9

Advantages and New Applications of DHT-Based Discovery Services in EPCglobal Network

Juan Pedro Muñoz-Gea, Pilar Manzanares-Lopez and
Josemaria Malgosa-Sanahuja
Department of Information Technologies and Communications
Polytechnic University of Cartagena
Spain

1. Introduction

Radio Frequency IDentification (RFID) technology can track the movement of products throughout an entire supply network by giving a Globally Unique Product Identifier (GUPI) (Främling et al., 2007) to every product. The identification is used to connect the physical object to an information service run by a host on the Internet. It is there that all the information associated to that physical object is stored. Nowadays, several approaches are used to implement this connection, such as the EPCglobal Network (Armenio et al., 2009) developed by the Auto-ID consortium, the DIALOG system (Främling et al., 2006) developed at the Helsinki University of Technology and the World Wide Article Information (WWAI) system[1] proposed by Trackway. The EPCglobal Network stands out among the rest because in 2003 it was authorized as a Global Standards I (GS1). The GS1 system of standards is the most widely-used supply-network standards system in the world, the traditional barcode being its most widely used standard.

1.1 Product identifier proposals

The DIALOG system was developed at the Helsinki University of Technology. In this approach an ID@URI notation is used to create a GUPI, where the ID part identifies the product item located at the URI. If the URI is an URL, it is a straightforward task to link it to an information service. For an ID@URI to be a GUPI, the ID part should be unique for the corresponding URI. In the DIALOG system every product is implemented as a software agent (Nwana, 1996), and the information of each of them is accessed and updated through methods in the product agent interface (Främling et al., 2006). These interface methods are as follows: *update()* and *getProductInformation()*, which are used to append and retrieve information, respectively, and *getCompositeInformation()*, which relates to managing component hierarchies. The World Wide Article Information (WWAI) protocol developed by Trackway (formerly known as Stockway) is based on P2P principles. The manufacturers form a network of nodes which are identified by company numbers. When a node has joined the network, it can autonomously issue identifiers for individual products, the product GUPI consisting of a concatenation of the company prefix and item-specific suffix. The service provided by

[1] http://www.trackway.eu

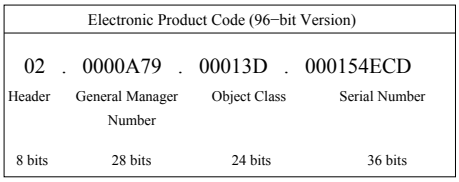


Fig. 1. EPC tag data structure.

the WWAI network is the mapping of the WWAI identifiers to the URL of the information provider, using a DHT (Distributed Hash Table). A lookup is only needed when a product identifier for a given company is seen for the first time. After that, URLs of known nodes are cached so that new node lookups do not need to be performed unless the cached address fails or changes for some reason.

The common characteristic of the two previous architectures is that all the data related to a specific product is available in only one information service. This means that every organization with information about a specific product has to publish it in the corresponding information service and then, the queries have to be addressed to a specific information service depending on the specific GUPI. On the other hand, one of the fundamental design principles for the EPCglobal Network is that each company retains control over the data that they collect or generate within their own organization, i.e. information about a specific product is decentralized across multiple organizations.

The EPC[2] serves the role of a GUPI in the EPCglobal Network. There are two different lengths of EPC tag data that have been ratified by the EPCglobal board: 64 bits and 96 bits. Figure 1 shows the EPC tag data structure of one 96-bit version: Header identifies the version of EPC itself; General Manager Number identifies an organization that maintains the numbers in the field of Object Class and Serial Number; Object Class refers to a unique type of products produced by an EPC general manager; Serial Number uniquely identifies each item within an object class. The EPC serial number allows individual items tracking, which is the key feature and advantage that distinguish EPC from a bar code standard. This feature enables RFID technology to capture the information about the movement of individual products in supply networks. Next section explains the rest of components of the EPCglobal Network architecture in more depth.

1.2 EPCglobal Network

One of the fundamental design principles for the EPCglobal Network is that each company should be able to keep control of the data that they collect or generate about a specific product within their own organization, i.e. information is decentralized across multiple organizations. With a suitable service-oriented architecture the EPC can be used both to locate a source of information, via lookup services, as well as for extracting relevant information about a particular product from each source, by using the EPC as a lookup key within a database.

The EPCglobal Network architecture includes specifications that deal with the collection of captured data and their distribution across organizations. However, the initial standards development has focused more on the EPCglobal Network's lower levels than on data exchange across supply networks. Figure 2 represents the EPCglobal Network architecture framework.

[2] http://www.epcglobalinc.org/standards/tds/tds_1_4-standard-20080611.pdf

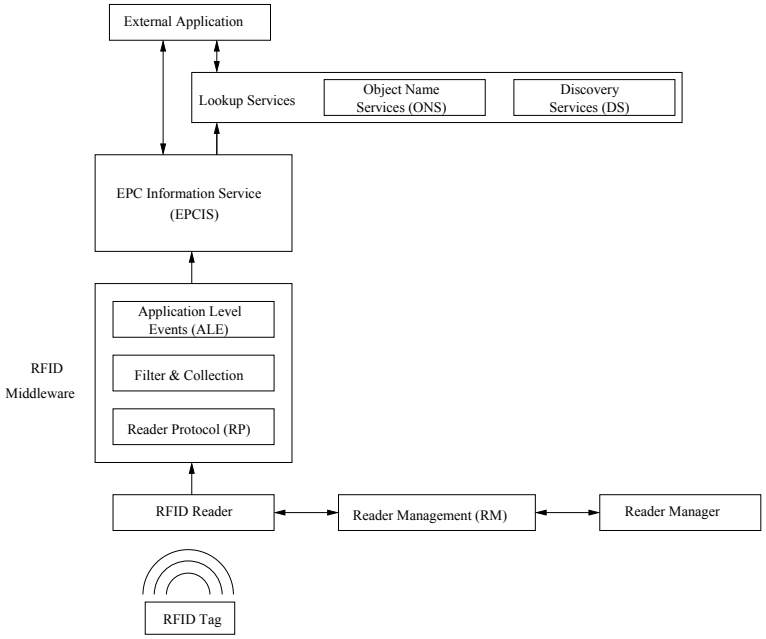


Fig. 2. EPCglobal Network Architecture.

The reader protocol (RP) is an API that abstracts from the underlying RFID hardware readers. RP is complemented by the reader management (RM) protocol, which facilitates the naming of readers and locations. In this context, it is necessary to take into account the fact that the enormous quantity of data generated by even a few readers can easily result in an unacceptable load. For this reason, an additional middleware layer is needed that filters the data collected from readers. In the EPCglobal Network, this task is supported by the application-level events (ALE) interface. ALE allows for the conversion of a series of tag reads into a single event message with a time interval attribute. In addition, the ALE standard also supports the declaration of logical readers that collect the data streams from multiple physical readers.

RFID readers are distributed across factories, warehouses and stores. In this context, EPCIS provide a repository of historical events and related information, and they are fed by an EPCIS capturing application, such as a management system connected to an ALE middleware. To find EPC-related information, business applications use the Object Name Service (ONS) to provide an EPCIS URL when queried with a tag's EPC. The granularity of ONS resolution is currently limited to product type, i.e. an ONS is not expected to retain distinct records for two objects of the same product type. Another point to note is that ONS is currently implemented using the Domain Name System (DNS) (Mockapetris, 1987), using Type 35 Naming Authority Pointer (NAPTR) (Mealling & Daniel, 2000) records to return the information. Queries to ONS are therefore performed by means of a DNS query for a hostname derived from an EPC. However, the ONS is allowed only to point to the manufacturer's EPCIS repository. For this reason, the EPCglobal Network is being extended to include Discovery Services (DS)[3]. These

[3] http://www.epcglobalinc.org/standards/discovery

services will allow applications to find third parties' EPCIS repositories with events related to a specific EPC. A critical component of the DS is the data storage component, which stores information about the list of EPCIS instances that have information about a particular EPC. There currently exist several options to implement this component, although two in particular are worthy of note, LDAP (Lightweight Directory Access Protocol) and DHT (Distributed Hash Table).

2. Distributed Hash Tables (DHT)

DHTs are decentralized distributed systems that distribute the management of a key table between the participating nodes. Each node maintains a routing table with a list of its neighbors, so that it can route messages to the unique owner of a given key. DHT is designed to scale to a large number of nodes and is prepared to deal with continuous node arrivals and failures by constructing a structured P2P (peer-to-peer) network.

There currently exist several structured P2P overlay network proposals such as Chord (Stoica et al., 2003), Pastry (Rowstron & Druschel, 2001), Tapestry (Zhao et al., 2004) and Kademlia (Maymounkov & Mazières, 2002). The difference among them is the way in which nodes are organized in the overlay network. In Chord, nodes are organized in a ring. However, in Pastry, Tapestry and Kademlia nodes are organized in a tree structure (Balakrishnan et al., 2003).

As an example, we are going to present an overview of the Pastry operation. The node identifiers and the routing procedure in Pastry depend on the base used to represent both node and key identifiers. For example, if a hexadecimal base is used and the system requires five digits to represent the full node space, an example of a node identifier could be 65A1F. Each node maintains a routing table formed by as many rows as digits needed to represent the full node space and as many columns as possible digit values (in our example each routing table is formed by 5 rows and 16 columns). Row 0 maintains the IP addresses corresponding to nodes without a common prefix with the local node identifier. Row 1 contains the IP addresses corresponding to nodes with a 1 digit-long prefix common to the local node identifier, and so on. As we have seen, a row is formed by some columns. The *i*-th column of a row stores, if it exists, the IP address of a node whose identifier is determined by the associated row prefix followed by the *i* digit. Please note that there could be more than one node that satisfies the requirements of an entry. If so, the Pastry implementation will use some predefined criterion to select an entry (for example, the node with lowest physical delay). In our example, row 0 stores the IP addresses of nodes without a common prefix with 65A1F: The first entry at row 0 stores the IP address of a node whose identifier is 0**** (if it exists), the second entry stores the IP address of anode whose identifier is 1**** (if it exists), and so on. Row 1 stores the IP addresses of a maximum of 16 nodes, with one digit-length common prefix: the first entry stores the IP address of a node whose identifier is 60*** (if it exists), the second entry stores the IP address of a node whose identifier is 61***, and so on. It must be emphasized that if the local node does not know of a suitable node to fill a concrete entry in the routing table, that entry is left empty.

A requester node elects from its routing table the node matching the longest prefix with the key. The lookup message is sent to that node, and then it repeats the same algorithm until the searched node is found. During this process, a required entry in the routing table might be empty. In this case, the node uses the closest numerical non-empty entry. The routing table definition assures the convergence of the lookup algorithm. In our example, if the 65A1F node is searching for the key 654B2, the longest common prefix in the routing table is 654** (that is

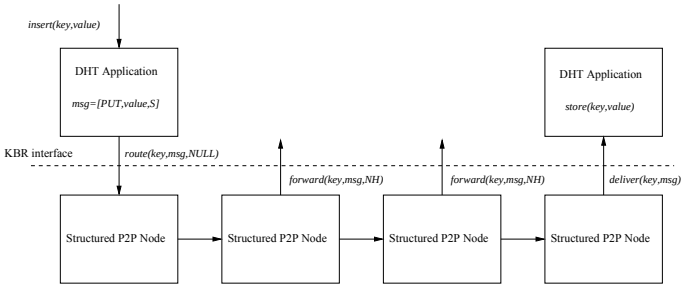


Fig. 3. *insert()* operation.

to say, the entry associated to row 2 and column 5). The local node uses the IP address stored in this entry (or the numerically closest node if the entry is empty) to forward the lookup message.

In general, structured P2P networks offer a routing service to send a message, with an associated key, to a single node, responsible for that key. For this reason, this service is called Key-Based Routing (KBR). There exists a common API (Dabek et al., 2003) with the ability needed to interact with the KBR service offered by any structured P2P network. These are as follows: *void route(key k, msg m)*, which forwards a message, *m*, towards the responsible key node *k* (it is implemented in the structured P2P network layer); *void forward(key k, msg m, nodehandle nexthopnode)*, which is invoked from the structured P2P network layer of each node that forwards message *m* during its routing (this function is implemented in the application layer); *void deliver(key k, msg m)*, which is invoked from the structured P2P network layer of the node that is responsible for key *k* upon the arrival of message *m* (this function is also implemented in the application layer).

Over the previous interface (in the application layer), the services offered by a DHT application can be implemented. The DHT abstraction provides the same functionality as a traditional hash table, by storing the mapping between a *key* and a *value*. That is, it implements a simple store and retrieve functionality, where the value is always stored at the overlay node to which the key is mapped by the KBR layer. This application provides two operations: *insert(key, value)*, and *value = lookup(key)*. A simple implementation of *insert()* routes a PUT message containing value and the *local node?s nodehandle* (S), using *route(key, [PUT,value,S], NULL)*. The node responsible of that *key*, upon receiving the message, stores the *(key, value)* pair in its local storage. On the other hand, the lookup operation routes a GET message using *route(key, [GET,S], NULL)* to the node responsible of that *key*, and it returns the associated *value* in a single hop using *route(NULL, [value], S)*. The single hop routing option uses the *nodehandle* of the source node (S) to send the message directly to that node, without hoping along other nodes in the network. Figures 3 and 4 represent the *insert()* and *lookup()* operation.

There exist several free software implementations of structured P2P networks, like Chimera[4] (a C implementation of Tapestry), the Chord project[5] (a C implementation of Chord) and FreePastry[6] (a Java implementation of Pastry). From among these FreePastry has been selected because it is implemented in Java and because it is an active project in development at the MPI-MPG[7], although it initially started at Rice University in USA.

[4] http://current.cs.ucsb.edu/projects/chimera
[5] http://pdos.csail.mit.edu/chord
[6] http://www.freepastry.org/FreePastry/
[7] http://www.mpi-inf.mpg.de

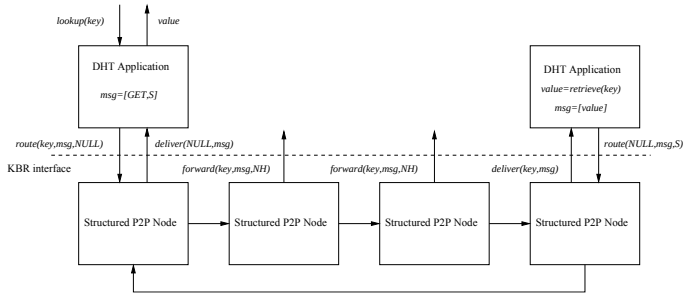


Fig. 4. *lookup()* operation.

3. Advantages of DHT-based discovery services

Recently, there have been several proposals for the implementation of the DS, like the DS prototype of the Bridge project[8], or the Extensible Supply-chain Discovery Service (ESDS) developed by Afilias (Young, 2008). In both the DS has been implemented as a centralized database, to be more specific in the Bridge Project it has been developed as a centralized database based on LDAP (Lightweight Directory Access Protocol) (Zeilenga, 2006). In this work, we are going to develop a DS prototype based on DHT.

LDAP is a networking protocol for querying and modifying directory services running over TCP/IP. In this context, a directory is a set of information with similar attributes organized in a hierarchical manner. LDAP could be thought of as type of database, different from a relational database. Databases are usually designed to perform many changes to their data whereas LDAP directories are optimized to read performance, as such, it is particularly useful for storing information that needs to read from many locations. On the other hand, LDAP permits secure delegation of reading and modification authority based on specific needs using ACLs (Access Control Lists); although this is not part of the LDAP protocol, many implementations offer this feature. The integration of the DS records into the data structure of the LDAP is based on the decomposition of the EPC tag into the LDAP tree. That is, fields such as Company Prefix, Item Reference or Serial Number can be used to distribute the EPC among the tree structure. In addition, security can be integrated with the fine grained access control policies of the LDAP implementation and can be used to limit the access to a particular record.

On the other hand, one of the characteristics of the DHT model is its decentralization, meaning that there is no central coordinator in the nodes. For this reason, the failure of a node never compromises the whole system. Another advantage is that nodes are organized in a way that a node only needs to coordinate with a few other nodes in the system (usually $log\ N$, with N participants). Therefore, if a node joins or leaves the system, this does not affect the whole system. These features provide the system with high scalability and fault tolerance. The integration of DS records into the data structure of the DHT is based on the use of a SHA1 hash of the EPC as the key for the hash table. However, in these systems, security options like authentication or fine grained access control have to be implemented.

As a conclusion, DHTs might be said to have several advantages over LDAP: in LDAP there is a bottleneck in the root of the architecture, whereas in the DHT the failure of one node does not affect the whole system; LDAP is not optimized for massive update operations, while DHTs are optimized for massive search and update operations. On the other hand,

[8] http://www.bridge-project.eu

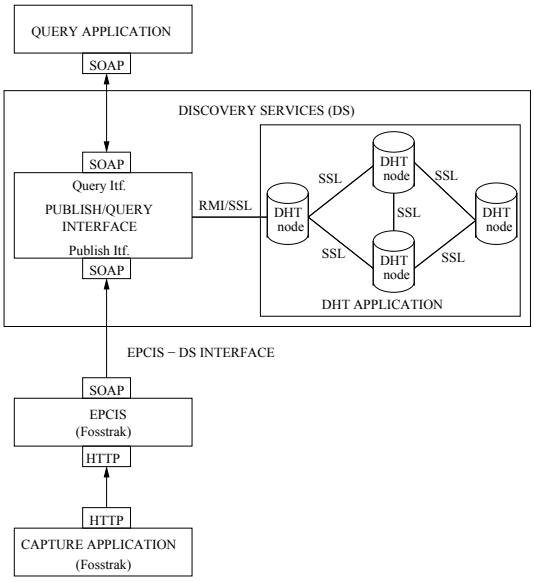


Fig. 5. Discovery Services architecture.

DHTs have an important drawback with respect to LDAP: while in LDAP the access control is already implemented by Access Control Lists (ACLs) in DHT fine grained controls have to be implemented.

4. DHT-based discovery services prototype

The objective of the designed DS is to return a time-ordered list of links to multiple ECPIS instances which hold information related to a specific EPC. Therefore, the DS is designed to create this list of links. The architecture of the full prototype has been divided into a set of logical components represented in Figure 5.

4.1 Discovery services

The DS implementation is composed by two components: the DHT application and the Publish/Query interface.

4.1.1 DHT application

The public methods (*insert()* and *lookup()*) of every DHT node in our implementation can be accessed by a remote application using Java RMI[9], which is a Java API which performs the object-oriented equivalent of remote procedure calls (RPC). The prototype of the previous methods is as follows: *public Boolean insert(MessageDigest key, String url)* and *public String[] lookup(MessageDigest key)*. The insert method has two parameters: *String url* represents the URL of the EPCIS query interface with information about a specific EPC; *MessageDigest key* represents the SHA1 hash (Eastlake & Jones, 2001) of the related EPC. This method returns a *Boolean* which indicates if the insert operation has been properly finished. On the other hand,

[9] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

the lookup method has a single parameter: the SHA1 hash of an EPC, and it returns an array with all the associated URLs.
The *insert()* method of the DHT node creates a content object with the *key*, the URL and the its own nodehandle. After that, it inserts the previous content in a message with the following fields: the identifier of the origin node, the identifier of the destination node (which corresponds with the *key* of the message), the message type (for an INSERT message corresponds type 0) and the previously created content. Once the message has been created, the insert method calls the *route()* function with the previously created message.
The previous message is routed to the node responsible of the message's key. Then the FreePastry software will call the *deliver()* implementation of our application. This method extracts the type of the message, and if it is an INSERT message it extracts the *key* (the SHA1 hash of the EPC) and the URL and it inserts both of them in a MySQL[10] database. The communication between our Java application and the database is possible thanks to the JDBC[11] driver. Our database only has one table, with two columns: one for the key (SHA1 hash of the EPC) and the other for the URL. Multiple insertions of the same key, each one of them with a different URL, are allowed. By this way, the database stores the different URLs of the EPCISs with information about a specific EPC. The operation of the *lookup()* method is equivalent to the *insert()* operation, but in this case the destination node extracts from the database all the URLs associated to the key of a specific EPC.

4.1.2 Publish/query interface

This interface has been implemented as a Web Service[12] using two methods available thorugh SOAP[13] operations: *publish()* and *query()*, and it has been deployed in a Glassfish application server[14]. There are currently several frameworks to program Java Web Services, but the two most important are Apache Axis2[15] and Sun JAX-WS[16]. The Java API for XML based Web Services (JAX-WS) is the successor of the JAX-RPC specification. Its configuration is managed by annotations[17], therefore Java 5 or higher is required. With JAX-WS it is relatively easy to write and consume web services. The default values of numerous parameters are comfortable from the point of view of the programmer and simple methods declared with a @WebService annotation can be used as a service. A suitable WSDL document can also be generated from the class.
The *publish()* method of the web service implementation has two parameters: the SHA1 hash of the EPC and the URL where the information related to the EPC is available. The web service has previously been configured with the URL where the RMI stubs of all the DHT nodes are available. Therefore, the web service selects one of the DHT nodes of the network, it obtains its RMI stub and, after that, it calls the *insert()* method of the DHT application. After calling the *insert()* method, the web service receives if the insert operation has been performed in a correct way or not. Therefore, this interface is only a proxy between the EPCIS repository of the query application and the DHT nodes.

[10] http://www.mysql.com
[11] http://dev.mysql.com/downloads/connector/j/5.1.html
[12] http://www.w3.org/standards/webofservices/
[13] http://www.w3.org/TR/soap/
[14] https://glassfish.dev.java.net/
[15] http://ws.apache.org/axis2/
[16] https://jax-ws.dev.java.net/
[17] http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

The *query()* method is similar to that previously mentioned, but in this case it only accepts one parameter: the SHA1 hash of an EPC. This method selects one of the DHT nodes and it calls the RMI *lookup()* method of the DHT application. After that, the web service receives the URLs associated to all the EPCISs with information about that EPC. All this information will be returned to the application which called the *query()* method.

4.2 EPCIS-DS interface

In the first place, we have deployed the Fosstrak EPCIS software[18] in a host with a Glassfish application server. The Fosstrack EPCIS software is a complete implementation of the EPCIS standard specification (Version 1.0.1 of September 21, 2007) and it allows users to deploy an EPCIS repository. In addition, Fosstrak developers also provide an interactive EPCIS capture application which allows users to fill the EPCIS repository with EPC data using a graphical user interface. In our prototype, we have used this graphical tool to insert events into the EPCIS Repository.

The current EPCIS standard does not include a specific communication mechanism between an EPCIS and a Discovery Service. For this reason, we have developed a module to be integrated into the Fosstrak software. Its goal is to send the association between an EPC, with information registered in that EPCIS repository, and the URL to query that information to the DS, but only once. That is, it is possible that the EPCIS registers several events associated to the same EPC because it is possible that it has several associated RFID readers. Therefore, our module has to assure the DS stores only one association between that EPC and the public URL of the EPCIS. We have studied the EPCIS software implementation and we have detected that it has two independent modules: the EPCIS capture interface and the repository, implemented in a MySQL database. The communication among them is by means of the TCP protocol, and the SQL queries are sent to the 3306 TCP port of the localhost. SQL queries are sent in plain text; therefore, we decided to develop a traffic sniffer to detect the SQL queries sent to the MySQL database.

The sniffer module has been developed in Java language, using the jNetPcap[19] software development kit. The basic function of jNetPcap is to provide a java wrapper to popular libpcap library[20] for capturing network packets. Our module captures the packets in the loopback interface, because the TCP communication between the EPCIS capture interface and the MySQL database is in the localhost. But it does not capture all the traffic; we have implemented a filter in order to detect the SQL queries with EPCs. After the filter detects one of these SQL queries in a specific packet it will be redirected to a special method to process it. This method inspects the full SQL query to obtain the associated EPC. After detecting the EPC, our program checks if the detected EPC has been observed previously. In the case that it has not been detected, that is, it is the first time that information associated to that EPC is registered in the EPCIS repository, the program calls the *publish()* method of the publish/query web service interface.

4.3 Query application

This application allows a user to make real queries to the DS deployed. It accepts an EPC number and returns necessary information to reconstruct the supply network. In order to do

[18] http://www.fosstrak.org
[19] http://jnetpcap.com
[20] http://www.tcpdump.org

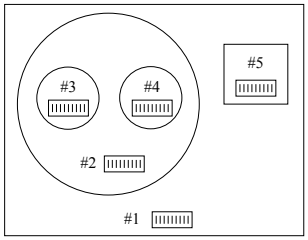


Fig. 6. A product structure.

that, the application implements a web service client which interacts with the *query()* method of the publish/query web service interface.
The last step in the supply network reconstruction process is the graphic representation of the received data, so that the information can be easily understood and interpreted by the final user. This functionality has to be added to the query application. It will depends on the final application developed using the DHT-based Discovery Service Architecture.

5. New Applications of DHT-based discovery services

5.1 Automatic traceability

5.1.1 Introduction

Suppliers, manufacturers, distributors and retailers are typically interconnected within networks and it is for this reason that the relationships among them are represented as supply networks (Wareham et al., 2005),(Phillips et al., 2006),(Poulin et al., 2006),(Li & Chandra, 2007). In this respect, supply chains are special types of supply networks in which organizations are organized in linear chains. However, many organizations do not have sufficient information about the full supply networks in which they are involved, for example, they do not know who supplies their suppliers because they are not directly connected to them. This information would be very useful in planning strategies or in assuring product quality. For example, in order to solve some problems of delayed shipments from suppliers, the organization might need to analyze its full supply network to identify which supplier is most to blame for the delay. Unfortunately however, many organizations do not have access to this information.
In this work, we propose a mechanism for automatically obtaining the supply network associated to a specific product using the EPCglobal Network. In (Bi & Lin, 2009) the authors proposed a methodology with the same objective but the main difference with the proposal set out in this article is that in (Bi & Lin, 2009) the client has to do all the operations to reconstruct the supply network. That is, initially, it has to obtain the URLs (Uniform Resource Locator) of the information services with information about a specific product, then it has to access the corresponding information services to get the necessary information, and after that it can reconstruct the full supply network. However, in our proposal the client does not have to perform any operations, that is, it queries the supply network associated to a specific product and the results are obtained directly.
A concrete example (extracted from (Bi & Lin, 2009)) is used to demonstrate how a supply network can be mapped using our methodology. Suppose that a retailer A sells a product that is assembled by a manufacturer B. Every product item and each of its components are given an EPC tag. The structure of the product with five EPC tags is shown in Figure 6. Note that component #2 is a sub-assembly that consists of two smaller components.

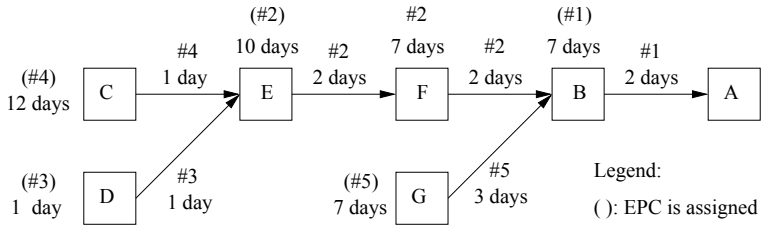


Fig. 7. Supply network associated to EPC #1.

After receiving a several requests to repair component #4, retailer A decides to map its supply network for this product in order to redirect the requests for repairs to the company that produces component #4. Retailer A is also interested in calculating the amount of time taken for products to flow between entities in this supply network. The full supply network that retailer A wants to reconstruct is represented in Figure 7.

Now, the necessary information to reconstruct the previous supply network needs to be deduced. Firstly, we concentrate on the events that assign an EPC to a new product (ASSIGN events, they are represent by ( ) in the figure). These events can be divided in two different sub-types. The first of which corresponds to the assignment of EPCs #3, #4, and #5 to the corresponding products. They are not all composed of other products, that is, they do not all have smaller components. The necessary information to represent the previous events is the following: #EPC, Company, Event=ASSIGN and Date. On the other hand, the assignment of EPCs #1 and #2 is different from the previous one, because in this case the new products are composed of several components with an assigned EPC. In this case, the necessary information to represent these events is similar to the previous one with the difference that the identity of the components of the product is added. That is, it will be: #EPC, Company, Event=ASSIGN, Date and #EPC of Components.

Next, we concentrate on the flow of products. To represent this we need to deduce the amount of time for the flow of products from one company to another company and the direction of the flow. The previous information can be deduced if we extract the exact date on which the product is sent from the company of origin (SHIP event) to a specific destination company, and the exact date on which the product is received by the destination company (RECEIVE event). In order to represent the SHIP event we need the following information: #EPC, Company, Event=SHIP, Date and Destination. On the other hand, to represent the RECEIVE event we only need: #EPC, Company, Event=RECEIVE and Date. In conclusion, in order to reconstruct a full supply network we need a time-ordered list of ASSIGN, SHIP and RECEIVE events with the necessary information to represent them all.

In order to obtain the previous information, it is necessary to send a query to the query interface of the Foostrak EPCISs with information about a specific EPC, asking for some information which depends on the event (ASSIGN, SHIP or RECEIVE) which we want to identify.

For an ASSIGN event without components, it is necessary to ask for the *date* of an *ObjectEvent* with *action=ADD* associated to a specific EPC. For an ASSIGN event with components, it is necessary to ask for the *date* and the *childEPCs* of an *AgregationEvent* with *action=ADD* associated to a specific EPC. For a SHIP event it is necessary to ask for the *date* and the *purchase order* associated to the *business transaction* of an *ObjectEvent* with *action=OBSERVE* and *businessStep=SHIPPING* associated to a specific *EPC*. In order to guess the destination company of a SHIP event we are going to use the purchase order associated to the business

Event	eventType	EQ_action	3rd Parameter
ASSIGN	ObjectEvent	ADD	MATCH_epc = urn:epc:id:sgtin:0034000.987650.2686
ASSIGN with comp.	AggregationEvent	ADD	MATCH_parentID = urn:epc:id:sscc:0614141.1234567890
SHIP	ObjectEvent	OBSERVE	EQ_bizStep = urn:epcglobal:cbv:bizstep:shipping
RECEIVE	ObjectEvent	OBSERVE	EQ_bizStep = urn:epcglobal:cbv:bizstep:receiving

Table 1. Parameters to query the ASSIGN, SHIP and RECEIVE events.

transaction. This purchase order includes the identities of the buyer and the supplier, the product related to the purchase order and the quantity of the ordered product. Finally, for a RECEIVE event, it is necessary to ask for the *date* of an *ObjectEvent* with *action=OBSERVE* and *businessStep=RECEIVING* associated to a specific *EPC*.
The query interface of the Foostrak EPCISs is provided by means of a web service; therefore, it is necessary to program a client web service to ask for the previous information.Table 1 represents the parameters to query the previous ASSIGN, SHIP and RECEIVE events.
The *poll()* method returns an array with the following information: Event, occurred, recorded, Parent ID, Quantity, EPCs, Action, Business step, Disposition, ReadpointId, Business location, Business transaction. Therefore, we only have to go to the corresponding column to obtain the necessary information.

5.1.2 Implementation

In this section, we are going to present the integration of a supply networks discovery mechanism within our DHT-based DS prototype. We have added a new public method to our DHT application, called *supply_network()*, which can also be accessed by a remote application using Java RMI. The prototype of the previous method is the following: *public Object[][] supply_network(MessageDigest key)*. This method has a single parameter, the SHA1 hash of an EPC, and it returns a bi-dimensional array of *Objects*. The Java *Object* class sits at the top of the class hierarchy tree in the Java development environment, that is to say, every class in the Java system is a descendent of the *Object* class. The bi-dimensional *Object* array has all the necessary information to represent the supply network. That is, the columns have these values: #EPC, Company, Event (ASSIGN, RECEIVE, SHIP), Data, Destination Company (for SHIP events) and Components (for ASSIGN events with different sub-products). In addition, every row represents an event (ASSIGN, RECEIVE or SHIP) associated to a specific EPC.
We have also added a new method to the web service publish/query interface called *network()*. The *network()* method is similar to the *query()* method. It only accepts one parameter: the SHA-1 hash of an EPC. This method selects one of the DHT nodes and it calls the RMI *supply_network()* method of the DHT application. After that, the web service receives the previous *Object* bi-dimensional array, with all the necessary information to represent the supply network. All this information will be returned to the application which called the *network()* method.
The *supply_network()* method of the DHT node creates a content object with the key and its own nodehandle. After that, it inserts the previous content in a message with the following fields: the identifier of the origin node, the identifier of the destination node (which corresponds to the key of the message), the message type (for a SUPPLY_NETWORK message corresponds to type 4) and the previously created content. Once the message has been created, the *supply_network()* method calls the *route()* function with the previously created message. The *supply_network()* method also uses the *continuations* functionality offered by FreePastry.

```
IF URL[].length == 1 THEN
    ASSIGN_function(URL)
ELSE
    FOR each component of URL[]
        IF URL is the last
            URL.poll(RECEIVE)
        ELSEIF URL is the first
            URL.poll(SHIP)
            ASSIGN_function(URL)
        ELSE
            URL.poll(SHIP)
            URL.poll(RECEIVE)
        ENDIF
    ENDFOR
ENDIF

FUNCTION ASSIGN_function(URL)
    COMPONENTS[]=URL.poll(ASSIGN with components)
    IF COMPONENTS[].length > 0
        FOR each COMPONENT
            supply_network(COMPONENT.\#EPC)
        ENDFOR
    ELSE
        URL.poll(ASSIGN without components)
    ENDIF
RETURN
```

Table 2. Algorithm of the *supply_network()* method.

The previous message is routed to the node responsible for the message's key. Then the FreePastry software calls the *deliver()* implementation of our application. This method extract the type of the message, and if it is a SUPPLY_NETWORK message it extracts the key of the EPC, and after that it extracts from the database all the URLs associated to the key of a specific EPC. It is necessary to take into account that the URLs are obtained in a time order. That is, the first extracted URL corresponds with the first EPCIS which registered information about the corresponding EPC. Then, the application follows the reconstruction algorithm presented in Table 2.

As an example, we present the operations performed by the DHT application in order to reconstruct the supply network associated to the previously presented product. In this process, we assume that the DHT node responsible for storing the URLs of the EPCIS with information about the EPC #x is the DHTx node (e.g., DHT1 has the URLs with information about the EPC #1). Therefore, the call to the *supply_network()* method with EPC #1 as a parameter, gets to DHT1. This node has two URLs associated to EPC #1, [URLB, URLA]. The application takes the last URL (URLA) and it sends a query to the query interface of the corresponding EPCIS asking for a RECEIVE event. After that, it receives the necessary information to construct an *Object* array. That is, it constructs [#1; RECEIVE; A; 10/02/09; NULL; NULL]. Then, the application takes the first URL (URLB), it sends a query asking for a SHIP event, with the received information it construct the *Object* array [#1; SHIP; B, 08/02/09; A; NULL] and it adds this array to the previously constructed *Object* array. Immediately, the application sends a new query to the same URL asking for an ASSIGN event with components, with the received information it construct the *Object* array [#1; ASSIGN; B; 01/02/09; NULL; #2, #5], and it adds this array to the previously array. Finally, the application calls the *supply_network()* method taking EPC #5 as a parameter. When it receives all the information related to this EPC, it will call the *supply_network()* method taking EPC #2 as a parameter. The operation mode of these two *suply_network()* calls is equivalent to the presented one and, as a result, both of them return an *Object* bi-dimensional array, with all the information associated to the corresponding sub-supply networks associated to EPC #2 and EPC #5. All this information is added to the previously constructed *Object* bi-dimensional array (with the information about EPC #1) and it is returned to the *network()* web service

EPC	EVENT	COMPANY	DATE	TO	CONTAINS
#1	ASSIGN	B	01/02/09	—	#2,#5
#1	SHIP	B	08/02/09	A	—
#1	RECEIVE	A	10/02/09	—	—
#2	ASSIGN	E	10/01/09	—	#3,#4
#2	SHIP	E	20/01/09	F	—
#2	RECEIVE	F	22/01/09	—	—
#2	SHIP	F	29/01/09	B	—
#2	RECEIVE	B	31/01/09	—	—
#5	ASSIGN	G	13/12/08	—	—
#5	SHIP	G	20/12/08	B	—
#5	RECEIVE	B	23/12/08	—	—
#3	ASSIGN	D	10/12/08	—	—
#3	SHIP	D	11/12/08	E	—
#3	RECEIVE	E	12/12/08	—	—
#4	ASSIGN	C	08/12/08	—	—
#4	SHIP	C	20/12/08	E	—
#4	RECEIVE	E	21/12/08	—	—

Table 3. Information to reconstruct the supply network associated to EPC #1.

method. The information contained in this *Object* bi-dimensional array corresponds to the information presented in Table 3. Figure 8 represents the interaction among the different DHT notes involved in the reconstruction of this supply network.
In addition, every DHT node which receives a SUPPLY_NETWORK query stores temporarily in a new database all the deduced information (contained in the *Object* bi-dimensional array), and it associates all this information to the corresponding EPC. By this way, subsequent calls to the *supply_network()* method will be resolved in a shorter period of time. Finally, the access control service is also used by this mechanism. The PEP located in the DHT node (responsible for storing the URLs associated to the EPC included in the *supply_network()* method) calls the *getDecision()* method of the PDP situated in the publish/query interface in order to get a read authorization decision. If it is allowed, the DHT application gets the necessary information to reconstruct the supply network. Otherwise, the DHT application returns an error message. This is possible because we have added a new field to the SUPPLY_NETWORK message: the identity of the query application which calls the *network()* web service method. Our access control mechanism implements one policy for the read operation implemented in the *supply_network()* method of the DHT application. The default behavior is to deny the access to the information except for those clients defined by the companies of the consortium as partners.

5.2 Nested package in supply chains

5.2.1 Introduction

In the EPC technology research, most of the work about track and trace corresponds to item level tracking (Bi & Lin, 2009), (Goebel et al., 2009), (Beier et al., 2006). To face this challenge, all of them assume that items are always visible along the whole supply chain. However, in many industrial fields this supposition does not reflect the reality. For example, clothing industry tags at item level, but products are distributed and move along the supply chain

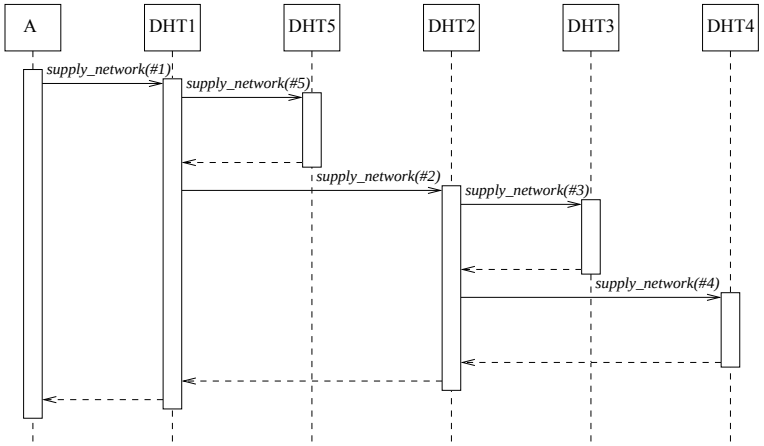


Fig. 8. Interaction among the DHT nodes involved in the reconstruction of this supply network.

within different storage systems (trays, packages, boxes, etc.) (*Charles Voegele Group Finds RFID Helps It Stay Competitive*, 2009). Although each item has its own EPC, items do not go independently towards the destination. In many cases, some items will be packaged together to facilitate transport and distribution.
Item package may produce scenarios with null (or limited) item level visibility. Each item is associated to a RFID tag but, if they are packaged together in a particular storage system, items are not visible to RFID readers. Only the RFID tag associated to the new package is visible. Storage process may be repeated as many times as needed (items into small boxes, small boxes into cases, cases into pallets, etc.). Usually, the EPC associated to the overall wrapper identifies the order.
This section presents an extension of the automatic traceability application to recover, in an efficient way, the complete supply chain of an item in a nested package scenario.

5.2.2 Description of the scenario

An example of nested package identification is shown in figure 9. There, four levels of package are carried out. $\#it_i$ tags identify the items, C_{1x}, C_{2x}, C_{3x} and C_{4x} tags identify the different level packages, being C_{41} the tag associated to the overall wrapper.
Figure 10 shows two supply chains corresponding to the previous nested package scenario: item 1's and item 10's supply chains. Because item tags are not always readable, supply chains will be reconstructed from the tag collections obtained by the RFID readers from the own item or the packages that, at different levels, contain the item. Each supply chain shows three possible actions: package actions, advance actions and unpackage actions. Associated to these actions, three zones can be identified from left to right:

- The first zone represents the package process. $(\#it_i)$ indicates the assignment of an EPC to an item. $(\#C_{jk})$ indicates the assignment of an EPC to a level j package where $j = 1...4$. $\#it_i$ and $\#C_{jk}$ indicate the reading of RFID tags by a RFID reader. For the sake of simplicity and without loss of generality, it is considered that all package actions, from the item creation to the highest level package, are done at the same chain element.

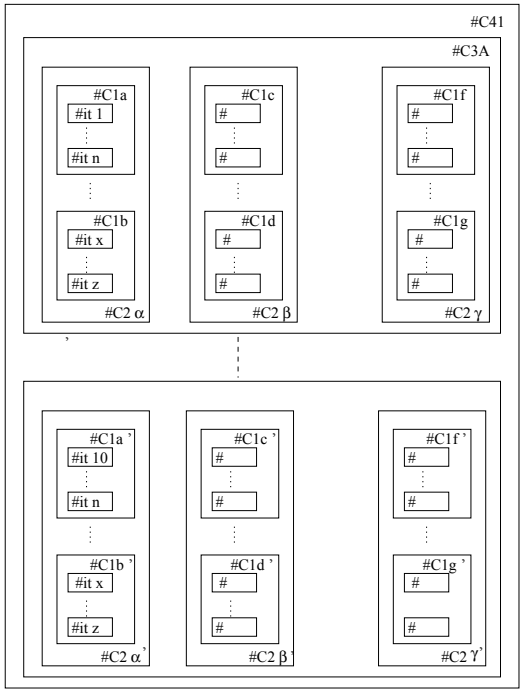


Fig. 9. An example of nested package identification, corresponding to four level package.

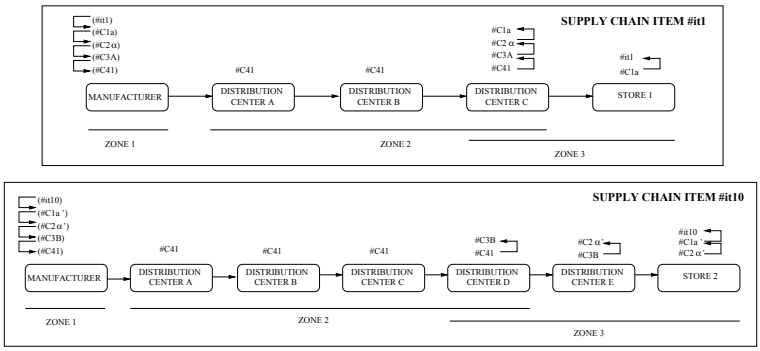


Fig. 10. Supply chains corresponding to items #it1 and #it10. $(\#it_i)$ and $(C\#_{jk})$ indicate the assignment of EPCs to items and packages. $\#it_i$ and $C\#_{jk}$ indicate the reading of tags by RFID readers. Package and unpackage actions are represented by arrows.

- When the highest package level is created (in our example it corresponds with C_{41}), it will pass through several organizations during the advance along the distribution channel.
- The third zone corresponds with the unpackage process. This task could involve several organizations, depending on the unpackage level reached at each organization. In item 1's supply chain, the unpackage process involves Distribution Center C (DC C) and Store 1. DC C unpackages to the lowest package level (C_{1a}), and after receiving the smallest

package unit, Store 1 unpackages the item. On the other hand, the unpackage process in item 10's supply chain involves three organizations: Distribution Center D, Distribution Center E and Store 2.

Although it is not represented in the supply chain example above, in some cases there is tracking information which is obtained during the second zone. In many industrial sectors, random verifications at item level are done at intermediate points of the supply chain. That means, a random lowest level package is chosen and then, all the items of this package (or just some of them) are examined. This action will generate additional information, corresponding to events happened in the central zone of the supply chain.

5.2.3 Supply chain recovery

The solution described in this section will allow the reconstruction of a supply chain at item level, in a totally distributed way. As it was described, the DHT-based DS architecture minimizes the number of messages that a requester organization must send (normally the requester will be an organization with a limited network connection). Most of the queries required to obtain all the information will be performed by DHT (Distributed Hash Table) nodes that, like EPCIS servers, will be components with better features in connectivity, reliability and security. On the other hand, the use of this architecture in nested package scenarios reduces the total number of messages required to obtain multiple supply chains corresponding to different items. Due to the distributed work during supply chain reconstruction, and only maintaining a cache at the DHT nodes, previously requested information can be reused during the reconstruction of new supply chains.

Figure 11 shows, by means of an example, the behavior of the distributed architecture during the storage and recovery of information required to reconstruct the supply chain of a nested package. In this example, the item whose EPC is #1 is created and packaged into three levels (the package systems are identified by $\#C_{11}$, $\#C_{21}$ and $\#C_{31}$ respectively) by company A. The highest level package passes through the distribution channel (companies B and C) to company D, where all the unpackage actions will be done. Interaction among the different elements involved in the reconstruction of the supply chain is represented in the figure. Messages 1 to 20 correspond with the storage in the EPCISs of the read EPC events and the registration of the EPCISs with the DHT network. After that, DHT node P -the responsible of key hash(#1)- stores the IP addresses of $EPCIS_A$ and $EPCIS_D$ (that is, the EPCISs storing events about EPC #1). DHT node Y -the responsible of key hash($\#C_{11}$)- stores the IP addresses of $EPCIS_A$ and $EPCIS_D$ (that is, the EPCISs storing events about EPC $\#C_{11}$). DHT node M -the responsible of key hash($\#C_{21}$)- stores the IP addresses of $EPCIS_A$ and $EPCIS_D$ (that is, the EPCISs storing events about EPC $\#C_{21}$). Finally, DHT node O -the responsible of key hash($\#C_{31}$)- stores the IP addresses of $EPCIS_A$, $EPCIS_B$, $EPCIS_C$ and $EPCIS_D$ (that is, the EPCISs storing events about EPC $\#C_{31}$). The rest of the messages are required to reconstruct the supply chain of item #1, process that is initiated by company D. Organization D will query its associated DHT node (node Z) to initiate the recovering of the supply chain (message 21). That DHT node will locate the DHT node responsible for the item EPC #1 (message 22). The node responsible for #1 (in the example node P) stores the addresses of the EPCIS that have data about the EPC ($EPCIS_A$ y $EPCIS_D$). Consequently, node P will be able to query them all the information about #1 (messages labeled jointly in the figure as 23). Information associated to EPC #1 indicates that the item was packaged into $\#C_{11}$. Therefore, instead of replying directly to node Z (the node that initiates the lookup), node P will locate the DHT node responsible for $\#C_{11}$ (message 24). Only when node P receives the data about $\#C_{11}$, it will

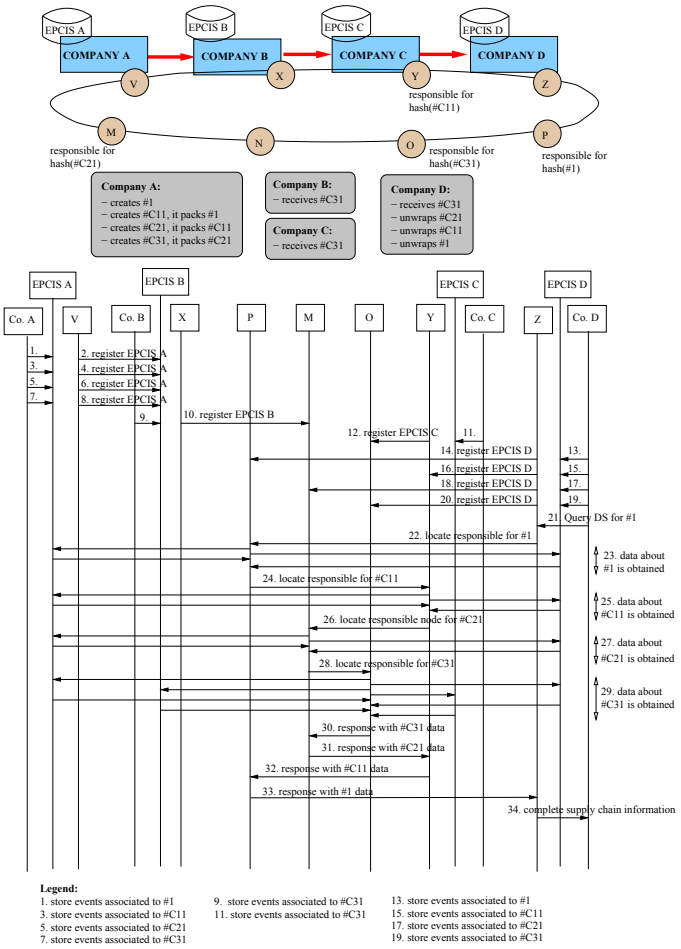


Fig. 11. Reconstruction of a supply chain in a nested package scenario.

reply to node Z with all the information. Due to $\#C_{11}$ is associated to $\#C_{21}$, node Y also locates the DHT node responsible for $\#C_{21}$ before answering to node P (message 26). Finally, node M, which is responsible for $\#C_{21}$, will locate the DHT node responsible for $\#C_{31}$ (message 28). Starting at node O, all the involved DHT nodes will respond successively and in the opposite direction to the lookups, with the requested information (messages 30 to 33). Finally, node Z will be able to answer the initial query performed by company D. This response will contain all the information required to reconstruct the supply chain of item #1.

This solution allows to recover, in a distributed way, supply chains in a nested package scenario. As it can be seen, each DHT node will find out a portion of the supply chain, and all these portions will form the complete supply chain. In fact, thanks to this feature the proposed solution offers a significant gain against the EPCglobal solution in terms of network usage. To obtain this gain, it would be sufficient if DHT nodes acted as proxies, that is, if they stored in a cache the information that was recovered during the reconstruction of a supply chain after

answering the requesting node. In this way, when the supply chain of a new item is required, and this item coincides at any package level with another item whose supply chain has been obtained before, some portions of the supply chain have already been reconstructed and it is not necessary to query again. Depending on the level of coincidence between the items, the gain will be greater or smaller. On the other hand, depending on the cache maintenance (in terms of time-to-live of entries), the system gain will also vary.

5.2.4 Evaluation

5.2.4.1 Use of DHT nodes

The main contribution of this solution is the use of a DHT network to implement, in an efficient and distributed way, the item level track&trace service. Therefore, it is interesting to evaluate the consequences in the DHT nodes of storing the information associated to the supply chains. Specifically, the amount of involved nodes and information that each node must maintain is going to be evaluated. Remember that each node will store, for all the EPC under its responsibility, the URL of the EPCIS that have information about the EPC.
If an item package process reaches i levels, the number of DHT nodes which will be involved is $(i+1)$ (the nodes responsible for keys $\#it$, $\#C_{1k}$,..., $\#C_{im}$). Each node should store an entry, that corresponds to the Internet address of the manufacturer. On the other hand, the movement of the highest level package along the distribution chain does not involve other nodes, though it will add new entries on the node responsible for that package. That is, if j elements are gone through, the node responsible for $\#C_{im}$ should store j new entries. In the same way, the unpackage tasks will only affect to the number of entries stored on the nodes who are responsible for the assigned EPCs, but it does not modify the number of involved DHT nodes. Due to this last phase, a new entry for each unpackage level will be generated.
Usually, any industrial activity does not create a single item but a high number of them. If P items are produced, and they are grouped into i levels, the number of EPCs that are used by the manufacturer is:

$$P + \lceil P/n_1 \rceil + \lceil \lceil P/n_1 \rceil / n_2 \rceil + \lceil \lceil \lceil P/n_1 \rceil / n_2 \rceil / n_3 \rceil + ... \leq P + i + \sum_{j=1}^{i} \frac{P}{\prod_{m=1}^{j} n_m} \quad (1)$$

where n_i is the number of packages of $i-1$ level that are packaged into a i level package.
Because the number of involved nodes depends only on the number of EPCs, if N is the number of nodes that belong to the DHT network and considering that the amount of items is substantially higher than the number of nodes ($N << P$), due to the DHT network features, each node will be responsible for the following number of EPCs:

$$\frac{P + i + \sum_{j=1}^{i} \frac{P}{\prod_{m=1}^{j} n_m}}{N} < \frac{2P + i}{N} \sim \frac{2P}{N} \quad (2)$$

that is, a limited number of entries that can easily be managed by any current database system.

5.2.4.2 Network capacity gain

One of the most frequently used performance features of information system is the efficient use of network capacity. Here, this feature is measured in terms of absolute number of application messages exchanged to store and obtain the data related to an item supply chain. Application messages are the queries, responses, inserts, lookups, etc. generated in the evaluated systems. In fact, each application message could involve more than one network

message. For example, in the centralized EPCglobal proposal, a query to the Discovery Service will involve a certain number of messages partly due to the use of a DNS-based ONS. In the proposed distributed solution, an insert or a lookup message will involve a certain number of messages due to the structured overlay network mechanisms (Chord, Pastry, Tapestry,...), which is due to the use of a DHT-based ONS.
Both ONS architectures are compared by simulation in section 5.2.4.4. As it will be concluded there, the number of nodes that must be contacted to resolve a ONS query is always lower in the DHT-based solution than in the DNS-based solution. Therefore, an DHT-based application message implies a lower number of network messages.
In this section, it is considered that both solutions implement the same functionality. That is, DHT nodes in the distributed solution do not use cache tables.
To carry out the evaluation, the following parameters are considered: l is the number of elements of a supply chain ($l \in N^+ \backslash \{1\}$), i is the number of package levels and p is the coincidence level between two items. If both items are packaged in the same level 1 package, p value is 1. However, if both items are packaged in different level 1 units, but in the same level 2 package, p value is 2, and so on.
For the sake of simplicity (like in the example in figure 11), it is considered that the package and unpackage actions take place in the beginning and at the end of the supply chain respectively, and therefore, the intermediate elements along the supply chain just read the highest level package. It is also considered that the item supply chain reconstruction is initiated by the last element.
According to a centralized solution, the number of messages required to store and later reconstruct the supply chain of an item is:

$$M_{centralized} = 2 \cdot (i+1) + (l-2) + 2 \cdot (i+1) + (l-2) + 2 \cdot (i+1) + 1 \cdot 2 \cdot l + i \cdot 2 \cdot 2 \quad (3)$$

The first four terms correspond with messages which are generated during the storage of data, that is, messages which are generated when the item is created and moves along the distribution channel until the destination. $2 \cdot (i+1) + (l-2)$ messages are sent to the EPCISs and $2 \cdot (i+1) + (l-2)$ are sent to DS. The rest of terms are created due to the recovering of the item supply chain. Let's identify each one: $2 \cdot (i+1)$ messages are exchanged between the element requiring the supply chain and the DS, and $1 \cdot 2 \cdot l + 2 \cdot 2 \cdot i$ messages are exchanged between the last element and each involved EPCIS.
According to a distributed solution, the total number of messages is:

$$M_{distributed} = 2 \cdot (i+1) + (l-2) + 2 \cdot (i+1) + (l-2) + 2 + 2 \cdot (i+1) + 1 \cdot 2 \cdot l + i \cdot 2 \cdot 2 \quad (4)$$

Like equation 3, the four first terms correspond to messages which are generated during the storage of data. If $2 \cdot (i+1) + (l-2)$ messages are sent to the DS in the centralized solution, in this solution all these messages are distributed amount $(i+1)$ nodes: The nodes that are responsible for the $i+1$ EPCs involved in the item supply chain. The rest of messages are generated during the supply chain reconstruction process. Although the amount of messages is almost the same, the sending and the reception of the messages is completely decentralized. Whereas in a centralized solution the last element is responsible for sending all the messages, in a distributed solution, all the DHT nodes that store data about the required supply chain will participate in the supply chain data recovery. In the distributed solution, the requester element initiates the supply chain reconstruction process by querying the DHT

node associated to its EPCIS. Finally, that element will receive from that DHT node all the data required to reconstruct the supply chain. That is the reason for the $+2$ term in equation 4.
It can be concluded that the gain between the centralized solution and the distributed proposal is not related to message consumption. In fact, the main improvement of the last solution is the distribution and decentralization of message sending and reception. In addition, as it will be described in the next section, the distributed behavior of the last solution will make possible to reduce the number of required messages when reconstructing more than one supply chain if they work as proxies.

5.2.4.3 Network capacity gain using cache

The DHT-based DS architecture offers an efficient and distributed solution to reconstruct supply chains. In fact, the proposed distributed solution also offers an additional advantage in nested package scenarios. To obtain this improvement, it is necessary that the DHT nodes maintain a cache table, where the information that they have obtained to respond to another DHT node's query during a supply chain recovery process will be stored. Thus, the total number of messages needed to reconstruct the supply chain of different items is reduced. For example, using the example shown in figure 11, if item #2 is packaged together with item #1 (and item #1 supply chain has been obtained), data about $\#C_{11}$, $\#C_{21}$ and $\#C_{31}$ is already available to DHT node Y, which will respond to the lookup sent by the DHT node responsible for #2 during the supply chain reconstruction process.
In the previous section, equation 4 corresponds to the number of messages that are exchanged during the storage of supply chain information and the recovery of that information without using cache. If DHT nodes maintain a cache table, the number of messages required to store and retrieve the supply chain information of an item which does not coincide with other item at any level (level of coincidence p=0) is logically the same in that equation. However, if the required item coincides at any level (p) with a previous item whose supply chain was already reconstructed, the number of messages is reduced according to the next expression:

$$M_{distrcache} = 2 \cdot [2 \cdot (i+1) + (l-2)] + 2 + 2 \cdot (p+1) + 2 \cdot 2 \cdot p \tag{5}$$

As in equation 4, the first term corresponds with messages which are generated during the storage of data. Actually, both equations only differ on the number of message during the data recovering process. Therefore, to obtain the gain of the distributed solution using DHT nodes as proxies, only these terms are considered, as described in equation 6.

$$gain = 1 - \frac{2 + 2 \cdot (p+1) + 2 \cdot 2 \cdot p}{2 + 2 \cdot (i+1) + 2 \cdot 2 \cdot i + 1 \cdot 2 \cdot l} \tag{6}$$

Table 4 shows the gain values according to equation 6 corresponding to different package levels (i) and different levels of coincidence (p). l indicates the length of the supply chains. Next, the main conclusions of these results are exposed.
First, it can be noticed that for the same i and p values (that is, for the same number of package levels and the same level of coincidence between items), the gain increases as supply chain length is longer. This is because the information about the highest level package, which contains both coincidence items within any of its internal packages, has already been obtained during the first supply chain reconstruction. It will not be necessary to find out again what happened with the highest level package along the supply chain. Therefore, the greater the number of elements in the supply chain is, the greater the gain is.

		i=1	i=2	i=3	i=4	i=5
l=2	p=1	0.2857	0.5000	0.6154	0.6875	0.7368
	p=2	-	0.200	0.3846	0.5000	0.5768
	p=3	-	-	0.1538	0.3125	0.4211
	p=4	-	-	-	0.1250	0.2632
	p=5	-	-	-	-	0.1053
l=3	p=1	0.3750	0.5455	0.6429	0.7059	0.7500
	p=2	-	0.2727	0.4286	0.5294	0.6000
	p=3	-	-	0.2143	0.3529	0.4500
	p=4	-	-	-	0.1765	0.300
	p=5	-	-	-	-	0.1500
l=4	p=1	0.4444	0.5833	0.6667	0.722	0.7619
	p=2	-	0.3333	0.4667	0.5556	0.6190
	p=3	-	-	0.2667	0.3889	0.4762
	p=4	-	-	-	0.2222	0.3333
	p=5	-	-	-	-	0.1905
l=5	p=1	0.5000	0.6154	0.6875	0.7368	0.7727
	p=2	-	0.3846	0.5000	0.5789	0.6364
	p=3	-	-	0.3125	0.4211	0.5000
	p=4	-	-	-	0.2632	0.3636
	p=5	-	-	-	-	0.2773

Table 4. Network capacity gain when using caches. i is the number of package levels, p is the level of coincidence between the requested item and a previous item. l is the length of the supply chain.

Secondly, for items that move along the supply chain within the same highest level package, the lower p value is, the greater the gain is. That is, the best gain values are obtained when the coincidence between items happens in a lower level. Logically, if the number of common levels is higher, the number of queries is smaller because this information is already obtained. Finally, for supply chains of the same length and the same level of coincidence between items, the higher the total number of levels is, the greater the gain is. A bigger difference between p and i indicates that the number of common levels is higher. Thus, the amount of information to find out is reduced.

The above values represent the gain when recovering the information about an item using cache at DHT nodes in comparison to the recovering of a previous item if any cache is used. However, these values also represent the relation between the amount of messages generated during the first (all caches will be empty) and the second recovery in a distributed system with caches. Here, if a third item is considered, the gain is determined by the best level of coincidence with both previously requested items. Therefore, to obtain the network capacity gain of DHT-based DS architecture due to the use of cache tables, it is necessary to consider all the previously requested items.

Figure 12 shows the network capacity gain results in a nested package scenario. In this scenario, 100 items are packaged at level 1, 20 level 1 units are packaged at level 2, 10 level 2 units are packaged at level 3. Three level 3 packages have been created, which correspond to 100000 items. The length of the supply chain (l) is 5. The X axis corresponds to the amount of supply chains that have been resolved as time goes by. Items are randomly chosen. This figure represents the gain obtained for each requested supply chain and also the average network capacity gain. Since only three levels of package are used, the possible gain values are 0.6875, 0.5 and 0.3125, depending on the level of coincidence. Thus, three zones can be distinguished in the figure. In the first zone (until the value 100, approximately) the most common gain

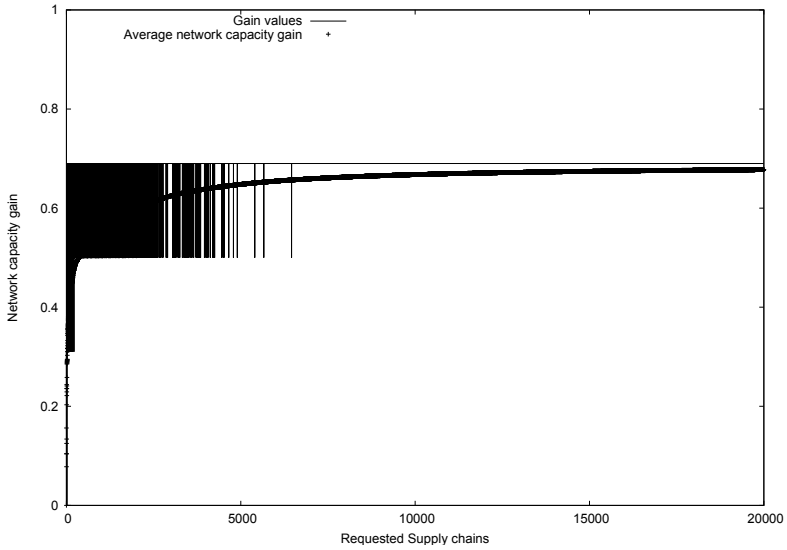


Fig. 12. Gain value of each requested supply chain and the average network capacity gain.

value begins 0.375, then changes up to 0.5 and even some 0.6875. In the second zone (between 100 and 2500), gain values move between 0.5 and 0.6875. Finally, in the last zone the most repeated gain value is 0.6785. Figure 12 shows how the average network capacity gain quickly increases until the constant value of 0.6743 is reached.

5.2.4.4 Application level messages

The parameter used in the previous sections to measure the network capacity is the number of application messages exchanged to store and obtain the data related to an item supply chain. In fact, these messages (queries, responses, inserts, lookups,...) involve more than one network message, depending on the number of nodes that it is necessary to contact in order to perform the required operation.

Here, this last parameter is going to be evaluated in two architectures: the first one implements a ONS based on the traditional DNS, and the second one implements a ONS based on a DHT network. The evaluation has been performed by simulation. The supply chains related to each EPC have been constructed using a C++ STL-like container class for trees, which depends of the probabilities of increasing the length and the width of the tree. The DNS-based approach has been evaluated using the OMNeT++ discrete event simulator, and the DHT-based approach has been evaluated using the OverSim P2P simulation framework for OMNeT++. All the simulation parameters are equally configured in both cases. For example, the number of nodes immersed in the ONS service is 65 (in the DNS-based implementation these nodes are organized in a tree hierarchy and in the DHT-based implementation these nodes compose a Chord overlay network (Stoica et al., 2003)).

Table 5 presents the simulation results. The first column represents the probability of increasing the length of the supply chain and the second one the probability of increasing the width. The third and fourth columns represent the average number of nodes that it is necessary to contact to reconstruct a full supply chain that has been built taking into account the previous probabilities. The average values are obtained from the reconstruction of 100000

p(length)	p(width)	Number of nodes	
		DHT	DNS
0.30	0.10	3.24	7.56
0.30	0.30	3.80	8.89
0.30	0.50	4.51	10.54
0.50	0.10	3.61	8.44
0.50	0.30	5.38	12.54
0.50	0.50	8.51	19.85
0.70	0.10	4.72	11.03
0.70	0.30	15.84	36.89
0.70	0.50	288.46	673.05

Table 5. Number of contacted nodes using DHT-based solution and DNS-based solution

supply chains, which is greater enough to guarantee stationary values. From the results it can be concluded that the number of nodes contacted is always lower in the DHT-based solution than in a DNS tree. On the other hand, it has also been noticed that the necessary number of contacted nodes in the DNS tree is roughly a 133% greater than the required number in the DHT-based solution. That is, the increment is approximately a constant value.

6. Conclusion

In this chapter we have analyzed the advantages of implementing the Discovery Services (DS) component of the EPCglobal Network architecture using a Distributed Hash Table (DHT) application. In addition, we have also showed that it is possible to develop new applications over the DHT-based discovery services.

Recently, there have been several proposals for the implementation of the DS, like the DS prototype of the Bridge project, or the Extensible Supply-chain Discovery Service (ESDS) developed by Afilias. In both the DS has been implemented as a centralized database, to be more specific in the Bridge Project it has been developed as a centralized database based on LDAP (Lightweight Directory Access Protocol).

DHTs might be said to have several advantages over LDAP: in LDAP there is a bottleneck in the root of the architecture, whereas in the DHT the failure of one node does not affect the whole system; LDAP is not optimized for massive update operations, while DHTs are optimized for massive search and update operations. On the other hand, DHTs have an important drawback with respect to LDAP: while in LDAP the access control is already implemented by Access Control Lists (ACLs) in DHT fine grained controls have to be implemented.

We have proposed a mechanism for automatically obtaining the supply network associated to a specific product. There are other systems with the same objective but the main difference with the proposal set out in this chapter is that in other systems the client has to do all the operations to reconstruct the supply network. That is, initially, it has to obtain the URLs (Uniform Resource Locator) of the information services with information about a specific product, then it has to access the corresponding information services to get the necessary information, and after that it can reconstruct the full supply network. However, in our proposal the client does not have to perform any operation, that is, it queries the supply network associated to a specific product and the results are obtained directly.

On the other hand, in the EPC technology research, most of the work about track and trace corresponds to item level tracking. To face this challenge, all of them assume that items are always visible along the whole supply chain. However, in many industrial fields this

supposition does not reflect the reality. For example, clothing industry tags at item level, but products are distributed and move along the supply chain within different storage systems (trays, packages, boxes, etc.). This chapter has proposed a distributed architecture to recover, in an efficient way, the complete supply chain of an item in a nested package scenario. In addition, the proposed solution improves the EPCglobal Network features in terms of network usage and also in terms of distribution and decentralization of tasks associated to capturing and querying event information.

7. Acknowledgments

This research has been supported by the MICINN/FEDER project grant TEC2010-21405-C02-02/TCM (CALM) and it is also developed in the framework of "Programa de Ayudas a Grupos de Excelencia de la Región de Murcia, de la Fundación Séneca, Agencia de Ciencia y Tecnología de la RM (Plan Regional de Ciencia y Tecnología 2007/2010)".

8. References

Armenio, F., Barthel, H., Dietrich, P., Duker, J., Floerkemeier, C., Garrett, J., Harrison, M., Hogan, B., Mitsugi, J., Preishuber-Pfluegl, J., Ryaboy, O., Sarma, S., Suen, K., Traub, K. & Williams, J. (2009). Epcglobal architecture framework, Available online at: http://www.epcglobalinc.org/standards/architecture/architecture_1_3-framework-20090319.pdf.

Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R. & Stoica, I. (2003). Looking up data in p2p systems, *Communications of the ACM* 46(2): 43–48.

Beier, S., Grandison, T., Kailing, K. & Rantzau, R. (2006). Discovery services - enabling rfid traceability in epcglobal networks, *Proceedings of International Conference on Management of Data (COMAD)*.

Bi, H. H. & Lin, D. K. J. (2009). Rfid-enabled discovery of supply networks, *IEEE Transactions on Engineering Management* 56(1): 129–141.

Charles Voegele Group Finds RFID Helps It Stay Competitive (2009). Available at: http://fridjournal.com/article/view/4836.

Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J. & Stoica, I. (2003). Towards a common api for structured peer-to-peer overlays, *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, pp. 33–44.

Eastlake, D. & Jones, P. (2001). RFC 3174: Secure hash algorithm 1 (sha1), Available online at: http://tools.ietf.org/rfc/rfc3174.txt.

Främling, K., Ala-Risku, T., Kärkkäinen, M. & Holmström, J. (2006). Agent-based model for managing composite product information, *Computers in Industry* 57(1): 72–81.

Främling, K., Harrison, M., Brusey, J. & Petrow, J. (2007). Requirements on unique identifiers for managing product lifecycle information: comparison of alternative approaches, *International Journal of Computer Integrated Manufacturing* 20(7): 715–726.

Goebel, C., Tribowski, C. & Günter, O. (2009). Epcis-based supply chain event management - a quantitive comparison of candidate system architectures, *Proceedings of the International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS'2009)*, pp. 494–499.

Li, X. & Chandra, C. (2007). Efficient knowledge integration to support a complex supply network management, *International Journal of Manufacturing Technology and Management* 10(1): 1–18.

Maymounkov, P. & Mazières, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric, *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, London, UK, pp. 53–65.

Mealling, M. & Daniel, R. (2000). RFC 2915: The naming authority pointer (naptr) dns resource record, Available online at: http://tools.ietf.org/rfc/rfc2915.txt.

Mockapetris, P. (1987). RFC 1035: Domain name system, Available online at: http://tools.ietf.org/rfc/rfc1035.txt.

Nwana, H. S. (1996). Software agents: An overview, *Knowledge Engineering Review* 11(3): 1–40.

Phillips, W., Johnsen, T., Caldwell, N. & Lewis, M. A. (2006). Investigating innovation in complex health care supply networks, *Health Services Management Research* 19(3): 197–206.

Poulin, M., Montreuil, B. & Martel, A. (2006). Implications of personalization offers on demand and supply networks design: A case from the golf club industry, *European Journal of Operational Research* 169(3): 996–1009.

Rowstron, A. & Druschel, P. (2001). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pp. 329–350.

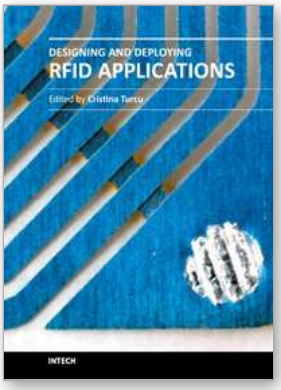
Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F. & Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications, *IEEE/ACM Transactions on Networking* 11(1): 17–32.

Wareham, J., Mathiassen, L., Rai, A., Straub, D. & Klein, R. (2005). The business value of digital supply networks: A program of research on the impacts of globalization, *Journal of International Management* 11(2): 201–227.

Young, M. (2008). Extensible supply-chain discovery service (esds) concepts, Available online at: http://tools.ietf.org/id/draft-young-esds-concepts-04.txt.

Zeilenga, K. (2006). RFC 4510: Lightweight directory access protocol (ldap), Available online at: http://tools.ietf.org/rfc/rfc4510.txt.

Zhao, B. Y., Huang, L., Stribling, J., rhea, S. C., Joseph, A. D. & Kubiatowicz, J. (2004). Tapestry: A resilient global-scale overlay for service deployment, *IEEE Journal on Selected Areas in Communications* 22(1): 41–53.



Designing and Deploying RFID Applications

Edited by Dr. Cristina Turcu

ISBN 978-953-307-265-4
Hard cover, 384 pages
Publisher InTech
Published online 15, June, 2011
Published in print edition June, 2011

Radio Frequency Identification (RFID), a method of remotely storing and receiving data using devices called RFID tags, brings many real business benefits to today world's organizations. Over the years, RFID research has resulted in many concrete achievements and also contributed to the creation of communities that bring scientists and engineers together with users. This book includes valuable research studies of the experienced scientists in the field of RFID, including most recent developments. The book offers new insights, solutions and ideas for the design of efficient RFID architectures and applications. While not pretending to be comprehensive, its wide coverage may be appropriate not only for RFID novices, but also for engineers, researchers, industry personnel, and all possible candidates to produce new and valuable results in RFID domain.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821