case, they have a choice of using the existing connection or setting up another connection using the Auto-ID technologies. If they decide to use EDI, what they have to do is to define messages (or modify existing messages) for EPC-IS and develop mapping rules between the EDI message and the EPC-IS interface. They need to do the same thing when they decide to use B2Bi.

On the other hand, if they decide to use the Auto-ID technologies, they need to set up a new connection. In this case, if they have an existing connection, it will become redundant. It is reasonable to assume that companies abandon the redundant infrastructure or merge them with the new connection. Based on these observations, we assume that all the companies are using or will use Auto-ID technologies even if they have EDI/B2Bi connections.

3.3. Methodology

The methodology we follow in this thesis has three steps: 1) Requirement Analysis, 2) Modeling, and 3) Evaluation. In the Requirement Analysis step, we define generic business processes. One of the reasons why EPC-IS interface specifications have not been published is the lack of business processes with EPC-IS, so we define generic business processes by investigating current trials and expected business processes for RFID. After we specify these business processes, we identify the requirements for the EPC-IS interface.

In the Modeling step, we model architecture of the EPC-IS interface. Since the requirements are not comprehensive, we do not suppose that this model is used as a standard. However, this architecture of the EPC-IS interface and the ways in which it is used will be applicable to a new standard.

In the Evaluation step, we evaluate the model we propose in the previous step with problems identified in the trials and the further possible business processes. With this evaluation step, we modify the model if necessary and improve the robustness of the architecture.

4. Requirement Analysis

In this chapter, we define generic business processes so that the requirements for EPC-IS interface are derived from the business processes.

4.1. Business process selection

The business processes we choose characterize the information flow in which the servers with EPC-IS is involved. One main characteristic is the type of data that those servers store. Since servers store data for both the product-level and the instance-level, we define the business processes to deal with these two types of data. Another data type that those servers handle is historical data, which is the event record of a specific instance when it is detected by readers. We also define the business processes that utilize this historical data. We see the classification of the product-level historical data, such as changes in a component of an assembly product. However, this kind of change is also categorized as a product attribute change, which can be managed as the product-level property. Therefore, we consider only instance-level for the historical data.

Based on the analysis above, we define four generic business processes: 1) query product-level data business process, 2) query instance-level data business process, 3) query location data business process, and 4) query path data business process.

Regarding the system layout, we premise the structure in Figure 4-1-1. We assume two companies, and the difference is indicated by the suffix. We also assume that ONS is public so that both companies can use the ONS equally. Regarding Savant, we assume two layers. Savant1 and Savant2 are connected to servers with EPC-IS (we represent these servers as EPC-IS in the system layout) and the enterprise application, and Savant1-1 ~ Savant1-3 and Savant2-1 ~ Savant2-3 are connected to readers and Savant1, Savant2 respectively. EPC-IS1 stores the property data of items and the historical data that is detected by Reader1-1 ~ Reader1-3, whereas EPC-IS2 stores the historical data that is detected by Reader2-1 ~ Reader2-3. The enterprise application controls business transaction data, such as purchase orders and advance ship notices, and also provides the interface for business process operations.

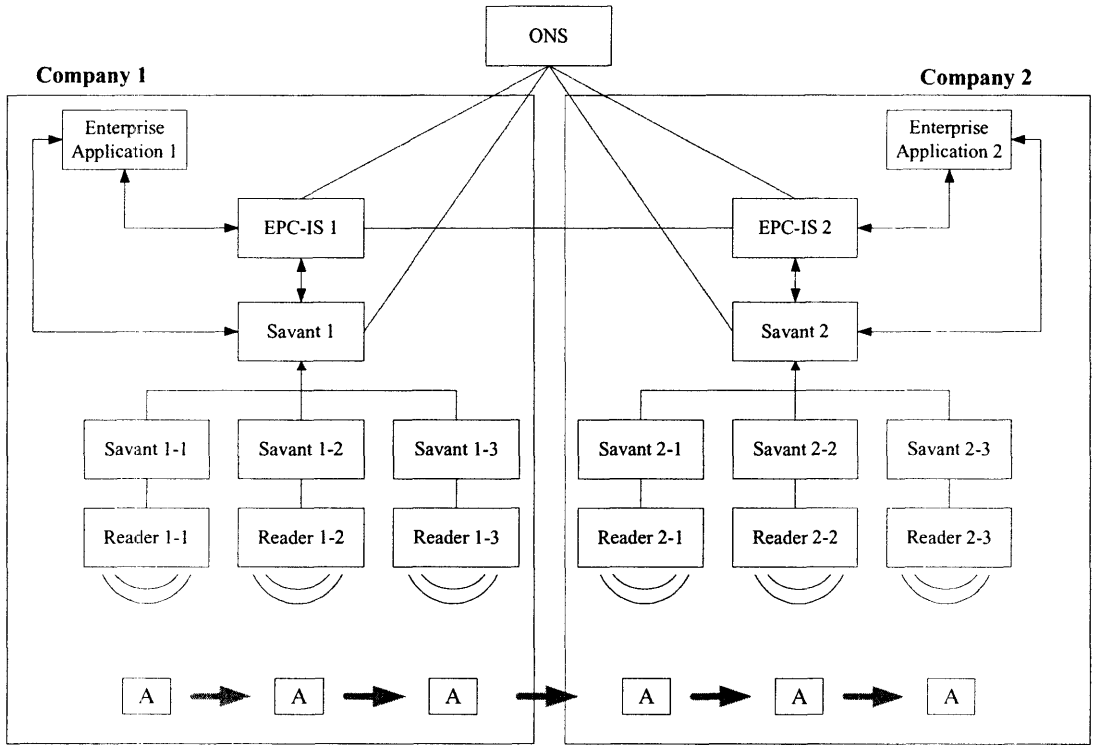

Figure 4-1-1: System layout for the generic business processes

4.2. Query product-level data business process

4.2.1. Overview

Query product-level data business process is the business process that is executed when Company2 queries the product-level data of a product to Company1's server (with EPC-IS) which stores the data of products. Preconditions of this business process are that these companies settle terms and conditions about this business process[1], original product data is stored in the Enterprise Application1, and Company1 and Company2 implement the Auto-ID infrastructure.

4.2.2. Business process flow

Figure 4-2-1 shows the flow of the business process. Each arrow represents the message sent between the system components.

[1] Articles that companies in the business process need to settle before the transactions are: the way to exchange business documents except for the messages we deal with, such as purchase order; the location of each Auto-ID compliant readers; and the technical specifications that are necessary to connect servers with EPC-IS, such as IP address.

1. Register product information: Original product-level data stored in the Enterprise Application1 is sent to the EPC-IS1. When the data is registered successfully, an acknowledgement message returns to the Enterprise Application1.

2. Register EPC and the server to the ONS: The relation between the product-level EPC and the EPC-IS1 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the enterprise application.

3. Send EPC to a server of the trading partner: After the product-level information is registered in the EPC-IS1, the data is sent to the EPC-IS2. Timing of the data update and the attributes of the product data may be defined in terms and conditions. When the data is sent successfully, an acknowledgement message returns to the EPC-IS1.

4. Send EPC from the EPC-IS2 to the Enterprise Application2: After the step 3, the EPC-IS2 sends the EPC to the Enterprise Application2. When the data is sent successfully, an acknowledge message returns to the EPC-IS2. The data that the Enterprise Application2 receives depends on the data sent from the EPC-IS1, but the Enterprise Application2 can get the EPC at least. This message can be asynchronous to the step 1 and 3. In this business process, we assume that sending a message from the EPC-IS2 to the Enterprise Application2, but this assumption is equivalent to the procedure that the Enterprise Application2 queries the EPC to the EPC-IS2.

5. Query the product-level data from the Enterprise Application2 to the EPC-IS2: After the step 4, the Enterprise Application2 has the product-level EPC of a specific product. Suppose an employee wants to know the size of the product, he operates the Enterprise Application2 and the Enterprise Application2 sends a query to the EPC-IS2.

6. EPC-IS look-up: After the step 5, the EPC-IS2 looks up the location of servers which store the data of a specific EPC by sending a query to the ONS. In this case, the EPC-IS2 gets the location information of the EPC-IS1.

7. Query the product-level data from the EPC-IS2 to the EPC-IS1: After the step 6, the EPC-IS2 recognizes the location of the EPC-IS1. Then the EPC-IS2 queries the product-level data of a specific product to the EPC-IS1 and gets the necessary information.

8. Response from the EPC-IS2 to the Enterprise Application2: In response to the step 5, the EPC-IS2 sends a response back to the Enterprise Application2. The response contains the data that the EPC-IS2 receives in the step 7.

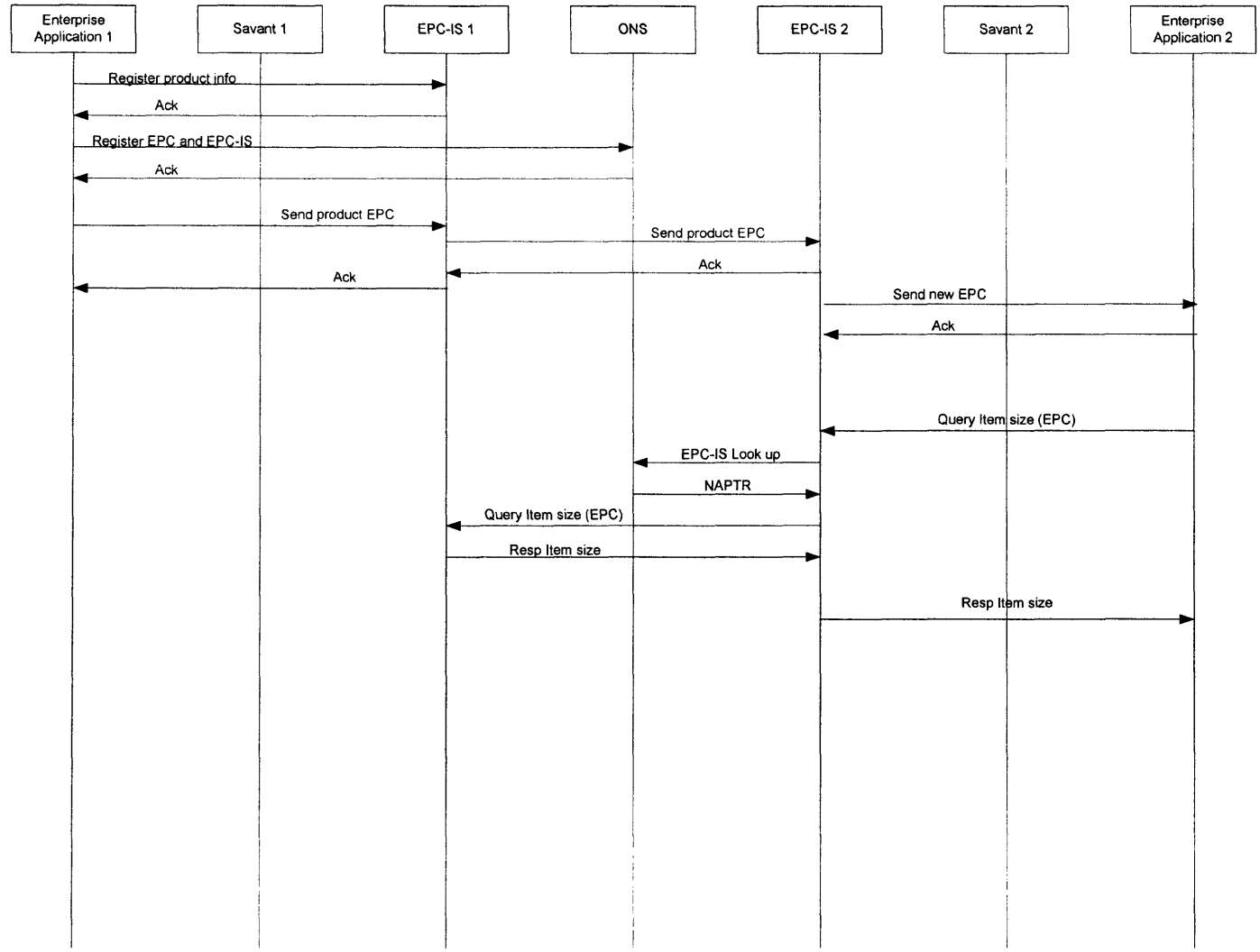

Figure 4-2-1: Work flow of the product-level information query

4.3. Query instance-level data business process

4.3.1. Overview

Query instance-level data business process is the business process that is executed when Company2 queries the instance-level data of a product to Company1's server (with EPC-IS) which stores the instance-level property data. Preconditions of this business process are that these companies settle terms and conditions about this business process, original product data is stored in the Enterprise Application1, Company2 has already received the product level data and placed an order to Company1, and they implement the Auto-ID infrastructure.

4.3.2. Business process flow

Figure 4-3-1 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Send the instance-level data with the EPC to the EPC-IS1: When each instance is scanned and the relevant data is inputted from one of the Savants, the data is sent to the EPC-IS1 from the Savant1. When the data is sent successfully, an acknowledge message returns to the Savant1.
2. Register the EPC and the server to the ONS: The relation between the instance-level EPC and the EPC-IS1 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the Savant1.
3. Send order information (order level) from the Enterprise Application1 to the Savant1: Based on the order Company1 received from Company2, the Enterprise Application1 sends an identifier of the purchase order, the product-level EPC and the quantity to the Savant1.[2] When the data is sent successfully, an acknowledge message returns to the Enterprise Application1.
4. Send order information (instance level) from the Savant1 to the EPC-IS1: After the step 3, each order is linked with the individual EPC. Then the individual EPC with the relevant information (e.g., order identifier) is sent to the EPC-IS1 and registered. When the data is sent successfully, an acknowledge message returns to the Savant1.
5. Query shipping information from the Enterprise Application1 to the EPC-IS1: After the step 4, the Enterprise Application1 queries shipping information (EPC of item, case, pallet etc.) to send an advance ship notice to Company2. The key of the query is

[2] We assume that Company2 assigns purchase order identifier and Company1 uses the same identifier. In the real practice, companies that receive purchase orders from their trading partner assign their own identifier, but they also maintain transform tables that link receiver's identifier with sender's identifier. We also assume that Company1 sends purchase order identifier that Company2 assigns when Company1 sends any messages related to the original purchase order.

the order identifier that the Enterprise Application1 sends to the Savant1 previously.

6. Send an advance ship notice to Company2: After the step 5, the Enterprise Application1 sends an advance ship notice to Company2. We assume the Enterprise Application1 sends an advance ship notice to the EPC-IS1 and then the EPC-IS1 sends an advance ship notice to the EPC-IS2, but this information may be sent via EDI/B2Bi connection if they have a connection. We also assume the EPC-IS2 pushes an advance ship notice to the Enterprise Application2, but the Enterprise Application2 may query information to the EPC-IS2.

7. Query the instance-level data from the Enterprise Application2 to the EPC-IS2: After the step 6, the Enterprise Application2 has the instance-level EPC of the items that Company2 orders. Suppose an employee wants to know the expiration date of the individual item, he operates the Enterprise Application2 and the Enterprise Application2 sends a query to the EPC-IS2.

8. EPC-IS2 look-up: After the step 7, the EPC-IS2 looks up the location of servers which store the data of a specific EPC by sending a query to the ONS. In this case, the EPC-IS2 gets the location information of the EPC-IS1.

9. Query the instance-level data from the EPC-IS2 to the EPC-IS1: After the step 8, the EPC-IS2 recognizes the location of the EPC-IS1. Then the EPC-IS2 queries the instance-level data of a specific individual item to the EPC-IS1 and gets the information.

10. Response from the EPC-IS2 to the Enterprise Application2: In response to the step 7, the EPC-IS2 sends a response back to the Enterprise Application2. The response contains the data that the EPC-IS2 receives in the step 9.

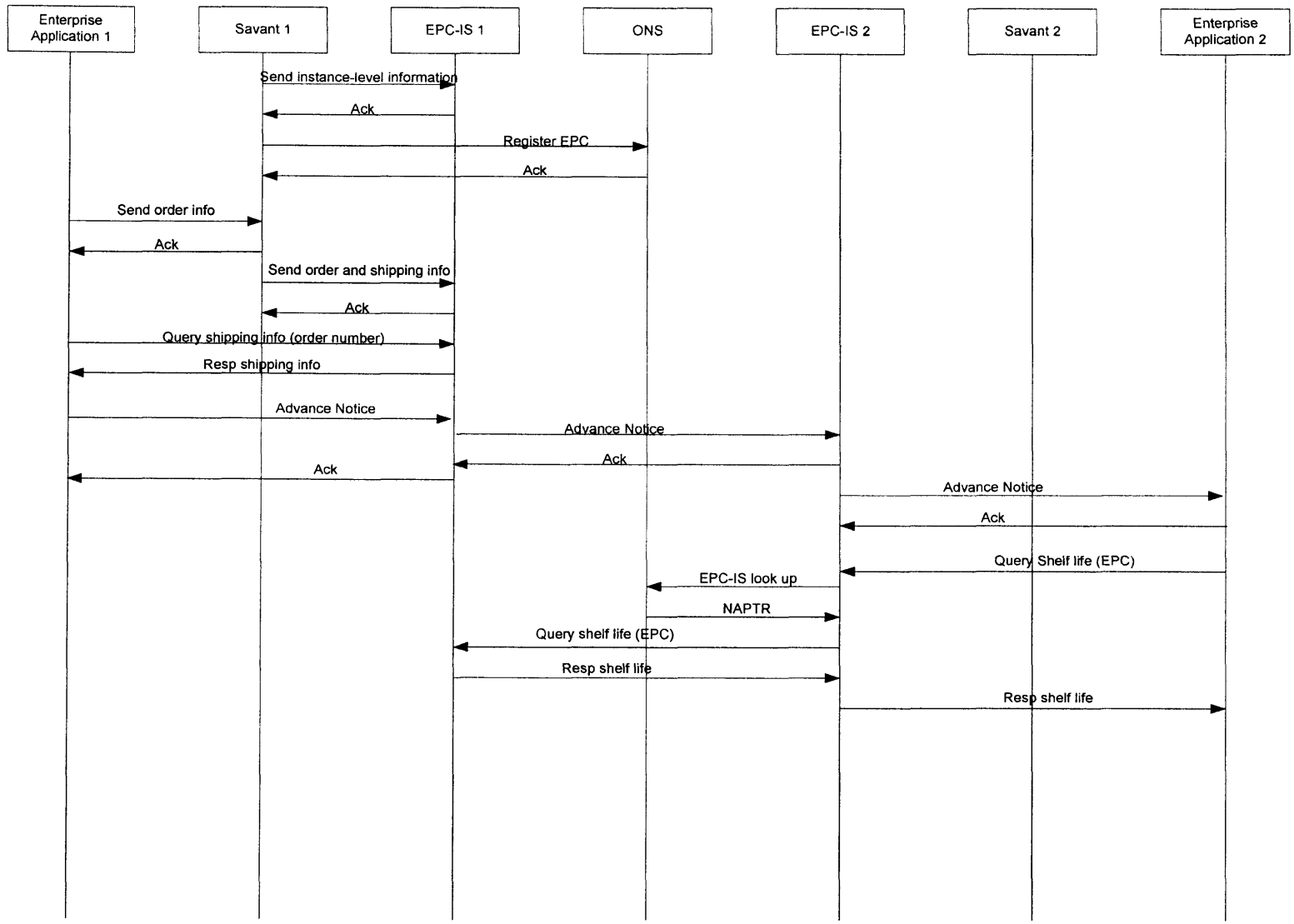

Figure 4-3-1: Work flow of the instance-level information query

4.4. Query location data business process

4.4.1. Overview

Query location data business process is the business process that is executed when Company2 queries current location data of an individual item to the server which stores location data. Location data is stored in the server of Company1, Company2 and the party which handles the item and stores the data. Preconditions of this business process are that these companies settle terms and conditions about this business process, original product data is stored in the Enterprise Application1, Company2 has already received product level data and placed an order to Company1, and they implement the Auto-ID infrastructure.

4.4.2. Business process flow

Figure 4-4-1 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Send the instance-level data with the EPC to the EPC-IS1: When each instance is scanned and the relevant data is inputted from one of the Savants, the data is sent to the EPC-IS1 from the Savant1. When the data is sent successfully, an acknowledge message returns to the Savant1.
2. Register the EPC and the server to the ONS: The relation between the instance-level EPC and the EPC-IS1 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the Savant1.
3. Send order information (order level) from the Enterprise Application1 to the Savant1: Based on the order Company1 received from Company2, the Enterprise Application1 sends an identifier of the purchase order, the product-level EPC and the quantity to the Savant1. When the data is sent successfully, an acknowledge message returns to the Enterprise Application1.
4. Send the order information (instance level) from the Savant1 to the EPC-IS1: After the step 3, each order is linked to the individual EPC. Then the individual EPC with the relevant information (e.g., order identifier) is sent to the EPC-IS1 and registered. When the data is sent successfully, an acknowledge message returns to the Savant1.
5. Query shipping information from the Enterprise Application1 to the EPC-IS1: After the step 4, the Enterprise Application1 queries shipping information (EPC of item, case, pallet etc.) to send an advance ship notice to Company2. The key of the query is the order identifier that the Enterprise Application1 sends to the Savant1 previously.
6. Send an advance ship notice to Company2: After the step 5, the Enterprise Application1 sends an advance ship notice to Company2. We assume the Enterprise

Application1 sends an advance ship notice to the EPC-IS1 and then the EPC-IS1 sends an advance ship notice to the EPC-IS2, but this information may be sent via EDI/B2Bi connection if they have a connection. We also assume the EPC-IS2 pushes an advance ship notice to the Enterprise Application2, but the Enterprise Application2 may query information to the EPC-IS2.

7. Send the detected data from the Savant1 to the EPC-IS1: When the items are shipped, they are scanned by the readers and the data is sent from the Savant1 to the EPC-IS1. In this sample process, we assume the Savant1-1 is located at Company1's plant and the Savant1-2 and the Savant1-3 are located at the distribution center of Company1. When they are scanned, the data is first sent from the Savant1-2 to the Savant1, and then the Savant1 sends the data to the EPC-IS1. When the data is sent successfully, an acknowledge message returns to the Sanvant1.

8. Query the instance-level data from the Enterprise Application2 to the EPC-IS2: After the step 6, the Enterprise Application2 has the instance-level EPC of the items that Company2 orders. Suppose an employee wants to know the location of the individual item, he operates the Enterprise Application2, and the Enterprise Application2 sends a query to the EPC-IS2.

9. EPC-IS2 look-up: After the step 8, the EPC-IS2 looks up the location of servers which store the data of a specific EPC by sending a query to the ONS. In this case, the EPC-IS2 gets the location information of the EPC-IS1.

10. Query the instance-level data from the EPC-IS2 to the EPC-IS1: After the step 9, the EPC-IS2 recognizes the location of the EPC-IS1. Then the EPC-IS2 queries the instance-level data (location data) of a specific individual item to the EPC-IS1 and gets the information.

11. Response from the EPC-IS2 to the Enterprise Application2: In response to the step 8, the EPC-IS2 sends a response back to the Enterprise Application2. The response contains the data that the EPC-IS2 receives in the step 10.

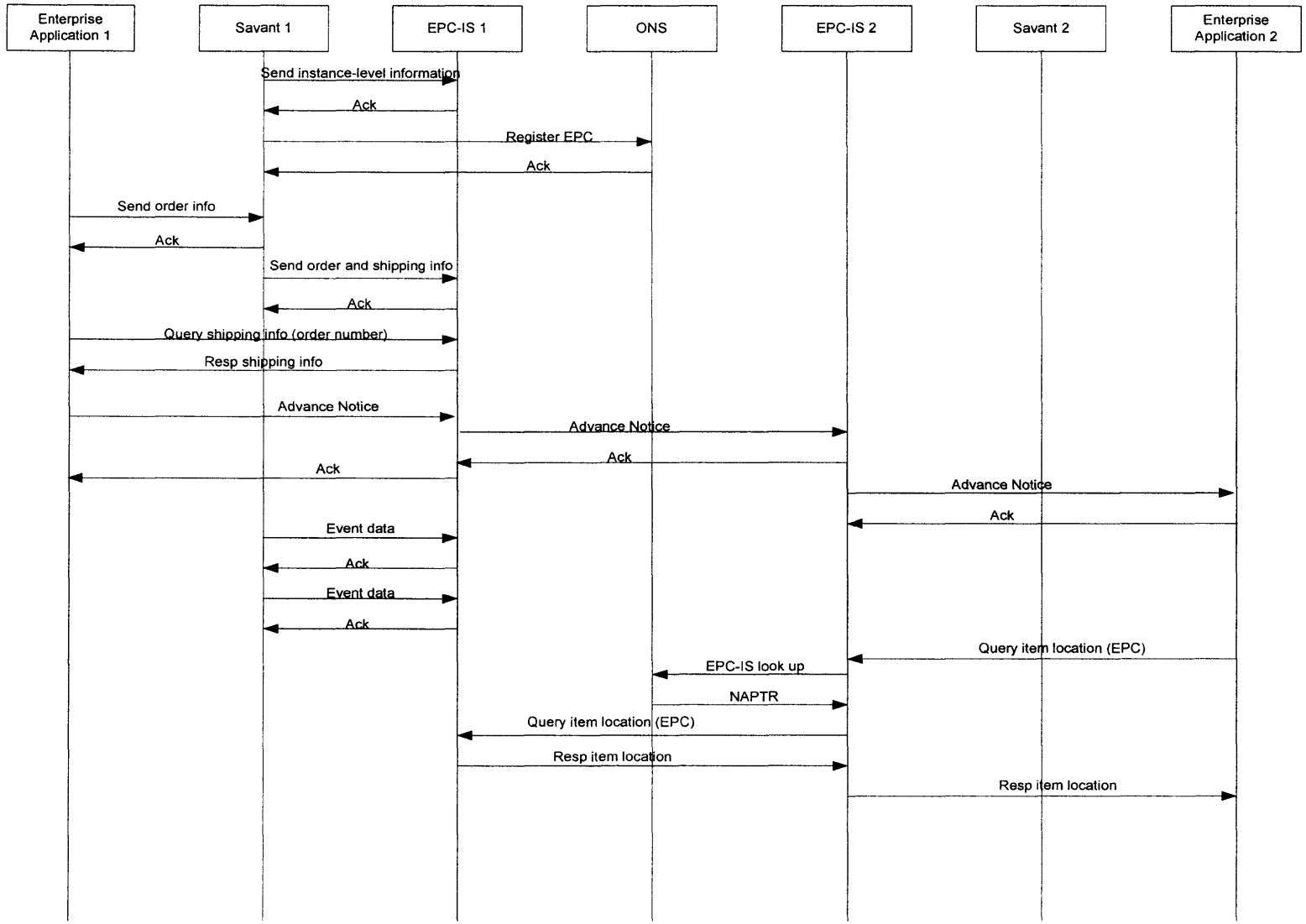

Figure 4-4-1: Work flow of the location data query

4.5. Query path data business process

4.5.1. Overview

Query path data business process is the business process that is executed when Company2 queries the path data that an individual item takes by using the EPC that is originally sent by Company1. The event data is stored by the servers of Company1, Company2 and the other parties which handle the item. With this business process, Company2 can retrieve EPC of readers that have scanned the specific instances, and, from EPCs of these readers and the location information of them, which are received by Company2 in terms and conditions, Company2 can trace the physical route that these instances have taken. Preconditions of this business process are that these companies settle terms and conditions about this business process, original product data is stored in the Enterprise Application1, Company2 has already received the product level data and placed an order to Company1, and they implement the Auto-ID infrastructure.

4.5.2. Business process flow

Figure 4-5-1 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Send the instance-level data with the EPC to the EPC-IS1: When each instance is scanned and the relevant data is inputted from one of the Savants, the data is sent to the EPC-IS1 from the Savant1. When the data is sent successfully, an acknowledge message returns to the Savant1.
2. Register the EPC and the server to the ONS: The relation between the instance-level EPC and the EPC-IS1 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the Savant1.
3. Send order information (order level) from the Enterprise Application1 to the Savant1: Based on the order Company1 received from Company2, the Enterprise Application1 sends an identifier of the purchase order, the product-level EPC and the quantity to the Savant1. When the data is sent successfully, an acknowledge message returns to the Enterprise Application1.
4. Send order information (instance level) from the Savant1 to the EPC-IS1: After the step 3, each order is linked to the individual EPC. Then the individual EPC with the relevant information (e.g., order identifier) is sent to the EPC-IS1 and registered. When the data is sent successfully, an acknowledge message returns to the Savant1.
5. Query shipping information from the Enterprise Application1 to the EPC-IS1: After the step 4, the Enterprise Application1 queries shipping information (EPC of item, case, pallet etc.) to send an advance ship notice to Company2. The key of the query is

the order identifier that the Enterprise Application1 sends to the Savant1 previously.

6. Send an advance ship notice to Company2: After the step 5, the Enterprise Application1 sends an advance ship notice to Company2. We assume the Enterprise Application1 sends an advance ship notice to the EPC-IS1 and then the EPC-IS1 sends an advance ship notice to the EPC-IS2, but this information may be sent via EDI/B2Bi connection if they have a connection. We also assume that the EPC-IS2 pushes an advance ship notice to the Enterprise Application2, but the Enterprise Application2 may query new information to the EPC-IS2.

7. Send detected data from the Savant to the server: When the items are shipped, they are scanned by the readers and the data is sent from Savant to the servers. In this sample process, we assume the Savant1-1, the Savant1-2, and the Savant1-3 are located at Company1's facility and the Savant2-1, the Savant2-2 and the Savant2-3 are located at Company2's facility. All these Savants are called edge Savant. When scanned, the data is first sent from the edge Savant to the Savant1 or the Savant2, and then each Savant sends the data to the server with EPC-IS in the facility. When the data is sent successfully, an acknowledge messages return to the Savants which send the message to each server.

8. Register the EPC and the server to the ONS: The relation between the instance-level EPC and the EPC-IS2 location is registered to the ONS at Company2 side. If the data is registered successfully, an acknowledgement message returns to the Savant2.

9. Query the instance-level data from the Enterprise Application2 to the EPC-IS2: After the step 6, the Enterprise Application2 has the instance-level EPC of the items Company2 orders. Suppose an employee wants to know the path that the individual item takes, he operates the Enterprise Application2 and the Enterprise Application2 sends a query to the EPC-IS2.

10. EPC-IS2 look-up: After the step 9, the EPC-IS2 looks up the location of severs which store the data of a specific EPC by sending a query to the ONS. In this case, the EPC-IS2 gets the location information of the EPC-IS1 and the EPC-IS2 itself.

11. Query the instance-level data from the EPC-IS2 to the EPC-IS1: After the step 10, the EPC-IS2 recognizes the location of the EPC-IS1. Then the EPC-IS2 queries the instance-level data of a specific individual item to the EPC-IS1 and gets the information.

12. Response from the EPC-IS2 to the Enterprise Application2: In response to the step 9, the EPC-IS2 sends a response back to the Enterprise Application2. The response contains both path data that the EPC-IS2 receives in the step11 and path data the EPC-IS2 stores.

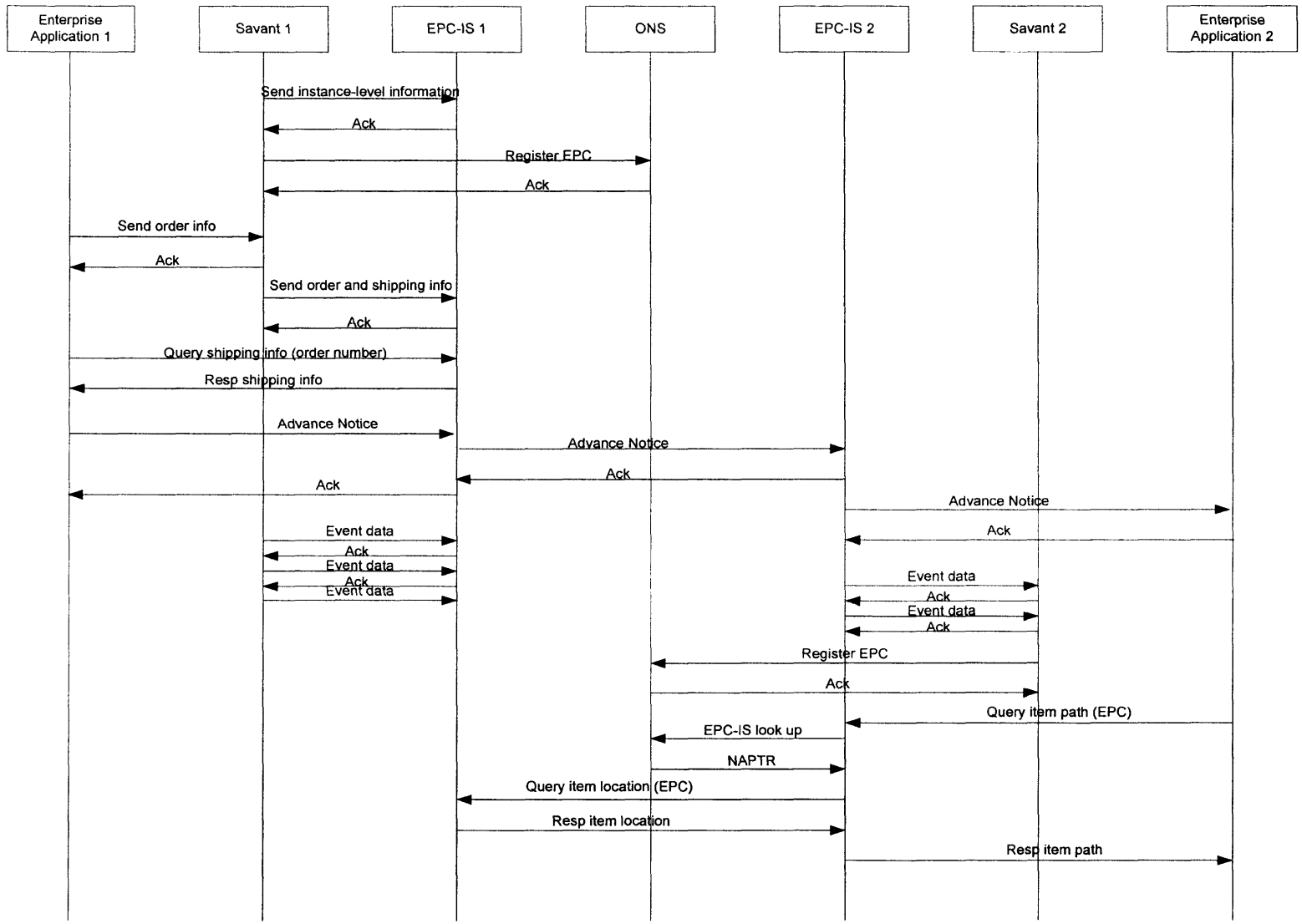

Figure 4-5-1: Work flow of the path data query

4.6. Exceptions

4.6.1. Overview

In addition to the generic business processes, we include exception business processes. We assume two types of exceptions: exceptions that happened after a system component received a notify message, and exceptions that happened after a system component received a query message. We exclude the exceptions that happen on account of the messaging protocol layer and the lower layer errors.

4.6.2. Business process flow for notify exception

Figure 4-6-1 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Register the product information: Original product-level data stored in the Enterprise Application1 is sent to the EPC-IS1. When the data is registered successfully, an acknowledgement message returns to the Enterprise Application1.
2. Register the EPC and the server to the ONS: The relation between the product-level EPC and the EPC-IS1 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the enterprise application.
3. Send the EPC to the server of the trading partner: After the product-level information is registered in the EPC-IS1, the data is sent to the EPC-IS2. Here we assume a case that the data is sent to the EPC-IS2, but the EPC-IS2 does not register the data for some reason. The EPC-IS2 sends an exception message back to the EPC-IS1, and the EPC-IS1 sends an exception message to the original requester, the Enterprise Applicaiton1.[3]

4.6.3. Business process flow for query exception

Figure 4-6-2 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Register the product information: Original product-level data stored in the Enterprise Application1 is sent to the EPC-IS1. When the data is registered successfully, an acknowledgement message returns to the Enterprise Application1.

[3] We see the necessity of another message that deals with recovering the data integrity after an exception occurs, but we exclude this issue out of the thesis's scope.

2. Register the EPC and the server to the ONS: The relation between the product-level EPC and the EPC-IS1 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the enterprise application.

3. Send the EPC to a server of the trading partner: After the product-level information is registered in the EPC-IS1, the data is sent to the EPC-IS2. Timing of the data update and the attributes of the product data may be defined in terms and conditions. When the data is sent successfully, an acknowledgement message returns to the EPC-IS1.

4. Send the EPC from the EPC-IS2 to the Enterprise Application2: After the step 3, the EPC-IS2 sends the EPC to the Enterprise Application2. When the data is sent successfully, an acknowledge message returns to the EPC-IS2. The data that the Enterprise Application2 receives depends on the data sent from the EPC-IS1, but the Enterprise Application2 can get the EPC at least. This message can be asynchronous to the step 1 and 3. In this business process, we assume that sending a message from the EPC-IS2 to the Enterprise Application2, but this assumption is equivalent to the procedure that the Enterprise Application2 queries the EPC to the EPC-IS2.

5. Query the product-level data from the Enterprise Application2 to the EPC-IS2: After the step 4, the Enterprise Application2 has the product-level EPC of a specific product. Suppose an employee wants to know the size of the product, he operates the Enterprise Application2 and the Enterprise Application2 sends a query to the EPC-IS2.

6. EPC-IS look-up: After the step 5, the EPC-IS2 looks up the location of servers which store the data of a specific EPC by sending a query to the ONS. In this case, the EPC-IS2 gets the location information of the EPC-IS1.

7. Query the product-level data from the EPC-IS2 to the EPC-IS1: After the step 6, the EPC-IS2 recognizes the location of the EPC-IS1. Then the EPC-IS2 queries the product-level data of a specific product to the EPC-IS1. Here we assume a case that the query is sent to the EPC-IS1, but the EPC-IS1 does not retrieve the data for some reason. The EPC-IS2 sends an exception response message back to the EPC-IS2, and the EPC-IS2 sends an exception response message back to the original requester, the Enterprise Applicaiton2.

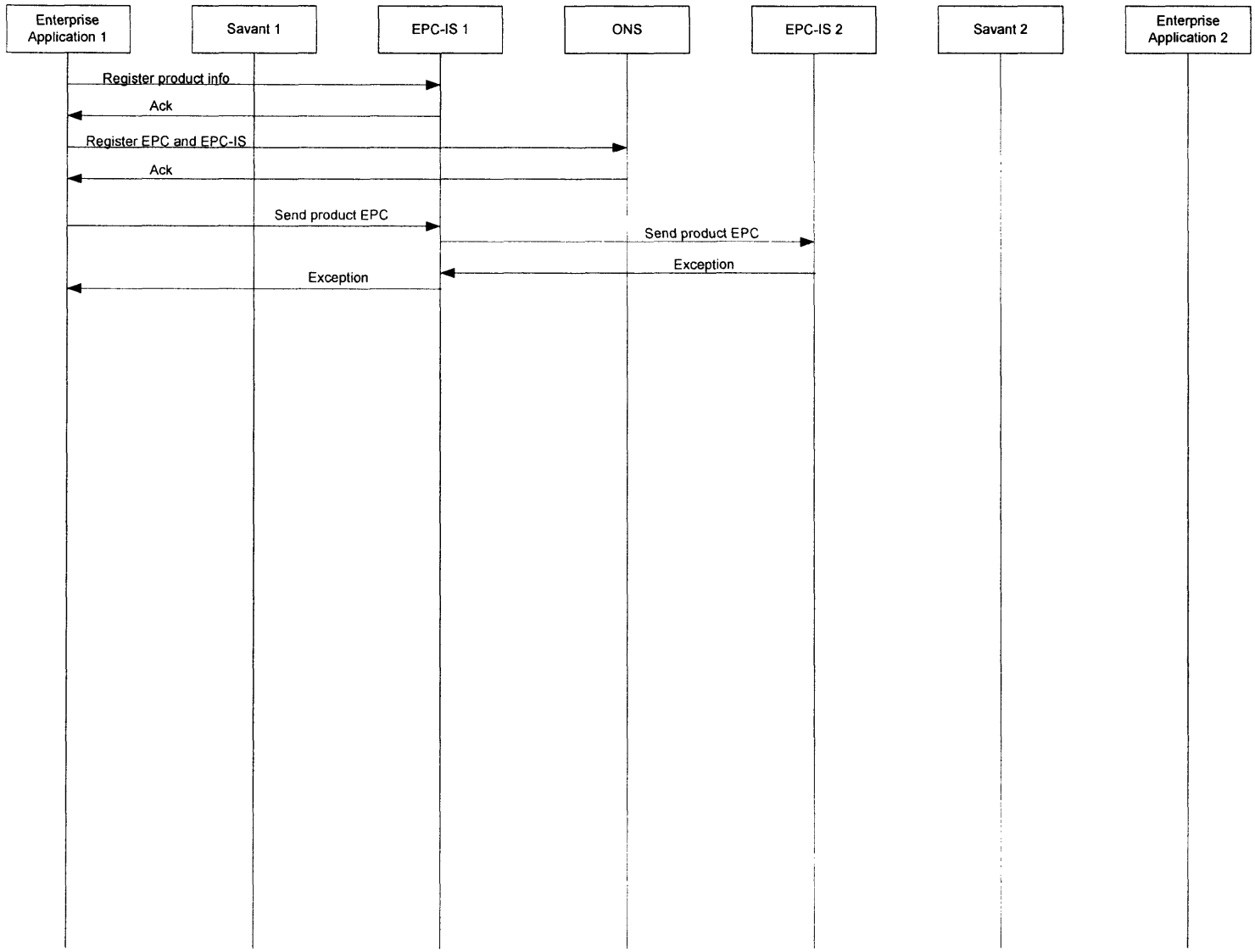

Figure 4-6-1: Work flow of the notify message exception

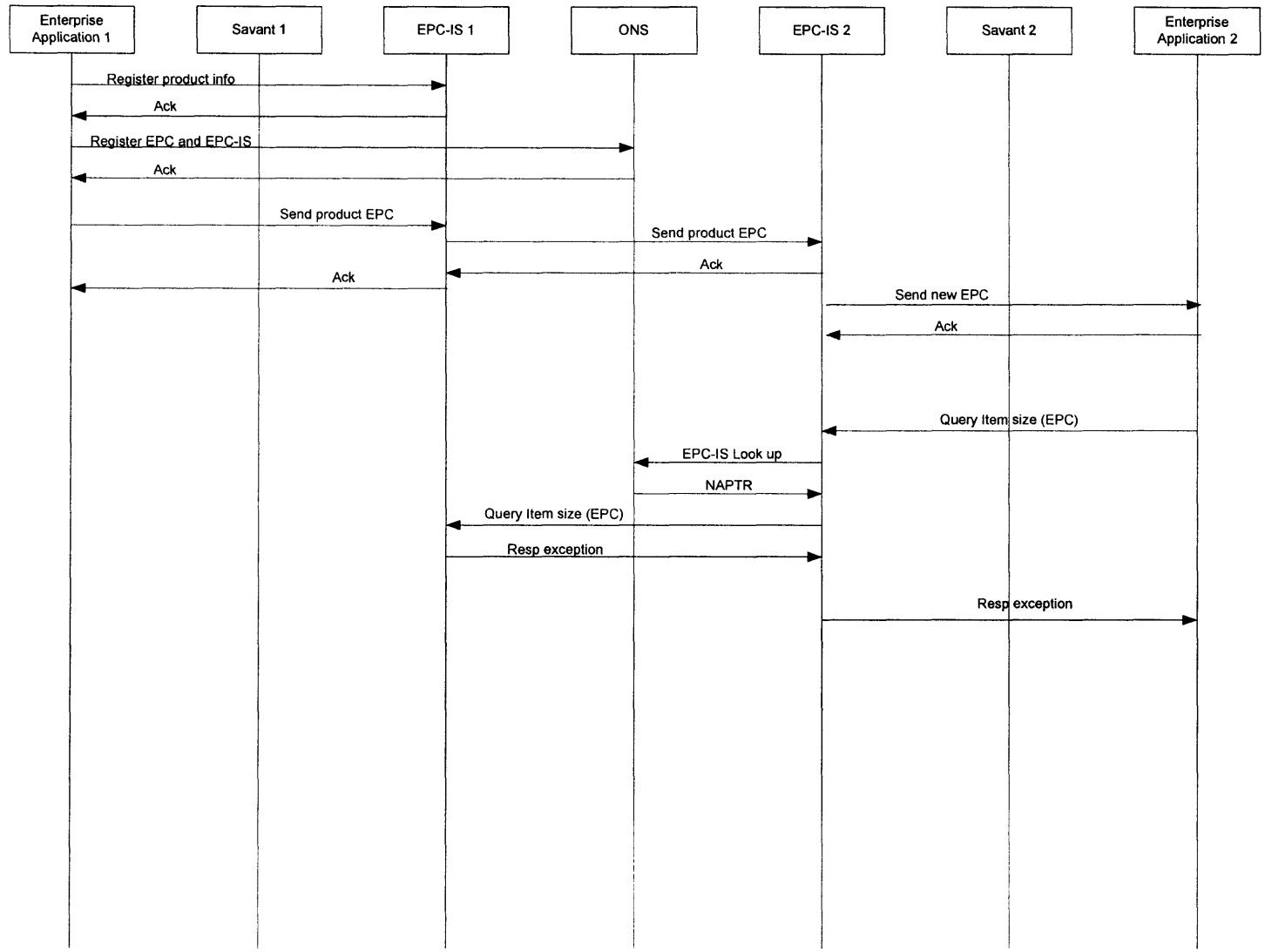

Figure 4-6-2: Work flow of the query exception

4.7. Requirement definition

From the generic business processes we defined in the previous section, we distill the requirements for EPC-IS interface.

4.7.1. Message types

From the generic business processes, we find that three types of messages are necessary for EPC-IS: Notify message, Query/Response message, and Acknowledge message. The Notify message is used when the Enterprise Application, Savant, EPC-IS or other Auto-ID compliant components create the data that is supposed to be stored in the server with EPC-IS. This message is also used when these software components ask servers with EPC-IS for transferring another Notify message with additional data stored in the servers. It is used for both property data and historical data, and can send data of multiple products and instances.

The Query message is used when the Enterprise Application, EPC-IS or other Auto-ID compliant components query the data to servers with EPC-IS, and Response message is the response to the query. The key for the query may be EPC, property data, date time stamp of historical data, and any other data defined in the message exchange.

The Acknowledge message is used to acknowledge the receipt of the Notify message. A sending system component uses the Acknowledge message to confirm that the Notify message has arrived at the receiving component. However, the use depends on the business context. The message may send either message level result or the product/instance level result.

4.7.2. Scalability of the message

Although four generic business processes are based on the RFID implementation trials and the expected business processes of RFID, they do not cover all the business processes with EPC-IS. Therefore, the model needs to have scalability to accommodate other business processes and future business processes.

4.7.3. Data type

From the generic business processes, we find that EPC-IS must be able to deal both property data and historical data, and the property data consists of product-level property data and instance-level property data.

Actions required from the property data are create, update and query, but it is conceivable that delete action is necessary as well. The historical data is stored when an event occurs to each individual item. Attributes which need to be stored are event type, EPC of the subjected item, date time stamp, and the relevant data, such as reader EPC.

4.7.4. Mechanism to translate historical data into business process

The historical data is stored in accordance with the event type. On the other hand, the business processes which utilize this historical data may not just check the existence of a specific individual item, which is directly known from the historical data. If a company wants to know the path of an individual item, some mechanism that translates the plain historical data into required the business process (e.g., query path of individual item) is necessary.

There are two types of procedures for this mechanism: one is the local procedure and the other is the remote procedure. Since functions of EPC-IS are not determined, EPC-IS interface needs to cover both procedures.

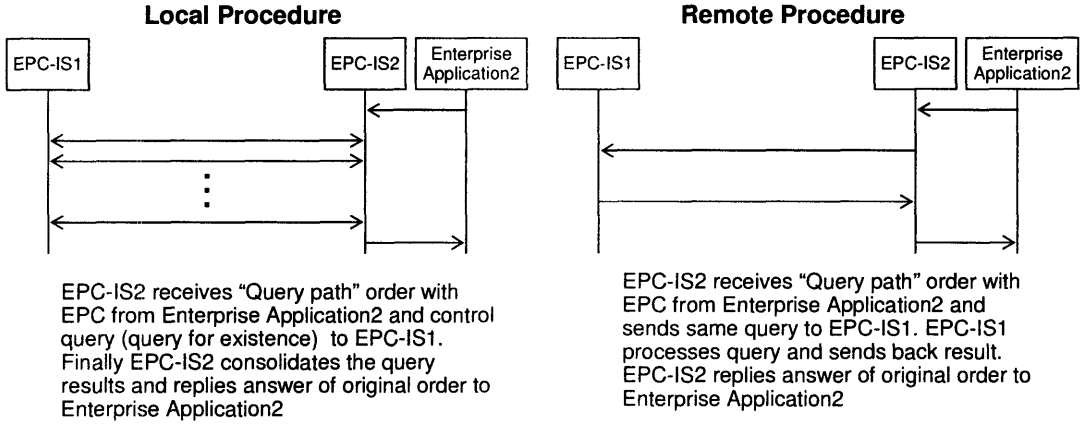

Figure 4-6-1: Types of procedure

5. Modeling

In this chapter, we develop the architecture of EPC-IS based on the requirements we define in Chapter 4. Since the generic business processes are not concrete enough to develop standard specifications, the architecture model proposed in this chapter is not used as a standard as it is. However, the fundamental structure is applicable to a new standard.

5.1. Modeling direction

From the requirement analysis, one key requirement for the modeling is scalability. There are two reasons: one is that functions of EPC-IS have not been determined, and the other is that Auto-ID technologies are applied to many industries, which may have requirements different from our assumptions. In order to guarantee the scalability, we adopt a two-layer structure for EPC-IS interface: 1) message structure and 2) dictionary structure. For the dictionary structure, we also adopt Action class, which describes the event of the historical data and the action to the property data, and Value class, which is an attribute of Action class.

5.1.1. Separation of message structure from dictionary structure

In order to guarantee the scalability, we take into account the maintenance cycle of the message structure and the components embedded in it. Generally the need for changing component requirement is more frequent than that for the message. If properties of physical objects are embedded in the message as XML tags, for example, if we send a box-shape product and model tags like <height>, <depth>, and <width> in the message, every time items of a new shape need to be sent, the entire message needs to be modified, even if fundamental message structure may not need to be modified. Therefore, we model the message structure and the components separately, and propose the way to maintain those components with the dictionary.

5.1.2. Meaning of adopting Dictionary

If the components in the messages are separately defined in the dictionary, the dictionary is a vocabulary set used to describe physical objects, which is equivalent to the PML Extension. By separating the components from the message, we propose the architecture of both EPC-IS and the PML Extension.

5.1.3. Dictionary structure

We define a schema of the dictionary by analyzing the data that needs to be stored in the dictionary. Table 5-1-1 shows the data structure for the property data. There are two types of

actions for the property data: one is for operating data stored in the servers, and the other is for asking data transmission to other servers. The actions for the data operation are "create," "update," and "delete," and the data for these actions includes EPC of the subjected item and the relevant data, such as size, expiration date, and the related business document of the item. Unit of measure and format of the value are also necessary for the property data.

On the other hand, the Action for the data transmission is "notify," and the data includes the shipment identifier, the purchase order identifier, the EPC of the shipped item, and the company profile. Details are defined in terms and conditions.

Considering the consistency of the data structure, we assume that we will use EPC for the document identifier, such as a shipment identifier and a purchase order identifier. There is no impact of this assumption for the dictionary schema. With this assumption, the format of the shipment identifier becomes EPC format, and without this assumption, it may be free form.

Table 5-1-1 Structure for the property data

Action	Key	Relevant data
Create, Update, Delete	EPC of instance, EPC of product	Size, expiration date, business document
Notify	Shipment identifier (EPC)	Purchase order identifier, EPC of shipped items, company profile

The same structure is applied to the historical data. The Actions for the historical data are events, such as "detect." The key for the historical data is EPC of the subjected individual item, and the relevant data, such as the date time stamp and the reader EPC, needs to be collected. (Table 5-1-2)

Table 5-1-2 Structure for the historical data

Action	Key	Relevant data
Detect	EPC of instance	Date time stamp, Reader EPC

From this observation, we propose the same dictionary schema structure for the property data and the historical data.

5.2. Dictionary

From what we observed, we propose the same dictionary schema for both property data and historical data. We use the UML Class Diagram for the dictionary schema design [24]. Figure

5-2-1 shows the schema for both dictionaries and Table 5-2-1 shows the description of each class. The instances of the dictionary are listed in the appendix.

Based on the data structure, we observe that each action takes more than one pieces of relevant data but that the amount of data is different in each case. Therefore, we develop Value class separate from Action class and link them with identifiers. By defining Action class as separate from Value class, we also include the mechanism of the local procedure and the remote procedure; the instance of Action class can be defined either as "query detect data" or "query path data." The relevant data, which is different in each case, can also be defined separately.

We also understand the need to aggregate Value class when a set of values has some special business meanings. For example street, city, state, zip, and country are individual values, but usually they are for specific business entities, such as the manufacturer address and the retailer address. Therefore, we develop ValueSet class and link it with Action class and Value class. We also understand that, in some cases, aggregation has multiple layers, so we add recursive structure to ValueSet class.

There are several ways to exchange units of measure among Auto-ID components, but to reduce the data size and the conversion load of each component, we propose to define unit of measure in the dictionary. This unit of measure is defined as Format class and UOM class associated with Value class.

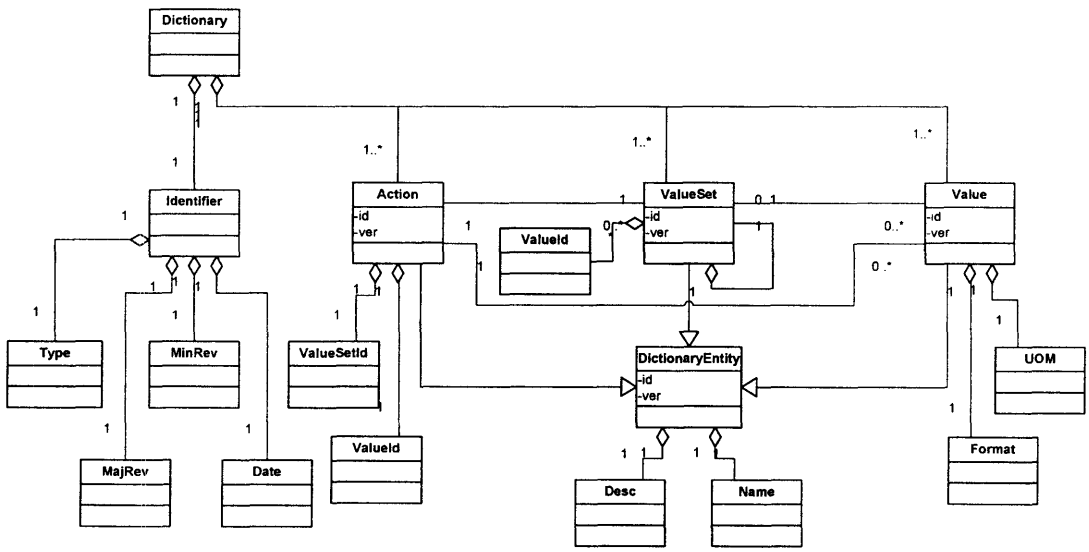

Figure 5-2-1: Class diagram of the Dictionary schema

Table 5-2-1: Class description of the Dictionary schema

Class Name	Description
Dictionary	Class for the dictionary

Class Name	Description
Identifier	Class for the dictionary identifier. Instances of this class are Event and Property.
MajRev	Class for the major dictionary revision.
MinRev	Class for the minor dictionary revision.
Date	Class for the date of issue.
DictionaryEntity	Abstract class for the classes used in the Dictionary schema.
Name	Class for the name of the Action, ValueSet, and Value class.
Desc	Class for the description of the Action, ElementSet, and Element class.
Action	Class to describe the event in the Event dictionary and the action in the Property dictionary.
ValueSetId	Class to indicate the relationship between the Action and the ValueSet class. Identifiers of the ValueSet class are the instance of this class.
ValueId	Class to indicate the relationship between the Action and the Value class. Identifiers of the Value class are the instance of this class.
ValueSet	Class to aggregate the ValueSet or the Value class. This class does not have any instances.
Value	Class to aggregate the value of the dictionary property. This class does not have any instances.
Format	Class to define the format of the property. This class may not be used in the Event dictionary.
UOM	Class to define the unit of measure of the property. This class may not be used in the Event dictionary.

5.3. Message

From what we observed in section 5.1, we propose message schemas for the Notify message, the Query/Response message, and the Acknowledge message. We use UML Class Diagram to model these schemas [25].

5.3.1. Notify message

Figure 5-3-1 shows the message structure of the Notify message, and Table 5-3-1 shows the description of each class and property.

The BusinessDocument class is an abstract class for all the messages. The Notify class is specialized from the BusinessDocument class. The relation between the Notify class and the PhysicalObject class is one to many in order for Savant to send data of multiple physical objects. The PhysicalObject class has EPC as a property. Properties of the Action class, dicRef and ver, come from the dictionary schema. By using identifier defined in the dictionaries, a sender and a receiver can identify the action of the data. The relation between the PhysicalObject class and the Action class is one to many. With this structure, Savant can send multiple events of a physical object.

The Action class has two associated classes, the ElementSet class and the Element class. The ElementSet class and the Element class correspond to the ValueSet class and the Value class in the dictionary schema, respectively. This is because some instances of the Action class are associated with instances of the Value class directly; and others are associated with instances of the ValueSets class first, and then instances of the ValueSets class are associated with instances of the Value class. Since an instance of the Action class needs at least one associated instance of either the ElementSet or the Elment class, there is a constraint between the ElementSet class and the Element class. The reason why the ElementSet class has recursive structure is the same as that of dictionary structure. With this structure, the physical address of a shipping facility of a manufacturer is modeled by defining the ship from location and the manufacturer as instances of the ValueSet class and the physical address as an instance of the Value class.

The Element class has the Value class for its associated class. The Value class takes historical data and property data as its instance. Since the instances of each class are always defined in pair in the dictionaries, the relation between the Element class and the Value class is one to one.

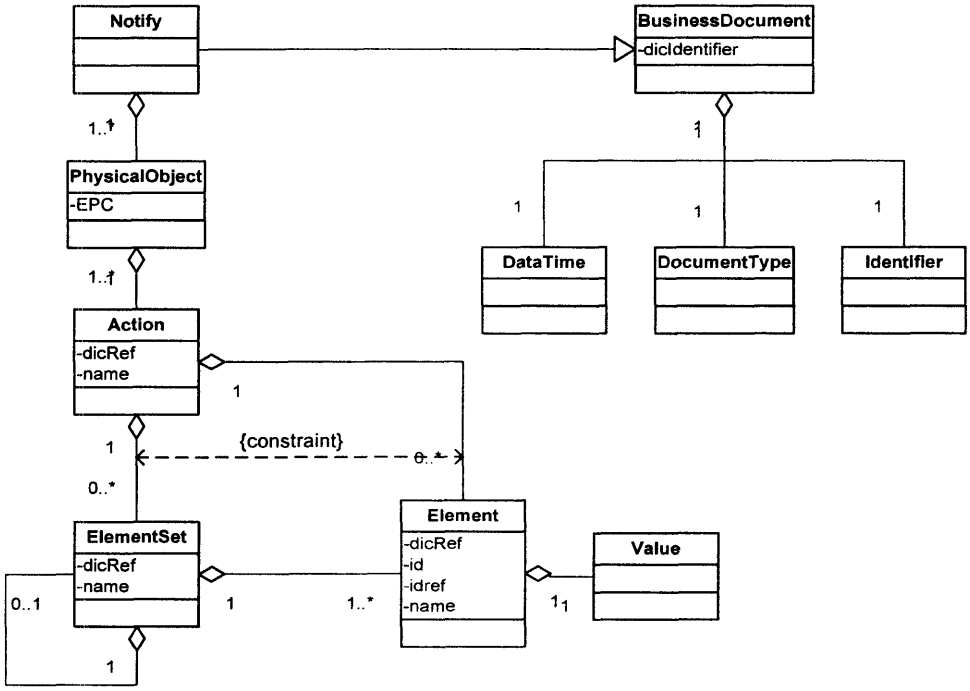

Figure 5-3-1: Class Diagram of the Notify message

Table 5-3-1: Class and property description of the Notify message

Class Name	Description
BusinessDocument	Abstract class for the business message. The Notify class, the Query/Response classes and the Acknowledge class are the generalized class of this abstract class.
DateTime	The data time stamp for the message sent.
DocumentType	Either "notify," "query," "response," or "acknowledge."
Identifier	Identifier of the business message.
Notify	Class for the Notify message.
PhysicalObject	Class for the subjected physical object. This class has one to many relation with the Notify class, which means one Notify message sends the data of multiple physical objects.
Action	Class for the action done to the physical objects, which is defined in the Action class in the dictionary. This class has one to many relation with the PhysicalObject class, which means multiple actions are sent about one physical object.
ElementSet	Class for the aggregated values of each Action class instance.

Class Name	Description
	The Action class takes either the ElementSet class or the Element class as the child class. This class does not have any instances.
Element	Class for the values of each Action class instance. The Action class takes either the ElementSet class or the Element class as the child class. This class does not have any instances.
Value	Class for the relevant information values of each Action class instance. The pair of the Name class and the Value class describes the relevant information of the Action class instance.
dicIdentifier	Property to identify the dictionary type and the version.
EPC	Property to identify the EPC of the class.
dicRef	Property to refer the Action, ValueSet, and Value class instance of the dictionary.
ver	Property to refer the version of each class.
name	Property to show the name of each dictionary instance.
id	Identifier for the instance.
idref	Identifier reference for the instance.

5.3.2. Query/Response message

Figure 5-3-3 shows the message structure of the Query/Response message, and Table 5-3-2 shows the description of each class and property.

The BusinessDocument class is an abstract class for all the messages. Both the Query and the Response class are specialized from the BusinessDocument class. The structure of the Response class is identical to the Notify class. Since the Query class describes the attributes that a query sender wants to get, the Query class does not have the PhysicalObject class as an associated class. If a sender wants to retrieve all the data that are related to one EPC, it can use the Value class. This query is enabled by the dictionary design that each instance of the Action class has an EPC as an instance of the associated class of the Action class.

There are several ways to develop instances of the Query message. Since our proposal uses XML, XML Query (XQuery) proposed at World Wide Web Consortium (W3C) may be applied [26], [27]. However, XQuery is more compatible to the XML with solid tag names than those with generic tag names like our model. Therefore, we propose a different query model in this thesis.

Simple query is constructed by sending tags with identifiers defined in the dictionaries. For example, when a retail company wants to retrieve the reader EPC by which a product is scanned, the company sends a Query message indicating the required data with tags. In the sample below, we assume that this company wants to know the EPC of readers that scan a

product whose EPC is known (urn:epc:1.10.100.2). When the trading company receives the query, it retrieves servers which store historical data and sends back the response. In this example, two readers scan the product, and EPCs of these readers (urn:epc.1.10.110.1 and urn:epc:1.10.110.2) are sent back to the retail company.

Sample Query Message

```
...
<Action dicQuery="EA001-01" name="detect">
        <Element dicQuery="EV001-01" name="EPC">
                <Value> urn:epc:1.10.100.2</Value>
        </Element>
        <Element dicQuery="EV003-01" name="ReaderEPC">
                <Value></Value>
        </Element>
</Action>
...
```

Sample Response Message

```
...
<PhysicalObject EPC="urn:epc:1.10.100.2">
        <Action dicQuery="EA001-01" name="detect">
                <Element dicQuery="EV001-01" name="EPC">
                        <Value> urn:epc:1.10.100.2</Value>
                </Element>
                <Element dicQuery="EV003-01" name="ReaderEPC">
                        <Value>urn:epc:1.10.110.1</Value>
                </Element>
        </Action>
        <Action dicQuery="EA001-01" name="detect">
                <Element dicQuery="EV001-01" name="EPC">
                        <Value> urn:epc:1.10.100.2</Value>
                </Element>
                <Element dicQuery="EV003-01" name="ReaderEPC">
                        <Value>urn:epc:1.10.110.2</Value>
                </Element>
        </Action>
</PhysicalObject>
...
```

Figure5-3-2: Sample instance of the simple query

More complex queries are also developed in the same manner. If a retail company wants to

know the data scanned during the specific time range, it fills the time range for the Value tag instance, and if the company wants to know the data scanned by the several readers, it enumerates EPCs of these readers for the Value tag instance.

Since we understand the need to send back the query result in the Response message, we designed this mechanism into the Response message. We propose a simple method although there are several ways to send results. We design a property, called category, which has a list of values: success, failure, and partial success. If all the results of the specific level, such as PhysicalObject class level, are the same, either success or failure is selected. When parts of the results of that level succeed, partial success is selected.

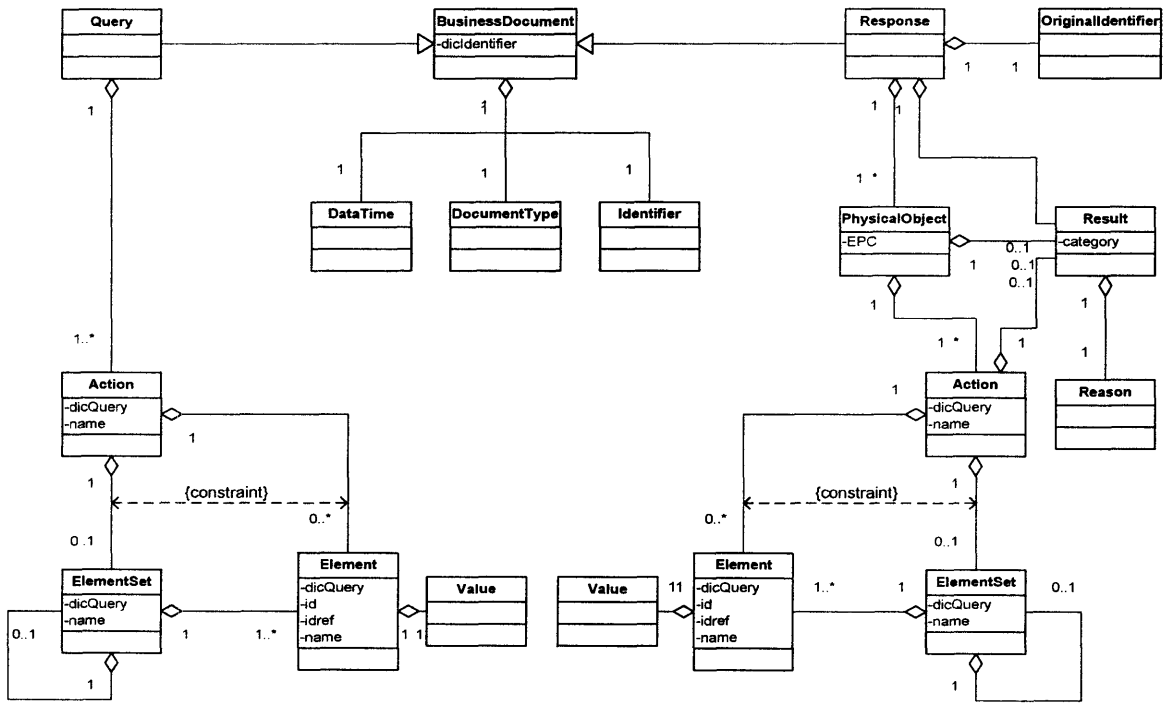

Figure 5-3-3: Class diagram of the Query/Response message

Table 5-3-2: Class and property description of the Query/Response message

Class Name	Description
BusinessDocument	Abstract class for the business message. The Notify class, the Query/Response classes and the Acknowledge class are the generalized class of this abstract class.

Class Name	Description
DateTime	The data time stamp for the message sent.
DocumentType	Either "notify," "query," "response," or "acknowledge."
Identifier	Identifier of the business message.
Query	Class for the Query message.
Response	Class for the Response message.
PhysicalObject	Class for the subjected physical object. This class has one to many relation with the Response class, which means one Response message sends the data of multiple physical objects.
Action	Class for the action done to the physical objects, which is defined in the Action class in the dictionary. This class has one to many relation with the Query class in the Query message, and the PhysicalObject class in the Response message.
ElementSet	Class for the aggregated values of each Action class instance. The Action class takes either the ElementSet class or the Element class as the child class. This class does not have any instances.
Element	Class for the values of each Action class instance. The Action class takes either the ElementSet class or the Element class as the child class. This class does not have any instances.
Value	Class for the relevant information values of each Action class instance. The pair of the Name class and the Value class describes the relevant information of the Action class instance.
OriginalIdentifier	Class for the identifier of the Query message to which the Response message is response.
Result	Class for the result of the query.
Reason	Class for the reason when an exception occurs.
dicIdentifier	Property to identify the dictionary type and the version.
EPC	Property to identify the EPC of the class.
dicRef	Property to refer the Action, ValueSet, and Value class instance of the dictionary.
ver	Property to refer the version of each class.
name	Property to show the name of each dictionary instance.
category	Property to show the result of the Query message. The value for this property is either "success", "failure", or "partial success.

Class Name	Description
id	Identifier for the instance.
idref	Identifier reference for the instance.

5.3.3. Acknowledge message

Figure 5-3-4 shows the message structure of the Acknowledge message, and Table 5-3-3 shows the description of each class and property.

The BusinessDocument class is an abstract class for all the messages. The Acknowledge class is specialized from the BusinessDocument class. Since the result of the Notify message may be defined either message level, PhysicalObject class level, or Action class level, the Result class is associated with the Acknowledge class, the PhysicalObject class and the Action class. Results of the Acknowledge message are sent in the same way as the Response message.

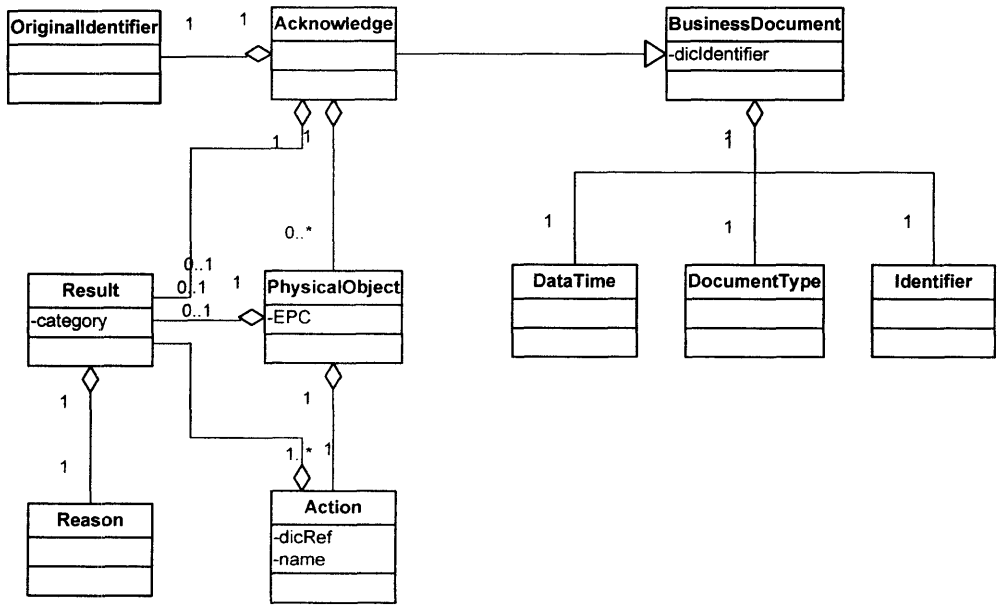

Figure 5-3-3: Class diagram of the Acknowledge message

Table 5-3-3: Class and property description of the Acknowledge message

Class Name	Description
BusinessDocument	Abstract class for the business message. The Notify class, the Query/Response classes and the Acknowledge class are the generalized class of this abstract class.
DateTime	The data time stamp for the message sent.
DocumentType	Either "notify," "query," "response," or "acknowledge."

Class Name	Description
Identifier	Identifier of the business message.
Acknowledge	Class for the Acknowledge message.
PhysicalObject	Class for subjected physical object. This class has one to many relation with the Acknowledge class, which means one Acknowledge message sends data of multiple physical objects.
Action	Class for the action done to the physical objects, which is defined in the Action class in the dictionary. This class has one to many relation with the PhysicalObject class.
OriginalIdentifier	Class for the identifier of the Notify message.
Result	Class for the result of the Notify message.
Reason	Class for the reason when an exception occurs.
dicIdentifier	Property to identify the dictionary type and the version.
EPC	Property to identify the EPC of the class.
dicRef	Property to refer the Action, ValueSet, and Value class instance of the dictionary.
ver	Property to refer the version of each class.
name	Property to show the name of each dictionary instance.
category	Property to show the result of the Query message. The value for this property is either "success", "failure", or "partial success.

6. Evaluation

In the previous two chapters, we defined requirements for EPC-IS and, based on the requirements, we model messages used for EPC-IS as well as the PML Extension as Property Dictionary and Event Dictionary. In this chapter, we evaluate the model from other perspectives, such as issues raised in the RFID trials and the expectation towards RFID. The points we deal with are as follows: 1) imperfect tag detection, 2) security, and 3) further business processes.

6.1. Imperfect tag detection

Although RFID obviates the need for contact and error rate is considered to be lower than that of bar code, the read rate is not 100%. From the field trial conducted by Auto-ID Lab, read rate was about 97% [28]. This rate will become worse when items are packed in cases or cases are loaded on pallets.

In order to increase read rate, companies start thinking about using the EPC of the case as a representing EPC of its entire items and the EPC of the pallet as a representing EPC of the cases on it and so forth. Suppose cases that are loaded on a pallet are sent from location A to location B, the relation between the EPCs of each case and the EPC of the pallet can be linked at location A. If the relation is successfully linked and operators at location B can use the data, they deem all the cases are arrived when they scan the EPC of the pallet. This technique is called aggregation, association or inferred reading [29].

In order to realize this inferred reading, there are three things that need to be done: the event of aggregation and disaggregation need to be defined, the historical data needs to be stored, and the data needs to be effectively retrieved by Query/Response message from the receiving company.

Table 6-1-1 shows the event data for aggregation and disaggregation. Attributes of the events are similar to detect event previously defined in the requirement analysis. This means that these events are accommodated with the Event Dictionary and that the messages modeled in Chapter 5 can handle the historical data of these events.

Table 6-1-1 Structure of aggregation and disaggregation event

Action	Key	Relevant information
Aggregate	EPC of instance	Date time stamp, Reader EPC, Aggregated EPCs, Aggregation direction(*)
Disaggregate	EPC of instance	Date time stamp, Reader

Action	Key	Relevant information
		EPC, disaggregated EPC

(*) Aggregation needs at least two types of physical objects: upper layer object such as container and pallet, and lower layer object such as item and case. This direction shows the relation between two physical objects.

A problem occurs when either items in a case or cases on a pallet are lost during transportation. If the actual reading and the inferred reading are recorded equally, one can not tell the difference and it becomes impossible to trace the path of each item. Therefore, the actual reading and the inferred reading should be recorded differently. In order to include this new requirement, we add a new class, the InferredReading class, as an associated class of the Action class.

Considering the data structure of the Notify message and the Response message, the same modification, adding the InferredReading class, needs to be done for the Response message as well.

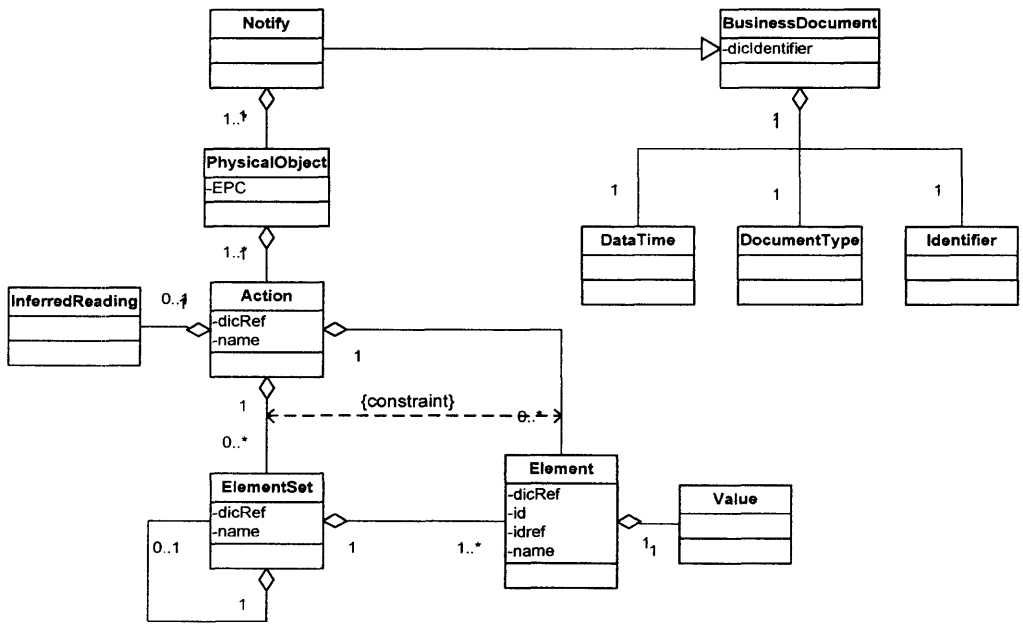

Figure 6-1-1: Modified class diagram for the Notify message

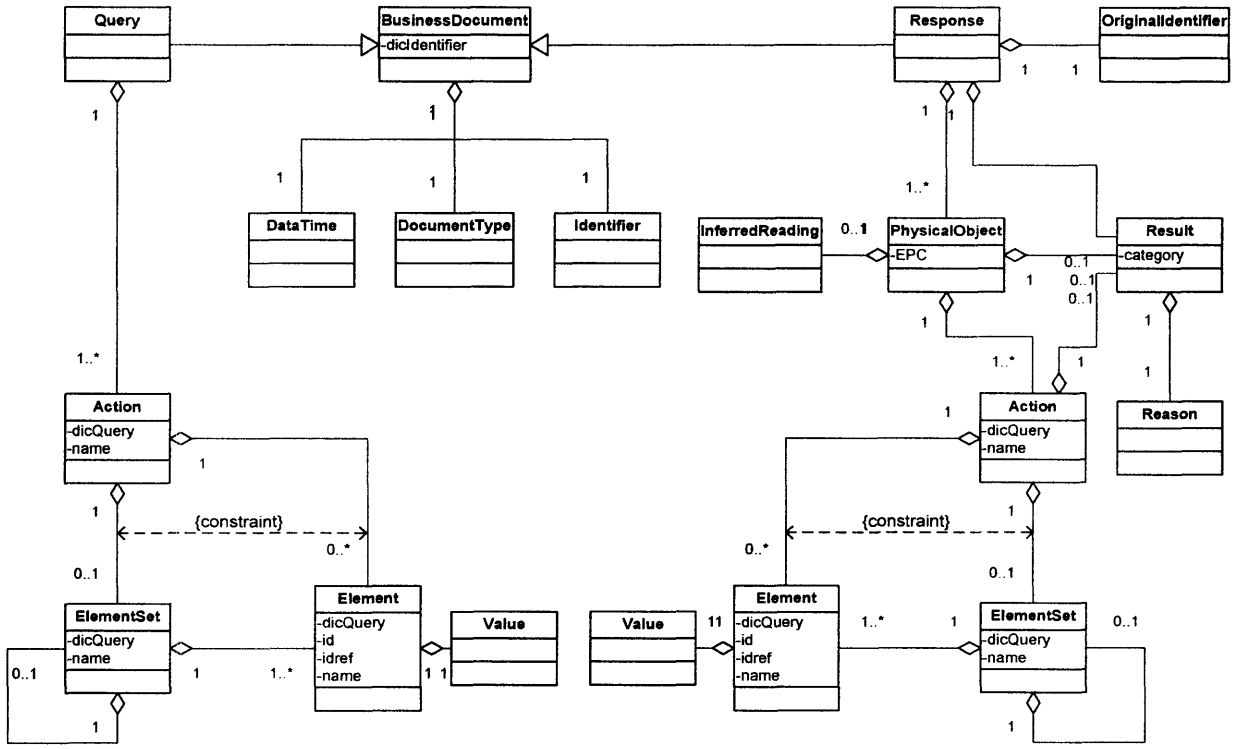

Figure 6-1-2: Modified class diagram for the Query/Response message

6.2. Security

Security is one of the most significant issues for the information system implementation. Security issues for the message exchange are categorized as Authorization, Authentication, Non-Repudiation, Confidentiality, and Integrity.

When considering the security issues, there are two important things that need to be considered: 1) the relation between the owner of a server which stores physical object data and the owner of a software component which accesses to the server, and 2) the messaging protocol technology used between the server and the software component.

If the server is accessed from a Savant in the same company, the security level of this connection could be lower than that when the server is accessed from a software component in the trading partner. This is also dependent on the nature of the data which is accessed. If the data is product-level property data, such as size, manufacturer, and other technical specifications, there may be few security issues. Just as the catalog information is accessed almost freely by everyone who uses the Internet.

From this observation that the data exchanged is the same but the security level may change dependent on the relation between owners, we conclude that the security issues should be taken care of not by the message schema but by the lower layer, which is the messaging protocol.

There are a couple of messaging protocols which may be used to exchange the physical object information. If both a server which stores data of physical objects and a software component which accesses to the server are located in the same company's network, they may be connected with MQ technology or JMS, whereas if a software component is outside of the company's network, SOAP is the strong candidate to connect servers [30], [31], [32].

The important thing here is that each messaging protocol has its own mechanism to guarantee the security issues. Therefore, we propose that the security issues are not taken care by the message schema but by the messaging protocols that are selected based on the business requirements.

6.3. Further business processes

One of the highly expected business processes for the RFID technologies is the process to track and trace physical object movement. Although we developed a track business process and a trace business process as the generic business processes and modeled EPC-IS based on them, the model might not meet the future business requirements. Therefore, after this section, we will evaluate our model with further complex business processes: 1) handling trace business process, 2) assembly trace business process, and 3) order track business process.

6.4. Handling trace business process

Handling trace business process is useful in such cases as the international shipment which requires handling by many different companies and pharmaceutical tracing which may be legislated in some of the states [33].

Regarding the system layout, we premise the structure described in Figure 6-4-1. We assume four companies, and the difference is indicated by the suffix. We also assume that ONS is public so that every company can use the ONS equally. Regarding Savant, we assume a one-layer structure. Each Savant is connected to a server with EPC-IS (we represent this server as EPC-IS in the system layout). Savant1 and Savant4 are also linked with the enterprise application. EPC-IS1 stores the property data of items and the historical data that is detected by Reader1, whereas EPC-IS2, EPC-IS3, and EPC-IS4 store the historical data that is detected by Reader2, Reader3 and Reader4, respectively. The enterprise application controls business transaction data, such as purchase order and advance ship notice, and also provides the interface of business process to operators.

Preconditions of this business process are that all the companies settle terms and conditions about this business process, original product data is stored in the Enterprise Application1 and the Enterprise Application2, Company4 has already received the product level data and placed an order to Company1, and Company1~4 implement the Auto-ID infrastructure.

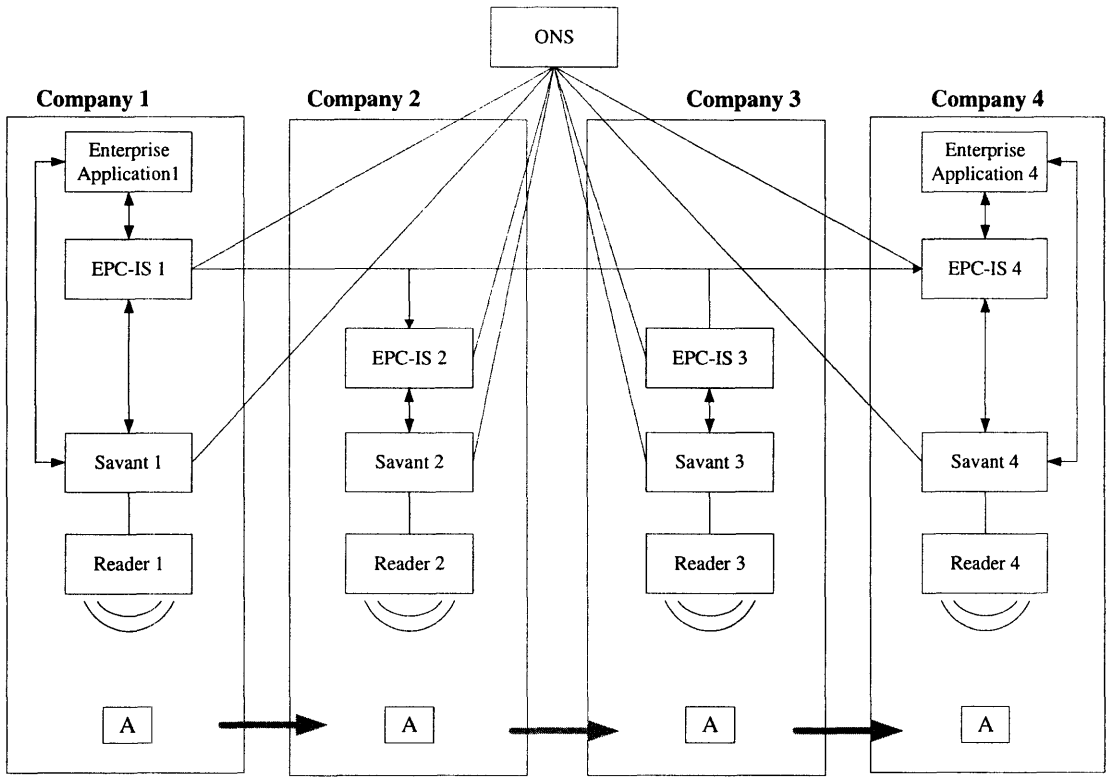

Figure 6-4-1: System layout for the handling trace business process

6.4.1. Business process flow

Figure 6-4-2 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Send the instance-level data of the items to Company4: The instance-level data is sent from Company1 to Company4. The process is the same as that of the Query instance-level data business process. (See flow explanation of the Query instance-level data business process step 1 through 6.)

2. Send the detected data from the Savants to the EPC-ISs: When the items are shipped, they are scanned by the readers and the data is sent from the Savant of each company to the server with EPC-IS of each company. In this sample process, we assume that the Savant2, the Savant3 and the Savant4 are located at Company2's Company3's, and Company4's facility, respectively. When the items are scanned; the data is sent to the Savant2, the Savant3 and the Savant4; and then the each Savant sends the data to the server in their facility. When the data is sent successfully, an acknowledge message returns to the Savant which sends the message to the server.

3. Register the EPC and the server to the ONS: The relation between the instance-level EPC and the server location is registered to the ONS at Company2, Company3 and Company4. If the data is registered successfully, acknowledgement messages return to each Savant.

4. Query the instance-level data from the Enterprise Application4 to the EPC-IS4: After the step 1, the Enterprise Application4 has the instance-level EPC of the items Company4 orders. Suppose an employee wants to know the company that the individual item is handled, he operates the Enterprise Application4 and the Enterprise Application4 sends a query to the EPC-IS4.

5. EPC-IS4 look-up: After the step 4, the EPC-IS4 looks up the location of servers which store the data of a specific EPC. In this case, the EPC-IS4 gets the location of the EPC-IS1, the EPC-IS2, the EPC-IS3 and the EPC-IS4 itself.

6. Query the instance-level data from the EPC-IS4 to other EPC-ISs: After the step 5, the EPC-IS4 recognizes the location of other EPC-ISs. Then the EPC-IS4 queries the instance-level data of a specific individual item to the EPC-IS1, the EPC-IS2 and the EPC-IS3 and gets the information.

7. Response from the EPC-IS4 to the Enterprise Application4: In response to the step 4, the EPC-IS4 sends a response back to the Enterprise Application4. The response contains all the company data that EPC-IS4 receives in the step 6 and company data that the EPC-IS4 stores.

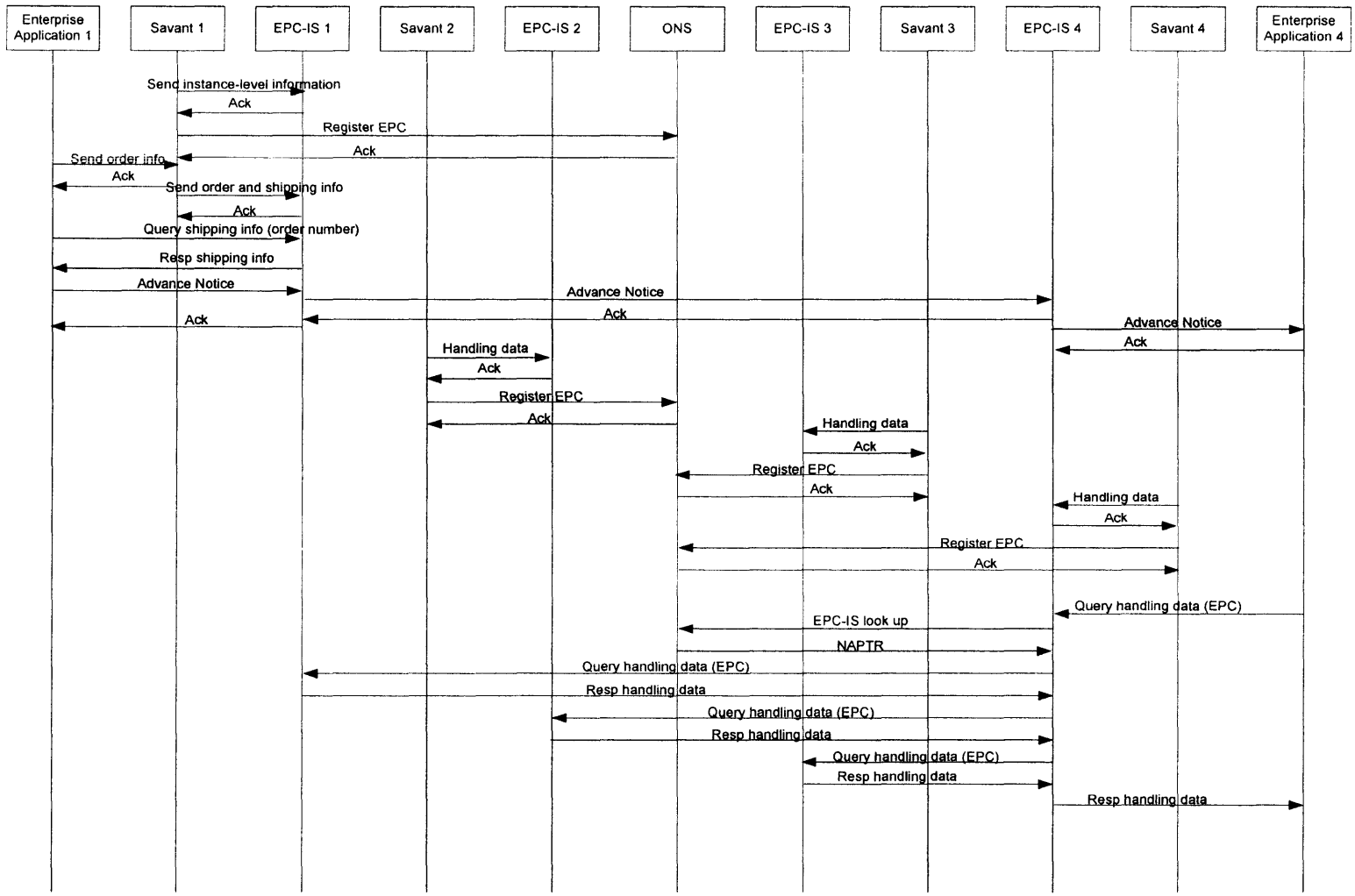

Figure 6-4-2: Work flow of the handling trace business process

6.4.2. Requirement analysis

Based on the business flow defined in the previous section, we identify the need for defining an event for handling. The event is described in Table 6-4-2. Attributes of the event are similar to the events previously defined. This means that this event is also accommodated with the Event Dictionary and that the messages modeled in Chapter 5 can handle the historical data of this event.

Table 6-4-2 Structure of the handling event

Action	Key	Relevant information
Handle	EPC of instance	Date time stamp, Reader EPC, business entity which handles the instance

6.5. Assembly trace business process

Assembly trace business process is useful in such cases as tracing hazardous materials in the electronic appliances, which is necessary in the environment where the use of hazardous substances is strictly prohibited [34].

Regarding the system layout, we premise the structure in Figure 6-5-1. We assume three companies, and the difference is indicated by the suffix. We also assume that ONS is public so that every company can use the ONS equally. Regarding Savant, we assume a one-layer structure. Each Savant is connected to a server with EPC-IS (we represent this server as EPC-IS in the system layout) and the enterprise application. EPC-IS1 stores the property data and the historical data of the component(A) that are detected by Reader1, in the same way, EPC-IS2 stores the property data and the historical data of the component(B) that are detected by Reader2. EPC-IS3 stores the property data and the historical data of product(C) that are detected by Reader3. The enterprise application controls the business transaction data, such as purchase order and advance ship notice, and also provides the interface of the business process to operators.

Preconditions of this business process are that all the companies settle terms and conditions about this business process; original product data is stored in the Enterprise Application1, Enterprise Application2 and Enterprise Application3; Company3 has already received the product level data from Company1 (component (A)) and Company2 (component (B)) and placed an order to Company1 and Company2; and Company1, Company2 and Company3 implement the Auto-ID infrastructure.

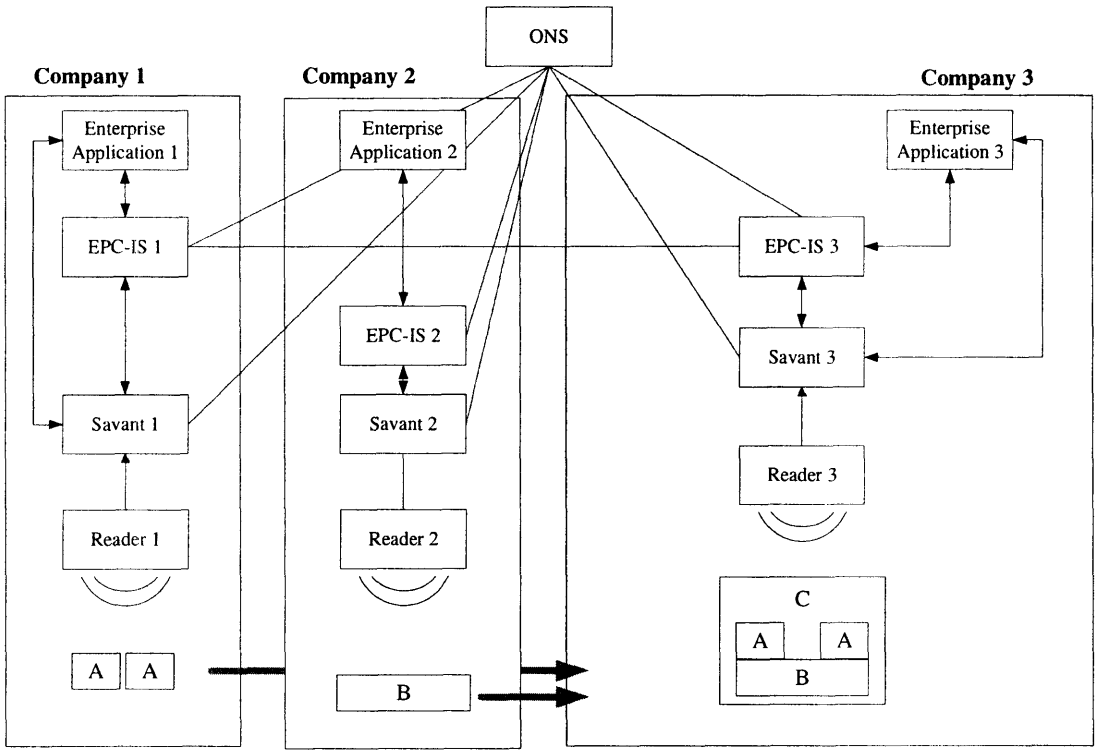

Figure 6-5-1: System layout for the assembly trace business process

6.5.1. Business process flow

Figure 6-5-2 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Send the instance-level data of the component (A) to Company3: The instance-level data is sent from Company1 to Company3. The process is the same as that of Query instance-level data business process. (See flow explanation of the Query instance-level data business process step 1 through 6.)
2. Send the instance-level data of the component (B) to Company3: The instance-level data is sent from Company2 to Company3. The process is the same as that of the Query instance-level data business process. (See flow explanation of Query instance-level data business process step 1 through 6.)
3. Register product (C) information: Original product-level data of the product (C) stored in the Enterprise Application3 is sent to the EPC-IS3. When the data is registered successfully, an acknowledgement message returns to the Enterprise Application3.

4. Send the instance-level data of the product (C) with the EPC to the EPC-IS3: After the product (C) is assembled and scanned, and the relevant data is inputted from the Savant3, the Savant3 sends the data to the EPC-IS3. When the data is sent successfully, an acknowledge message returns to the Savant3.

5. Register the EPC and the EPC-IS to the ONS: The relation between the instance-level EPC of the product (C) and the EPC-IS3 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the Savant3.

6. Query the instance-level data from the Enterprise Application3 to the EPC-IS3: After the step 4, the Enterprise Application3 has the instance-level EPC of the components that Company3 uses. Suppose an employee wants to know the location of a product (C) which contains a specific component (A) (whose EPC is known), he operates the Enterprise Application3, and the Enterprise Application3 sends a query to the EPC-IS3.

7. Response from the EPC-IS3 to the Enterprise Application3: In response to the step 6, the EPC-IS3 sends a response back to the Enterprise Application3 with the EPC of a product (C) which uses the specific component (A).

8. Query the instance-level data from the Enterprise Application3 for a product (C) location: After the step 7, the operator sends a query from the Enterprise Application3 to the EPC-IS3 using the EPC of the product (C) he gets in the previous step.

9. EPC-IS3 look-up: After the step 8, the EPC-IS3 looks up the location of servers which store the data of the specific EPC. In this case, the EPC-IS3 gets the location information of the EPC-IS3 itself.

10. Query the instance-level data from the EPC-IS3 to other EPC-ISs: After the step 9, the EPC-IS3 recognizes the location of other EPC-ISs. Then the EPC-IS3 queries the instance-level data of the specific individual item to the EPC-ISs. In this case, the EPC-IS3 does not send any query to other EPC-ISs, but if the product (C) has already shipped, the EPC-IS3 sends a query to the servers which store the historical data of the product.

11. Response from the EPC-IS3 to the Enterprise Application3: In response to the step 10, the EPC-IS3 sends a response back to the Enterprise Application3. The response contains all the location data that the EPC-IS3 receives in the step 10 and the location data that the EPC-IS3 stores.

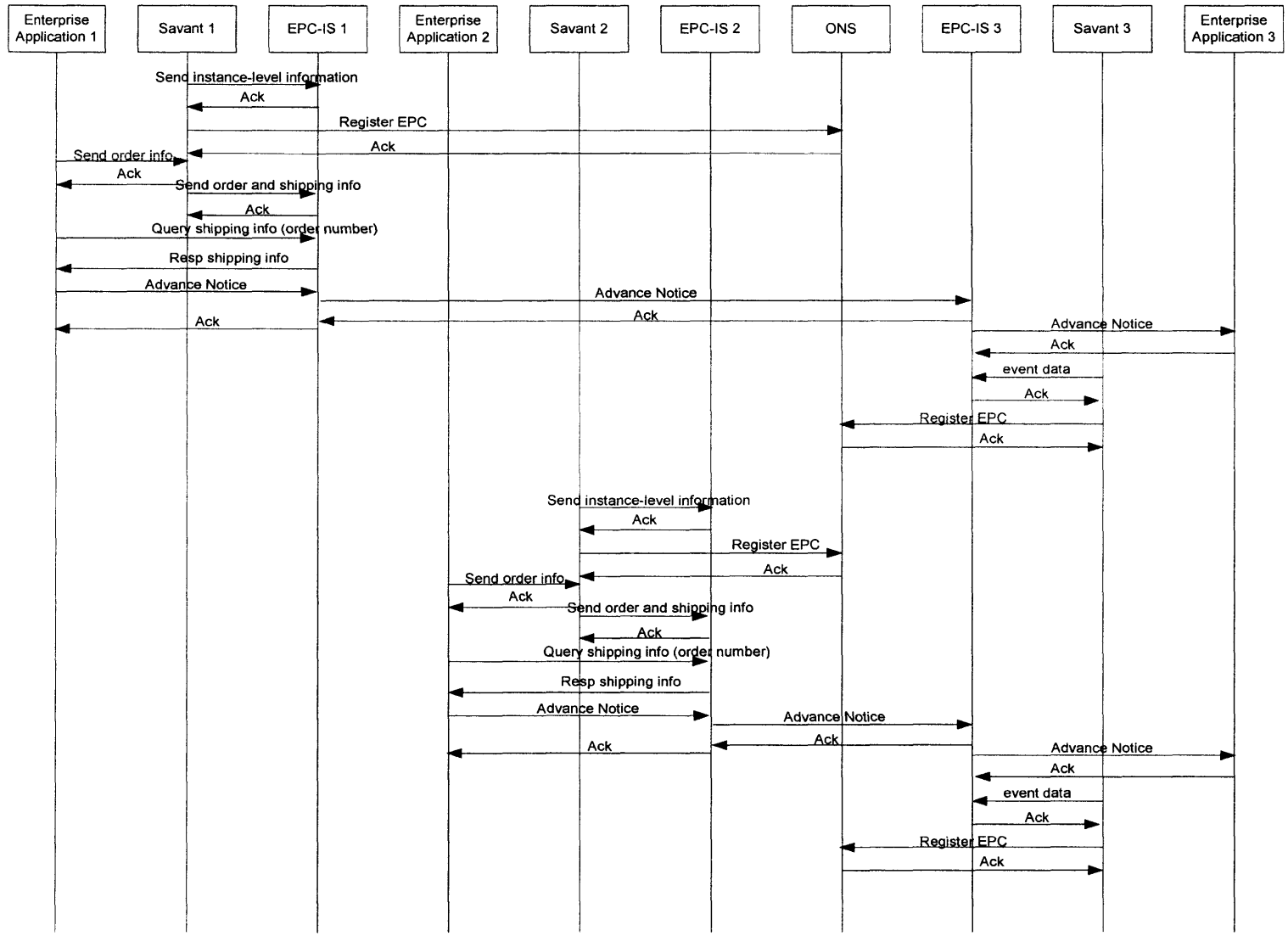

Figure 6-5-1: Work flow of the assembly trace business process (1/2)

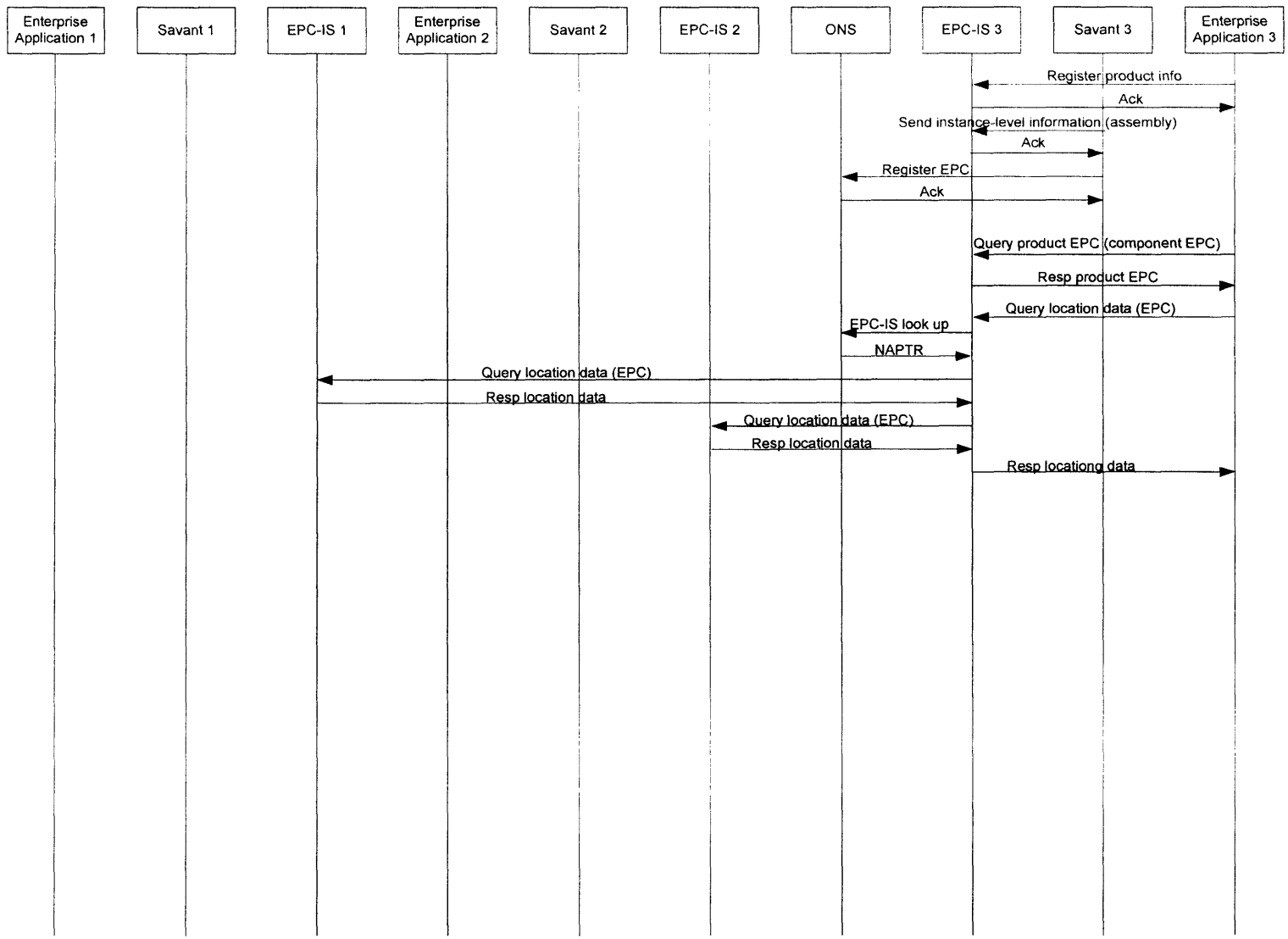

Figure 6-5-1: Work flow of the assembly trace business process (2/2)

6.5.2. Requirement analysis

Based on the business flow defined in the previous section, we identify the need for defining a new event, assembly. The event is described in Table 6-5-1. Attributes of the event are similar to the events previously defined. This means this event is also accommodated with the Event Dictionary and that the messages modeled in Chapter 5 handle the historical data of this event.

Table 6-5-1 Structure of the assembly event

Action	Key	Relevant information
Assembly	EPC of instance	Date time stamp, Reader EPC, component(s)EPC, Assembly direction

6.6. Order track business process

Order track business process is useful in such cases as tracking the status of the products which have long manufacturing process time and doing quality error management. [35].

Regarding the system layout, we premise the structure in Figure 6-6-1. We assume two companies, and the difference is indicated by the suffix. We also assume that ONS is public so that every company can use the ONS equally. Regarding Savant, we assume a two-layer structure. Savant1 is connected to servers with EPC-IS (we represent these servers as EPC-IS in the system layout) and the enterprise application, and Savant1-1 ~ Savant1-3 are connected to the readers and Savant1. EPC-IS1 stores the property data and the historical data of the component (A) that are detected by Reader1-1, component (B) that are detected by Reader1-2. Product (C) will be detected by Reader1-3 after the assembly process starts. The enterprise application controls the business transaction data, such as purchase order and advance ship notice, and also provides the interface of the business process to operators.

Preconditions of this business process are that these companies settle terms and conditions about this business process, the bill of materials (BoM) is stored in the Enterprise Application1, Company2 has already placed an order to Company1, and Company1 and Company2 implement the Auto-ID infrastructure.

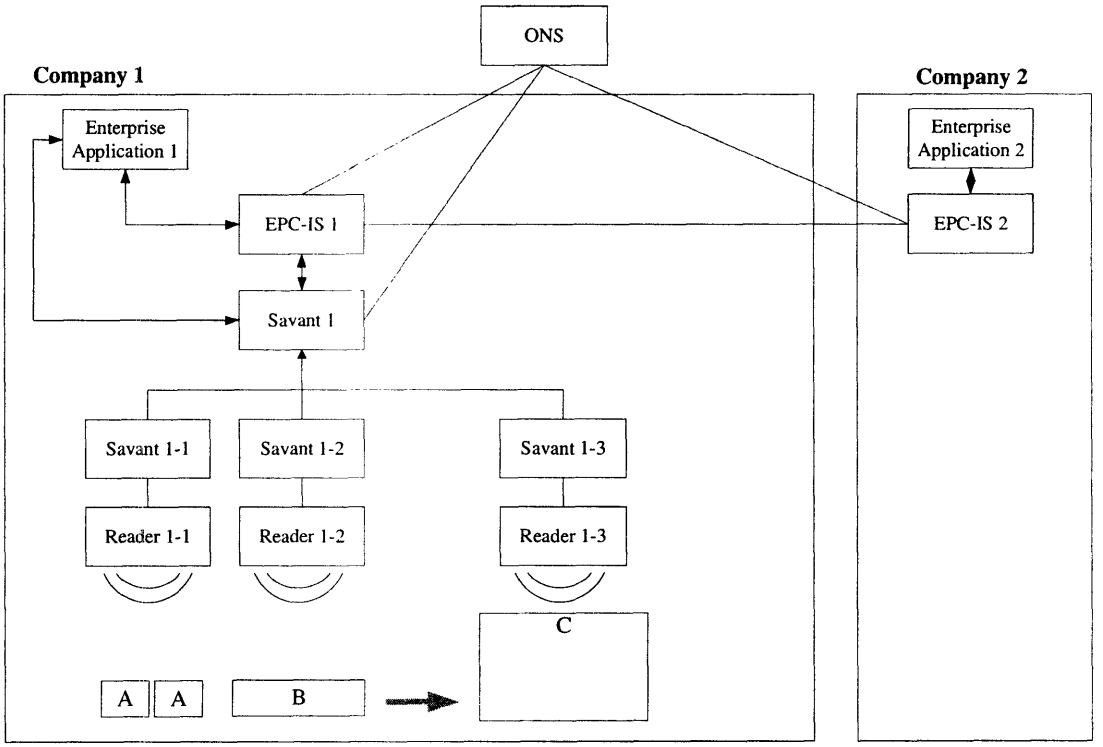

Figure 6-6-1: System layout of the Order track business process

6.6.1. Business process flow

Figure 6-6-2 shows the flow of the business process. Each arrow represents the message sent between the system components.

1. Send the instance-level data of the component (A) with the EPC from the Savant1-1 to the Savant1: After each instance is scanned and the relevant data is inputted from the Savant1-1, it sends the data to the Savant1. When the data is sent successfully, an acknowledge message returns to the Savant1-1.

2. Send the instance-level data of the component (B) with the EPC from the Savant1-2 to the Savant1: After each instance is scanned and the relevant data is inputted from the Savant1-2, it sends the data to the Savant1. When the data is sent successfully, an acknowledge message returns to the Savant1-2.

3. Send the instance-level data of the components with the EPC from the Savant1 to the EPC-IS1: After receiving the data from the Savant1-1 and the Savant1-2, the Savant1 sends the data to the EPC-IS1. When the data is sent successfully, an acknowledge message returns to the Savant1.

4. Query available components of the product (C) from the Enterprise Application1 to the EPC-IS1: the Enterprise Application1 queries available component information (EPC of component (A), (B)) to arrange a purchase order from Company3. The key of the query is the product-level EPC of the necessary components that the Enterprise Application1 has as a BoM of the product (C).

5. Send the order information and preassigned the EPC from the Enterprise Application1 to the EPC-IS1: Based on the order that Company1 received from Company2 and the component availability, the Enterprise Application1 preassigns an EPC for a product (C) and sends it to the EPC-IS1 with the relevant information. When the data is sent successfully, an acknowledge message returns to the Enterprise Application1.

6. Register the EPC and the server to the ONS: The relation between the instance-level EPC of the product (C) and the EPC-IS1 location is registered to the ONS. If the data is registered successfully, an acknowledgement message returns to the EPC-IS1.

7. Send an order status notice to Company2: After the step 6, the Enterprise Application1 sends an order status notice to Company2. We assume the Enterprise Application1 sends a notification to the EPC-IS1 and then the EPC-IS1 sends a notification to the EPC-IS2, but this information may be sent via EDI/B2Bi connection if they have a connection. We also assume that the EPC-IS2 pushes the notification to the Enterprise Application2, but the Enterprise Application2 may query new information to the EPC-IS2.

8. Query the instance-level data from the Enterprise Application2 to the EPC-IS2: After the step 7, the Enterprise Application2 has the instance-level EPC of the product (C) that Company3 will receive. Suppose an employee wants to know the status of the order, he operates the Enterprise Application2 and the Enterprise Application2 sends a query to the EPC-IS2.

9. EPC-IS2 look-up: After the step 8, the EPC-IS2 looks up the location of servers which store the data of the specific EPC. In this case, the EPC-IS2 gets the location of the EPC-IS1.

10. Query the instance-level data from the EPC-IS2 to the EPC-IS1: After the step 9, the EPC-IS2 recognizes the location of the EPC-IS1. Then the EPC-IS2 queries the instance-level data of the specific individual item to the EPC-IS1. In this case, the EPC-IS2 gets the status of the product (C) that Company2 will receive.

11. Response from the EPC-IS2 to the Enterprise Application2: In response to the step 8, the EPC-IS2 sends a response back to the Enterprise Application2. The response contains all the status information that the EPC-IS2 receives in the step10.

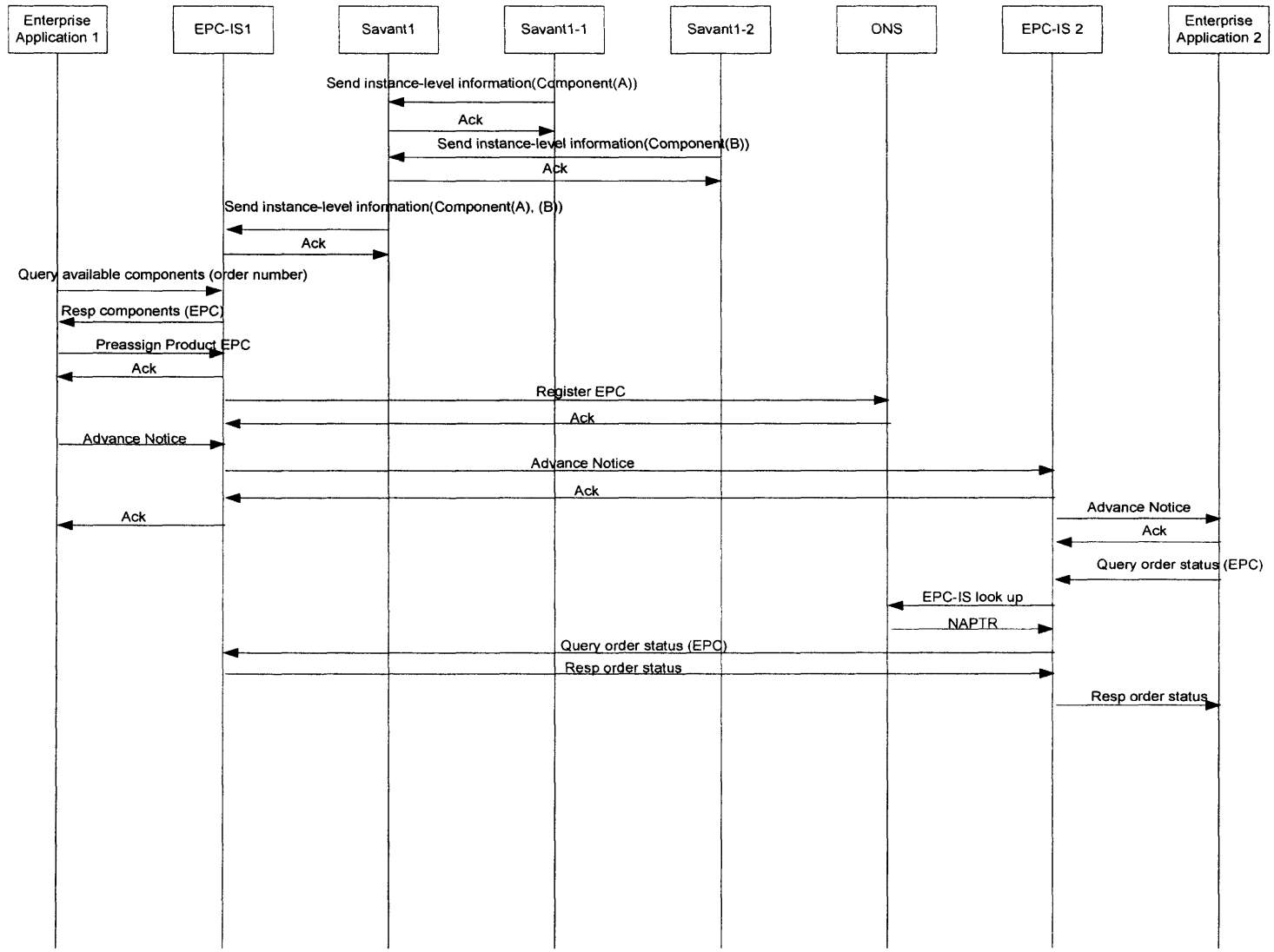

Figure 6-5-2: Work flow of the order track business process

6.6.2. Requirement analysis

Based on the business flow defined in the previous section, we identify the need for defining a new event, notify. The event is described in Table 6-6-1. Attributes of the event are similar to the events previously defined. This means this event is also accommodated with the Event Dictionary and that the messages modeled in Chapter 5 handle the historical data of this event.

Table 6-6-1 Structure of the notify event

Action	Key	Relevant information
Notify	EPC of instance	Purchase order identifier, status of the product

7. Summary and Suggestions

In this chapter, we summarize this thesis by discussing some of the findings of this thesis and making some suggestions for the future study.

7.1. Summary

7.1.1. EPC Information Service architecture

The key objective of this thesis is to propose EPC-IS architecture. We defined generic business processes by using the characteristics of information stored in the servers with EPC-IS. This characteristic was studied by Cambridge University, one of the Auto-ID Lab universities. By defining business processes this way and analyzing requirements in the following step, we guaranteed that the architecture we proposed was able to support these business processes.

Then we defined the requirements derived from those generic business processes, proposed architecture which consists of messages and dictionaries, and developed schemas for both messages and dictionaries. This separation makes it possible to maintain each of the schemas independently. It is also expected to reduce the impact of the business process change on the message structure because the impact of the change is absorbed by modifying dictionary instances.

Then we evaluated the message and the dictionary schemas with further requirements which are being raised in the RFID field trials and the expected future applications for RFID. Through this evaluation, we modified the message schemas, increased the dictionary instances, and made them more robust.

7.1.2. Message

One of the proposals we made is the necessity of the messages in order to realize the generic business processes. The messages we proposed are three: Notify message, Query/Response message and Acknowledge message. The Notify message is used for registering and updating the physical object information, the Query/Response message is used for retrieving information stored in the server with EPC-IS, and the Acknowledge message is used for notifying acknowledgement and exception of the Notify message. We also proposed the schemas of these messages. The document type definition (DTD) of schemas and the sample instances of them are listed in the Appendix.

7.1.3. Dictionary

Another proposal we made is the separation of the dictionary from messages in order to guarantee the scalability of the massages. We developed one dictionary schema and two sets of instances as the Event Dictionary, which maintains events and the attributes of them, and the Property Dictionary, which maintains attributes of both product-level and instance-level property.

Since the attributes defined in the dictionaries are used to describe the physical object information within Auto-ID infrastructure, these dictionaries are also used as the PML Extension. We list the document type definition (DTD) of the dictionary schema, the Event Dictionary and the Property Dictionary in the Appendix.

7.2. Suggestions for future study

7.2.1. Standard development

In this thesis we proposed architecture of EPC-IS. Since starting from generic business processes, we assume that this architecture could become core architecture of a new EPC-IS standard. However, the research to discuss this assumption has not done yet. Therefore, one future study area is to verify the architecture and develop a new EPC-IS standard with the real business requirements.

7.2.2. Impact assessment of merging EDI/B2B infrastructure with Auto-ID infrastructure

In this thesis, we discussed the possibility of sending data via EDI/B2Bi connection and concluded that there is little difference between connection of the EDI/B2Bi infrastructure and that of the Auto-ID infrastructure. However, we did not discuss the impact of using the Auto-ID infrastructure to exchange other transaction messages defined in the EDI/B2B standards, such as quote, purchase order, and invoice.

Since it is envisaged that companies will maintain fewer connections within their systems and between their systems and those of the trading partners, this study is essential to further deploy the Auto-ID infrastructure.

8. References

[1] The Association for Automatic Identification and Data Capture Technologies. *Shrouds of time, The history of RFID*
http://www.aimglobal.org/technologies/rfid/resources/shrouds_of_time.pdf

[2] Auto-ID Lab Silvio Albano. *EPC field test Board of overseers meeting June 2002, Cambridge, England*

[3] Timothy Porter Milne. *A Framework for Auto-ID Enabled Business*

[4] Auto-ID Lab. *About Auto-ID Lab*
http://www.autoidlabs.com/aboutthelabs.html

[5] EPCglobal. *About EPCglobal*
http://www.epcglobalinc.com/about/about_epcglobal.html

[6] EPCglobal. *FAQ, What is EPCglobal Network?*
http://www.epcglobalinc.com/about/faqs.html#8

[7] EPCglobal. *Auto-ID Reader Protocol 1.0*

[8] EPCglobal. *Auto-ID Savant Specification 1.0*

[9] Mark Harrison. *White Paper EPC. Information Service Data Model and Queries*
http://www.autoidlabs.com/whitepapers/CAM-AUTOID-WH025.pdf

[10] EPCglobal. *Auto-ID Object Name Service (ONS) 1.0*

[11] EPCglobal. *EPCTM Tag Data Standards Version 1.1 Rev.1.23*

[12] EPCglobal. *PML Core Specification 1.0*

[13] EPCglobal US. *Welcome to EPCglobal US!*
http://www.epcglobalus.org/

[14] EPCglobal. *13.56 MHz ISM Band Class 1Radio Frequency Identification Tag Interface Specification: Candidate Recommendation, Version 1.0.0*

[15] EPCglobal. *Draft protocol specification for a 900 MHz Class 0 Radio Frequency Identification Tag*

[16] EPCglobal. *860MHz 930MHz Class I Radio Frequency Identification Tag Radio Frequency & Logical Communication Interface Specification Candidate*

Recommendation, Version 1.0.1

[17] Sanjay Sarma, David L. Brock & Kevin Ashton. *The Networked Physical World Proposals for Engineering the Next Generation of Computing, Commerce & Automatic-Identification*
http://www.autoidlabs.com/whitepapers/MIT-AUTOID-WH-001.pdf

[18] Mark Harrison, Humberto Moran, James Brusey, Duncan McFarlane. *PML Server Developments*
http://www.autoidlabs.com/whitepapers/cam-autoid-wh015.pdf

[19] ebXML. *ebXML Technical Work*
http://www.ebxml.org/technical_work.htm

[20] ebXML. *About ebXML*
http://www.ebxml.org/geninfo.htm

[21] ebXML. *Core Component Overview Version 1.05*
http://www.ebxml.org/specs/ccOVER.pdf

[22] RosettaNet.
http://www.rosettanet.org/

[23] W3C. *Extensible Markup Language (XML)*
http://www.w3.org/XML/

[24] RosettaNet. *PIP2A9 Technical Product Information Exchange Protocol*

[25] RosettaNet. *PIP® Design Guidelines*

[26] World Wide Web Consortium.
http://www.w3.org/

[27] W3C. *XQuery 1.0: An XML Query Language*
http://www.w3.org/TR/xquery/

[28] Auto-ID Lab Silvio Albano, Daniel W. Engels. *Technical Report Auto-ID Center Field Trial: Phase I Summary*
http://www.autoidlabs.com/whitepapers/MIT-AUTOID-TR-006.pdf

[29] Auto-ID Lab Silvio Albano. *Auto-ID Field Test Lessons Learned in the Real World*

[30] IBM. *WebSphereMQ*
http://www-306.ibm.com/software/integration/wmq/

[31] Sun Microsystems. *J2EE Java Message Service (JMS)*
http://java.sun.com/products/jms/

[32] W3C. *XML Protocol Working Group*
http://www.w3.org/2000/xp/Group/

[33] U.S. Food and Drug Administration. *COMBATING COUNTERFEIT DRUGS A Report of the Food and Drug Administration*
http://www.fda.gov/oc/initiatives/counterfeit/report02_04.html

[34] European Union. *DIRECTIVE 2002/95/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment*
http://europa.eu.int/eur-lex/pri/en/oj/dat/2003/l_037/l_03720030213en00190023.pdf

[35] Duncan McFarlane, Sanjay Sarma, Jin Lung Chirn, C Y Wong, Kevin Ashton. *Submitted to Journal of EAIA, July 2002 THE INTELLIGENT PRODUCT IN MANUFACTURING CONTROL*
http://www-mmd.eng.cam.ac.uk/automation/Paper/2002_journal_eaia_dmss.pdf

[36] U.S. Department of Defense. *DOD ANNOUNCES RADIO FREQUENCY IDENTIFICATION POLICY*
http://www.defenselink.mil/releases/2003/nr20031023-0568.html

[37] Metro group. *News Release, METRO Group To Introduce RFID Across The Company*
http://www.metrogroup.de/servlet/PB/menu/1008965_12/index.htm

[38] RFID Journal. *Target Issues RFID Mandate*
http://www.rfidjournal.com/article/articleview/802/1/1/

[39] Computer World. *Wal-Mart Backs RFID Technology Will require suppliers to use 'smart' tags by 2005*
http://www.computerworld.com/softwaretopics/erp/story/0,10801,82155,00.html

9. Appendix

9.1. Dictionary

9.1.1. Dictionary Schema

```
<!--
    This is Dictionary schema.
    Copyright (c) Tatsuya Inaba 2004
  -->
<!-- ============================================================
                        CHANGE HISTORY
    Version             Date        Time    Author
    ================    ========    =====   =======
    Initial Draft       20040202    15:00   Inaba
    1.1                 20040224    15:00   Inaba
    1.2                 20040318    11:15   Inaba
============================================================ -->
<!ENTITY % common-elements "Name, Desc">
<!ENTITY % common-attributes "
  id   ID   #REQUIRED
  ver CDATA#REQUIRED
  ">
<!ELEMENT Dictionary (Identifier, Action*, ValueSet*, Value*)>
<!ELEMENT Identifier (Type, MajRev, MinRev, Date)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT MajRev (#PCDATA)>
<!ELEMENT MinRev (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Action (%common-elements;, ValueSetId*, ValueId*)>
<!ATTLIST Action
    %common-attributes;
>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Desc (#PCDATA)>
<!ELEMENT ValueSetId (#PCDATA)>
<!ELEMENT ValueId (#PCDATA)>
<!ELEMENT ValueSet (%common-elements;, ValueId*)>
<!ATTLIST ValueSet
    %common-attributes;
>
<!ELEMENT Value (%common-elements;, Format, UOM)>
<!ATTLIST Value
    %common-attributes;
>
<!ELEMENT Format (#PCDATA)>
<!ELEMENT UOM (#PCDATA)>
```

9.1.2. Event Dictionary

```
<?xml version="1.0"?>
<!DOCTYPE Dictionary SYSTEM "Dictionary.dtd">
<?xml-stylesheet type="text/xsl" href="dictionary.xsl"?>
<Dictionary>
    <Identifier>
        <Type>Event</Type>
        <MajRev>1</MajRev>
        <MinRev>0</MinRev>
        <Date>2004-03-18</Date>
    </Identifier>
    <Action id="EA001" ver="1.0">
        <Name>Detect</Name>
        <Desc>Event used when physical object is scanned.</Desc>
        <ValueId>EV001</ValueId>
        <ValueId>EV002</ValueId>
        <ValueId>EV003</ValueId>
    </Action>
    <Action id="EA002" ver="1.0">
        <Name>Aggregate</Name>
        <Desc>Event used when two physical objects are aggregated.</Desc>
        <ValueId>EV001</ValueId>
        <ValueId>EV002</ValueId>
        <ValueId>EV003</ValueId>
        <ValueId>EV004</ValueId>
        <ValueId>EV005</ValueId>
        <ValueId>EV013</ValueId>
    </Action>
    <Action id="EA003" ver="1.0">
        <Name>Disaggregate</Name>
        <Desc>Event used when two physical objects are disaggregated.</Desc>
        <ValueId>EV001</ValueId>
        <ValueId>EV002</ValueId>
        <ValueId>EV003</ValueId>
        <ValueId>EV004</ValueId>
        <ValueId>EV005</ValueId>
        <ValueId>EV013</ValueId>
    </Action>
    <Action id="EA004" ver="1.0">
        <Name>Handling</Name>
        <Desc>Event used when business entity handles physical object.</Desc>
        <ValueSetId>EVS001</ValueSetId>
        <ValueId>EV006</ValueId>
        <ValueId>EV007</ValueId>
    </Action>
    <ValueSet id="EVS001" ver="1.0">
        <Name>Physical Address</Name>
        <Desc>Physical Address of physical object location, business entity etc.</Desc>
        <ValueId>EV008</ValueId>
        <ValueId>EV009</ValueId>
        <ValueId>EV010</ValueId>
        <ValueId>EV011</ValueId>
        <ValueId>EV012</ValueId>
```

```
    </ValueSet>
    <Value id="EV001" ver="1.0">
        <Name>EPC</Name>
        <Desc>EPC of scanned physical object. Key of historical record.</Desc>
        <Format>EPC</Format>
        <UOM/>
    </Value>
    <Value id="EV002" ver="1.0">
        <Name>DateTime</Name>
        <Desc>DateTime of the event.</Desc>
        <Format>DateTime</Format>
        <UOM/>
    </Value>
    <Value id="EV003" ver="1.0">
        <Name>ReaderEPC</Name>
        <Desc>EPC of reader which scans physical object.</Desc>
        <Format>EPC</Format>
        <UOM/>
    </Value>
    <Value id="EV004" ver="1.0">
        <Name>RelatedEPC</Name>
        <Desc>EPC of related physical object. Used for Aggregation event.</Desc>
        <Format>EPC</Format>
        <UOM/>
    </Value>
    <Value id="EV005" ver="1.0">
        <Name>AggregationRelation</Name>
        <Desc>Relation of the aggregation. If value is "1", physical object is larger concept physical
object.</Desc>
        <Format>Binary</Format>
        <UOM/>
    </Value>
    <Value id="EV006" ver="1.0">
        <Name>BusinessEntity</Name>
        <Desc>Name of business entity which executes the event.</Desc>
        <Format>Text</Format>
        <UOM/>
    </Value>
    <Value id="EV007" ver="1.0">
        <Name>ContactName</Name>
        <Desc>Contact information</Desc>
        <Format>Text</Format>
        <UOM/>
    </Value>
    <Value id="EV008" ver="1.0">
        <Name>Street</Name>
        <Desc>Street of physical address.</Desc>
        <Format>Text</Format>
        <UOM/>
    </Value>
    <Value id="EV009" ver="1.0">
        <Name>City</Name>
        <Desc>City of physical address.</Desc>
        <Format>Text</Format>
        <UOM/>
```

```
    </Value>
    <Value id="EV010" ver="1.0">
        <Name>State</Name>
        <Desc>State of physical address.</Desc>
        <Format>Text</Format>
        <UOM/>
    </Value>
    <Value id="EV011" ver="1.0">
        <Name>Zip</Name>
        <Desc>Zip code of physical address.</Desc>
        <Format>Number</Format>
        <UOM/>
    </Value>
    <Value id="EV012" ver="1.0">
        <Name>Country</Name>
        <Desc>Country of physical address.</Desc>
        <Format>Text</Format>
        <UOM/>
    </Value>
    <Value id="EV013" ver="1.0">
        <Name>Dependency</Name>
        <Desc>Dependency of the two physical objects. If the item is contained in a case, the value is
"1".</Desc>
        <Format>binary</Format>
        <UOM/>
    </Value>
</Dictionary>
```

9.1.3. Property Dictionary

```
<?xml version="1.0"?>
<!DOCTYPE Dictionary SYSTEM "Dictionary.dtd">
<?xml-stylesheet type="text/xsl" href="dictionary.xsl"?>
<Dictionary>
    <Identifier>
        <Type>Property</Type>
        <MajRev>1</MajRev>
        <MinRev>1</MinRev>
        <Date>2004-03-18</Date>
    </Identifier>
    <Action id="PA001" ver="1.0">
        <Name>Create</Name>
        <Desc>Action used to create data</Desc>
        <ValueSetId/>
        <ValueId/>
    </Action>
    <Action id="PA002" ver="1.0">
        <Name>Delete</Name>
        <Desc>Action used to delete data</Desc>
        <ValueSetId/>
        <ValueId/>
    </Action>
```

```
<Action id="PA003" ver="1.0">
    <Name>Update</Name>
    <Desc>Action used to update data</Desc>
    <ValueId/>
</Action>
<Action id="PA004" ver="1.0">
    <Name>Notify</Name>
    <Desc>Action used to notify business document such as ASN</Desc>
    <ValueSetId>PVS002</ValueSetId>
    <ValueId/>
</Action>
<ValueSet id="PVS001" ver="1.0">
    <Name>Manufacturer</Name>
    <Desc>Data related to manufacturer</Desc>
    <ValueId>PV007</ValueId>
    <ValueId>PV008</ValueId>
    <ValueId>PV009</ValueId>
    <ValueId>PV010</ValueId>
    <ValueId>PV011</ValueId>
    <ValueId>PV012</ValueId>
    <ValueId>PV013</ValueId>
</ValueSet>
<ValueSet id="PVS002" ver="1.0">
    <Name>ASN</Name>
    <Desc>Data related to ASN</Desc>
    <ValueId>PV014</ValueId>
</ValueSet>
<Value id="PV001" ver="1.0">
    <Name>EPC</Name>
    <Desc>EPC of property data</Desc>
    <Format>EPC</Format>
    <UOM/>
</Value>
<Value id="PV002" ver="1.0">
    <Name>DateTime</Name>
    <Desc>DateTime of the event.</Desc>
    <Format>YYYYMMDDThhmm+hhmm</Format>
    <UOM>DateTime</UOM>
</Value>
<Value id="PV003" ver="1.0">
    <Name>Height</Name>
    <Desc>Height of physical object</Desc>
    <Format/>
    <UOM>meter</UOM>
</Value>
<Value id="PV004" ver="1.0">
    <Name>Depth</Name>
    <Desc>Depth of physical object</Desc>
    <Format/>
    <UOM>meter</UOM>
</Value>
<Value id="PV005" ver="1.0">
    <Name>Width</Name>
    <Desc>Width of physical object</Desc>
    <Format/>
```

```
    <UOM>meter</UOM>
</Value>
<Value id="PV006" ver="1.0">
    <Name>Weight</Name>
    <Desc>Weight of physical object</Desc>
    <Format/>
    <UOM>kg</UOM>
</Value>
<Value id="PV007" ver="1.0">
    <Name>BusinessEntity</Name>
    <Desc>Name of business entity which executes the event.</Desc>
    <Format>Text</Format>
    <UOM/>
</Value>
<Value id="PV008" ver="1.0">
    <Name>ContactName</Name>
    <Desc>Contact information</Desc>
    <Format>Text</Format>
    <UOM/>
</Value>
<Value id="PV009" ver="1.0">
    <Name>Street</Name>
    <Desc>Street of physical address.</Desc>
    <Format>Text</Format>
    <UOM/>
</Value>
<Value id="PV010" ver="1.0">
    <Name>City</Name>
    <Desc>City of physical address.</Desc>
    <Format>Text</Format>
    <UOM/>
</Value>
<Value id="PV011" ver="1.0">
    <Name>State</Name>
    <Desc>State of physical address.</Desc>
    <Format>Text</Format>
    <UOM/>
</Value>
<Value id="PV012" ver="1.0">
    <Name>Zip</Name>
    <Desc>Zip code of physical address.</Desc>
    <Format/>
    <UOM>Number</UOM>
</Value>
<Value id="PV013" ver="1.0">
    <Name>Country</Name>
    <Desc>Country of physical address.</Desc>
    <Format>Text</Format>
    <UOM/>
</Value>
<Value id="PV014" ver="1.0">
    <Name>DocumentId</Name>
    <Desc>Document identifier of transaction(e.g. PO, ASN)</Desc>
    <Format>Text</Format>
    <UOM/>
```

```
    </Value>
    <Value id="PV015" ver="1.0">
        <Name>ExpirationDate</Name>
        <Desc>Expiration date of the item</Desc>
        <Format>DateTime</Format>
        <UOM/>
    </Value>
</Dictionary>
```

9.2. Message

9.2.1. Schema

9.2.1.1. Notify message

```
<!--
    This is Notify schema.
    Copyright (c) Tatsuya Inaba 2004
  -->
<!-- ========================================
                    CHANGE HISTORY

    Version         Date        Time    Author
    =============   ========    =====   ======
    Initial Draft   20040212    15:00   Inaba
    1.1             20040225    14:00   Inaba
    1.2             20040311    16:00   Inaba
    1.3             20040318    12:00   Inaba

======================================== -->
<!ENTITY % common-elements "DateTime, Identifier, DocumentType">
<!ENTITY % common-attributes "
  dicRef   CDATA   #REQUIRED
  ">
<!ELEMENT Notify (%common-elements;, PhysicalObject*)>
<!ATTLIST Notify
    dicIdentifier CDATA #IMPLIED
>
<!ELEMENT PhysicalObject (Action*)>
<!ATTLIST PhysicalObject
    EPC CDATA #REQUIRED
>
<!ELEMENT Action (ElementSet*, Element*, Inferred?)>
<!ATTLIST Action
    %common-attributes;
>
<!ELEMENT ElementSet (ElementSet?, Element*)>
<!ATTLIST ElementSet
    %common-attributes;
>
<!ELEMENT Element (Name?, Value?)>
<!ATTLIST Element
    %common-attributes;
```

```
    id ID #IMPLIED
    idref IDREF #IMPLIED
>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
<!ELEMENT Inferred (#PCDATA)>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT DocumentType (#PCDATA)>
```

9.2.1.2. Query message

```
<!--
    This is Query schema.
    Copyright (c) Tatsuya Inaba 2004
  -->
<!-- ========================================
                    CHANGE HISTORY

    Version          Date        Time    Author
    ============   ========   =====   ======
    Initial Draft   20040311   16:00   Inaba
    1.1                        20040318   14:00   Inaba
======================================== -->
<!ENTITY % common-elements "DateTime, Identifier, DocumentType">
<!ENTITY % common-attributes "
  dicQuery  CDATA  #REQUIRED
  name CDATA #REQUIRED
  ">
<!ELEMENT Query (%common-elements;, Action+)>
<!ATTLIST Query
    dicIdentifier CDATA #IMPLIED
>
<!ELEMENT Action (ElementSet?, Element*, Inferred?)>
<!ATTLIST Action
    %common-attributes;
>
<!ELEMENT ElementSet (ElementSet?, Element*)>
<!ATTLIST ElementSet
    %common-attributes;
>
<!ELEMENT Element (Value)>
<!ATTLIST Element
    %common-attributes;
    id ID #IMPLIED
    idref IDREF #IMPLIED
>
<!ELEMENT Value (#PCDATA)>
<!ELEMENT Inferred (#PCDATA)>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT DocumentType (#PCDATA)>
```

9.2.1.3. Response message

```
<!--
    This is Response schema.
    Copyright (c) Tatsuya Inaba 2004
   -->
<!-- ========================================================
                        CHANGE HISTORY

    Version          Date          Time     Author
    =============    ============  =======  ======
    Initial Draft    20040311      16:00    Inaba
    1.1              20040318      14:00 Inaba
========================================================== -->
<!ENTITY % common-elements "DateTime, Identifier, DocumentType">
<!ENTITY % common-attributes "
   dicQuery   CDATA   #REQUIRED
   name CDATA #REQUIRED
   ">
<!ELEMENT Response (%common-elements;, OriginalIdentifier, PhysicalObject*, Result?)>
<!ATTLIST Response
    dicIdentifier CDATA #IMPLIED
>
<!ELEMENT PhysicalObject (Action+, Result?)>
<!ATTLIST PhysicalObject
    EPC CDATA #IMPLIED
>
<!ELEMENT Action (ElementSet?, Element*, Inferred?, Result?)>
<!ATTLIST Action
    %common-attributes;
>
<!ELEMENT ElementSet (ElementSet?, Element*)>
<!ATTLIST ElementSet
    %common-attributes;
>
<!ELEMENT Element (Value)>
<!ATTLIST Element
    %common-attributes;
    id ID #IMPLIED
    idref IDREF #IMPLIED
>
<!ELEMENT Value (#PCDATA)>
<!ELEMENT Inferred (#PCDATA)>
<!ELEMENT Result (Reason?)>
<!ATTLIST Result
    category (success | fail | partly_success)   #REQUIRED
>
<!ELEMENT Reason (#PCDATA)>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT OriginalIdentifier (#PCDATA)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT DocumentType (#PCDATA)>
```

9.2.1.4. Acknowledge message

```
<!--
    This is Acknowledge schema
    Copyright (c) Tatsuya Inaba 2004
  -->
<!-- ========================================================
                        CHANGE HISTORY

    Version          Date        Time     Author
    =============    ========    =======  ======
    Initial Draft    20040318    15:00    Inaba

========================================================= -->
<!ENTITY % common-elements "DateTime, Identifier, DocumentType">
<!ENTITY % common-attributes "
  dicQuery  CDATA  #REQUIRED
  name CDATA #REQUIRED
  ">
<!ELEMENT Acknowledge (%common-elements;, OriginalIdentifier, PhysicalObject*, Result?)>
<!ATTLIST Response
    dicIdentifier CDATA #IMPLIED
>
<!ELEMENT PhysicalObject (Action+, Result?)>
<!ATTLIST PhysicalObject
    EPC CDATA #IMPLIED
>
<!ELEMENT Action (Result?)>
<!ATTLIST Action
    %common-attributes;
>
<!ELEMENT Result (Reason?)>
<!ATTLIST Result
    category (success | fail | partly_success) #REQUIRED
>
<!ELEMENT Reason (#PCDATA)>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT OriginalIdentifier (#PCDATA)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT DocumentType (#PCDATA)>
```

9.2.2. Sample Instances

9.2.2.1. Notify the scanned item

When items are scanned by the Savant, the data is sent to the server which stores the physical object data. This sample instance is used in this situation. A reader (urn:epc:1.10.101.1001) scans four items at 15:13:43-5:00 2004 February 25. In response to the Notify message, the Acknowledge message is sent back to the Savant.

Notify message

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

```
<!DOCTYPE Notify SYSTEM "Notify.dtd">
<Notify dicIdentifier="EventDictionary ver1.0">
    <DateTime>20040225T15:13:43 -0500</DateTime>
    <Identifier>200402250001</Identifier>
    <DocumentType>notify</DocumentType>
    <PhysicalObject EPC="urn:epc:1.10.100.1">
        <Action dicRef="EA001" name="Detect">
            <Element dicRef="EV001" name="EPC">
                <Value>urn:epc:1.10.100.1</Value>
            </Element>
            <Element dicRef="EV002" id="D200402250001" name="DateTime">
                <Value>20040225T15:00:00-05:00</Value>
            </Element>
            <Element dicRef="EV003" id="E1001" name="ReaderEPC">
                <Value>urn:epc:1.10.101.1001</Value>
            </Element>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.2">
        <Action dicRef="EA001" name="Detect">
            <Element dicRef="EV001" name="EPC">
                <Value>urn:epc:1.10.100.2</Value>
            </Element>
            <Element dicRef="EV002" idref="D200402250001" name="DateTime"/>
            <Element dicRef="EV003" idref="E1001" name="ReaderEPC"/>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.3">
        <Action dicRef="PA001" name="Detect">
            <Element dicRef="EV001" name="EPC">
                <Value>urn:epc:1.10.100.3</Value>
            </Element>
            <Element dicRef="EV002" idref="D200402250001" name="DateTime"/>
            <Element dicRef="EV003" idref="E1001" name="ReaderEPC"/>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.4">
        <Action dicRef="PA001" name="Detect">
            <Element dicRef="EV001" name="EPC">
                <Value>urn:epc:1.10.100.4</Value>
            </Element>
            <Element dicRef="EV002" idref="D200402250001" name="DateTime"/>
            <Element dicRef="EV003" idref="E1001" name="ReaderEPC"/>
        </Action>
    </PhysicalObject>
</Notify>
```

Acknowledge message

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Acknowledge SYSTEM "Acknowledge.dtd">
<Acknowledge>
    <DateTime>20040225T15:13:43 -0500</DateTime>
    <Identifier>200402250002</Identifier>
    <DocumentType>acknowledge</DocumentType>
```

```
    <OriginalIdentifier>200402250001</OriginalIdentifier>
    <Result category="success"/>
</Acknowledge>
```

9.2.2.2. Notify the expiration date

When the property data is inputted at the Savant, the data is sent to the server which stores the physical object data. This sample instance is used in this situation. The expiration date data is sent to the server. In response to the Notify message, the Acknowledge message is sent back to the Savant. In this case, the result of an item(urn:epc:1.10.100.4) becomes error because the EPC has not registered in the server.

Notify message

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Notify SYSTEM "Notify.dtd">
<Notify dicIdentifier="PropertyDictionary ver1.0">
    <DateTime>20040318T14:13:43 -0500</DateTime>
    <Identifier>200403180009</Identifier>
    <DocumentType>notify</DocumentType>
    <PhysicalObject EPC="urn:epc:1.10.100.1">
        <Action dicRef="PA003" name="Update">
            <Element dicRef="PV015" name="ExpirationDate">
                <Value>20040425T12:00:00-5:00</Value>
            </Element>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.2">
        <Action dicRef="PA003" name="Update">
            <Element dicRef="PV015" name="ExpirationDate">
                <Value>20040425T12:00:00-5:00</Value>
            </Element>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.3">
        <Action dicRef="PA003" name="Update">
            <Element dicRef="PV015" name="ExpirationDate">
                <Value>20040425T12:00:00-5:00</Value>
            </Element>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.4">
        <Action dicRef="PA003" name="Update">
            <Element dicRef="PV015" name="ExpiratioDate">
                <Value>20040425T12:00:00-5:00</Value>
            </Element>
        </Action>
    </PhysicalObject>
</Notify>
```

Acknowledge message

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Acknowledge SYSTEM "Acknowledge.dtd">
<Acknowledge dicIdentifier="PropertyDictionary ver1.0">
    <DateTime>20040318T15:13:43 -0500</DateTime>
    <Identifier>200403180010</Identifier>
    <DocumentType>acknowledge</DocumentType>
    <OriginalIdentifier>200403180009</OriginalIdentifier>
    <Result category="partly_success">
        <Reason>EPC not find</Reason>
    </Result>
    <PhysicalObject EPC="urn:epc:1.10.100.4">
        <Action dicRef="PA001" name="Register">
            <Result category="fail">
                <Reason>EPC not find</Reason>
            </Result>
        </Action>
    </PhysicalObject>
</Acknowledge>
```

9.2.2.3. Notify the advance ship notice

When items/cases/pallets are sent from a company to its trading partner, the company sends advance ship notice to its trading partner. This sample instance is used in this situation. First, the enterprise application of the sending company sends the Notify message to its server with EPC-IS, and the server sends the Notify message to the server with EPC-IS which is located at the trading partner's facility. In this case, the sending company packs two items in cases (urn:epc:1.10.100.10, urn:epc:1.10.100.11). The shipment document identifier also uses EPC (urn:epc:1.10.102.1) in this case.

Notify message from the enterprise application to the server with EPC-IS

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Notify SYSTEM "Notify.dtd">
<Notify dicIdentifier="EventDictionary ver1.0">
    <DateTime>20040324T15:13:43 -0500</DateTime>
    <Identifier>200403240001</Identifier>
    <DocumentType>notify</DocumentType>
    <PhysicalObject EPC="urn:epc:1.10.102.1">
        <Action dicRef="PA004" name="Notify">
            <ElementSet dicRef="PV002" name="ASN">
                <Element dicRef="PV014" name="DocumentId">
                    <Value>urn:epc:1.10.102.1</Value>
                </Element>
            </ElementSet>
        </Action>
    </PhysicalObject>
</Notify>
```

Notify message from the server of the sending company to the sever of the receiving company

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Notify SYSTEM "Notify.dtd">
```

```
<Notify dicIdentifier="EventDictionary ver1.0">
    <DateTime>20040324T16:13:43 -0500</DateTime>
    <Identifier>urn.epc:1.10.102.1</Identifier>
    <DocumentType>notify</DocumentType>
    <PhysicalObject EPC="urn:epc:1.10.102.1">
        <Action dicRef="PA004" name="Notify">
            <ElementSet dicRef="PV002" name="ASN">
                <Element dicRef="PV014" name="DocumentId">
                    <Value>urn:epc:1.10.102.1</Value>
                </Element>
            </ElementSet>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.10">
        <Action dicRef="EA001" name="Detect">
            <Element dicRef="EV001" name="EPC">
                <Value>urn:epc:1.10.100.10</Value>
            </Element>
            <Element dicRef="EV002" id="D200403241000001" name="DateTime">
                <Value>20040324T10:00:00-5:00</Value>
            </Element>
            <Element dicRef="EV003" id="D200403241000002" name="ReaderEPC">
                <Value>urn:epc:1.10.101.1001</Value>
            </Element>
        </Action>
        <Action dicRef="EA003" name="Aggregate">
            <Element dicRef="EV004" name="RelatedEPC">
                <Value>urn:epc:1.10.100.1</Value>
            </Element>
        </Action>
        <Action dicRef="EA003" name="Aggregate">
            <Element dicRef="EV004" name="RelatedEPC">
                <Value>urn:epc:1.10.100.2</Value>
            </Element>
        </Action>
    </PhysicalObject>
    <PhysicalObject EPC="urn:epc:1.10.100.11">
        <Action dicRef="PA001" name="Detect">
            <Element dicRef="EV001" name="EPC">
                <Value>urn:epc:1.10.100.11</Value>
            </Element>
            <Element dicRef="EV002" idref="D200403241000001" name="DateTime"/>
            <Element dicRef="EV003" idref="D200403241000002" name="ReaderEPC"/>
        </Action>
        <Action dicRef="EA002" name="Aggregate">
            <Element dicRef="EV004" name="RelatedEPC">
                <Value>urn:epc:1.10.100.3</Value>
            </Element>
        </Action>
        <Action dicRef="EA002" name="Aggregate">
            <Element dicRef="EA004" name="RelatedEPC">
                <Value>urn:epc:1.10.100.4</Value>
            </Element>
        </Action>
    </PhysicalObject>
```

</Notify>

9.2.2.4. Query the product size

When a buyer wants to know the size of an item, the company sends a query to the supplier. This sample instance is used in this situation. Precondition of this sample is that the buyer gets the EPC of the product of which the company wants the size. First, the buyer sends a query with the EPC of the item, and then the supplier sends back the size of the item to the buyer.

Query message

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Query SYSTEM "Query.dtd">
<Query dicIdentifier="PropertyDictionary ver1.0">
    <DateTime>20040311T14:13:43 -0500</DateTime>
    <Identifier>200402250006</Identifier>
    <DocumentType>query</DocumentType>
    <Action dicQuery="PA001" name="Create">
        <ElementSet dicQuery="PVS001" name="Manufacturer">
            <Element dicQuery="PV007" name="BusinessEntity">
                <Value/>
            </Element>
        </ElementSet>
        <Element dicQuery="PV003" name="Height">
            <Value/>
        </Element>
        <Element dicQuery="PV004" name="Depth">
            <Value/>
        </Element>
        <Element dicQuery="PV005" name="Width">
            <Value/>
        </Element>
        <Element dicQuery="PV006" name="Weight">
            <Value/>
        </Element>
        <Element dicQuery="PV001" name="EPC">
            <Value>urn:epc:1.10.100.1</Value>
        </Element>
    </Action>
</Query>
```

Response message

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Response SYSTEM "Response.dtd">
<Response dicIdentifier="PropertyDictionary ver1.0">
    <DateTime>20040311T14:13:43 -0500</DateTime>
    <Identifier>200402250007</Identifier>
    <DocumentType>response</DocumentType>
    <OriginalIdentifier>200402250006</OriginalIdentifier>
    <PhysicalObject EPC="urn:epc:1.10.100.1">
        <Action dicQuery="PA001" name="Create">
            <ElementSet dicQuery="PVS001" name="Manufacturer">
                <Element dicQuery="PV007" name="BusinessEntity">
```

```
                <Value>MIT</Value>
            </Element>
        </ElementSet>
        <Element dicQuery="PV003" name="Height">
            <Value>0.3</Value>
        </Element>
        <Element dicQuery="PV004" name="Depth">
            <Value>0.1</Value>
        </Element>
        <Element dicQuery="PV005" name="Width">
            <Value>0.2</Value>
        </Element>
        <Element dicQuery="PV006" name="Weight">
            <Value>1.0</Value>
        </Element>
        <Element dicQuery="PV001" name="EPC">
            <Value>urn:epc:1.10.100.1</Value>
        </Element>
    </Action>
</PhysicalObject>
</Response>
```

9.2.2.5. Query the Expiration Date

When a buyer wants to know the expiration date of an item, the company sends a query to the supplier. This sample instance is used in this situation. Precondition of this sample is that the buyer gets the EPC of the product of which the company wants the expiration date. First, the buyer sends a query with the EPC of the item, and then the supplier sends back the expiration date of the item to the buyer.

Query message

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Query SYSTEM "Query.dtd">
<Query dicIdentifier="PropertyDictionary ver1.0">
    <DateTime>20040311T14:13:43 -0500</DateTime>
    <Identifier>200402250006</Identifier>
    <DocumentType>query</DocumentType>
    <Action dicQuery="PA001" name="Create">
        <Element dicQuery="PV001" name="EPC">
            <Value>urn:epc:1.10.100.2</Value>
        </Element>
        <Element dicQuery="PV015" name="ExpirationDate">
            <Value/>
        </Element>
    </Action>
</Query>
```

Response message

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Response SYSTEM "Response.dtd">
<Response dicIdentifier="PropertyDictionary ver1.0">
    <DateTime>20040311T15:13:43 -0500</DateTime>
```

```
    <Identifier>200402250007</Identifier>
    <DocumentType>response</DocumentType>
    <OriginalIdentifier>200402250006</OriginalIdentifier>
    <PhysicalObject EPC="urn:epc:1.10.100.2">
        <Action dicQuery="PA001" name="Create">
            <Element dicQuery="PV001" name="EPC">
                <Value>urn:epc:1.10.100.2</Value>
            </Element>
            <Element dicQuery="PV015" name="ExpirationDate">
                <Value>20040425T12:00:00-5:00</Value>
            </Element>
        </Action>
    </PhysicalObject>
</Response>
```