Building **R**adio frequency **ID**entification for the **G**lobal **E**nvironment

High level design for Discovery Services

Authors: University of Cambridge, AT4 wireless, BT Research, SAP Research

15 August 2007

This work has been partly funded by the European Commission contract No: IST-2005-033546

About the BRIDGE Project:

BRIDGE (**B**uilding **R**adio frequency **ID**entification for the **G**lobal **E**nvironment) is a 13 million Euro RFID project running over 3 years and partly funded (€7,5 million) by the European Union. The objective of the BRIDGE project is to research, develop and implement tools to enable the deployment of EPCglobal applications in Europe. Thirty interdisciplinary partners from 12 countries (Europe and Asia) are working together on : Hardware development, Serial Look-up Service, Serial-Level Supply Chain Control, Security; Anti-counterfeiting, Drug Pedigree, Supply Chain Management, Manufacturing Process, Reusable Asset Management, Products in Service, Item Level Tagging for non-food items as well as Dissemination tools, Education material and Policy recommendations.

For more information on the BRIDGE project: www.bridge-project.eu

This document:

This document describes a high-level design for EPC Discovery Services. It starts in section A with the description of the high level design we have decided. It is important to point out that despite being a Bridge design, it includes ideas form external sources. Furthermore, during the discussions within WP2, contact with external companies working on lookup services has been established and through those a wider understanding of the possibilities has been obtained. The second section on this deliverable is the result of many discussions held during the months while the task was active. The document begins by presenting all the possibilities we have considered and makes an evaluation of them, in order to justify the choice taken. Different models are presented with different operating mechanisms and message flows, although they aim for the same basic performance objectives. Finally, the document is completed with section C, which deals with the selected technology to implement the storage component in the serial level lookup service, which is a critical component, whose robustness and reliability of service will impact upon the response times, management of records, interface to low level components of the network, interface to clients making queries, and so on.

Disclaimer:

This document results from work being done in the framework of the BRIDGE project. It does not represent an official deliverable formally approved by the European Commission.

Introduction

The objectives of the BRIDGE project are twofold. Firstly, it is developing RFID and EPC Network technology covering different aspects tags, readers, serial level lookup services and user applications for track and trace. Secondly, Bridge aims to demonstrate and disseminate through Europe the value of RFID and EPC Network technology and its potential benefit for various business sectors.

Therefore, there is a clear division between those work packages working on the different aspects of RFID/EPC technology which may be considered as horizontal activities that provide the foundations for subsequent development of business oriented work packages activities, which implement RFID/EPC based solutions in the field and evaluate the benefits of the technology to improve business processes.

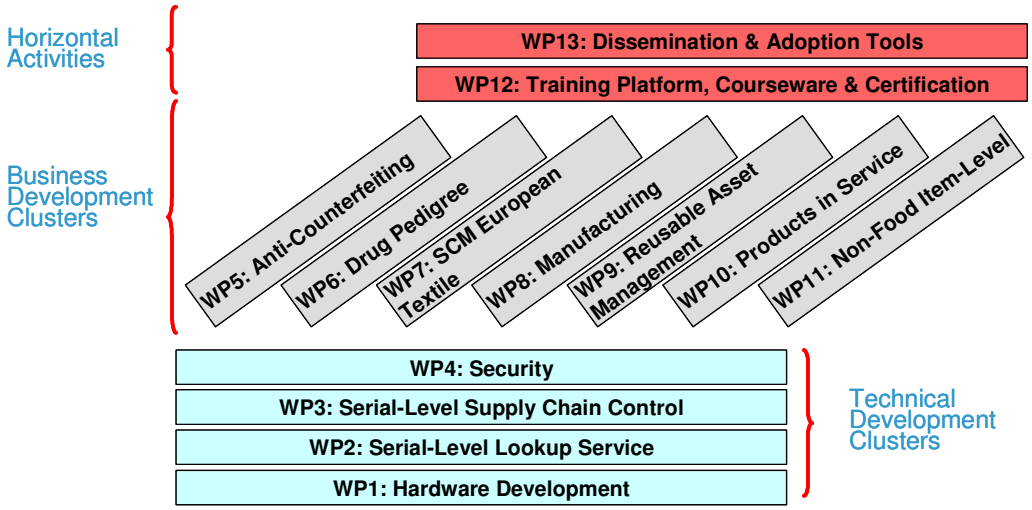


Figure 1: Work Package structure in the BRIDGE project

Among the technical WPs, WP2 focuses on serial level lookup services, which provide track and trace information of a given tagged item as it moves along the supply chain.

This deliverable, the second issued by WP2, is the result of the work developed on task 2.5 “Discovery Service – High Level Design”. It is important to notice that the development of this document was not initially contemplated, but through the system requirements elicitation process the group realized that the task and the document was necessary in order to provide the prototyping task (T2.3) with suitably detailed documentation to achieve its objectives.

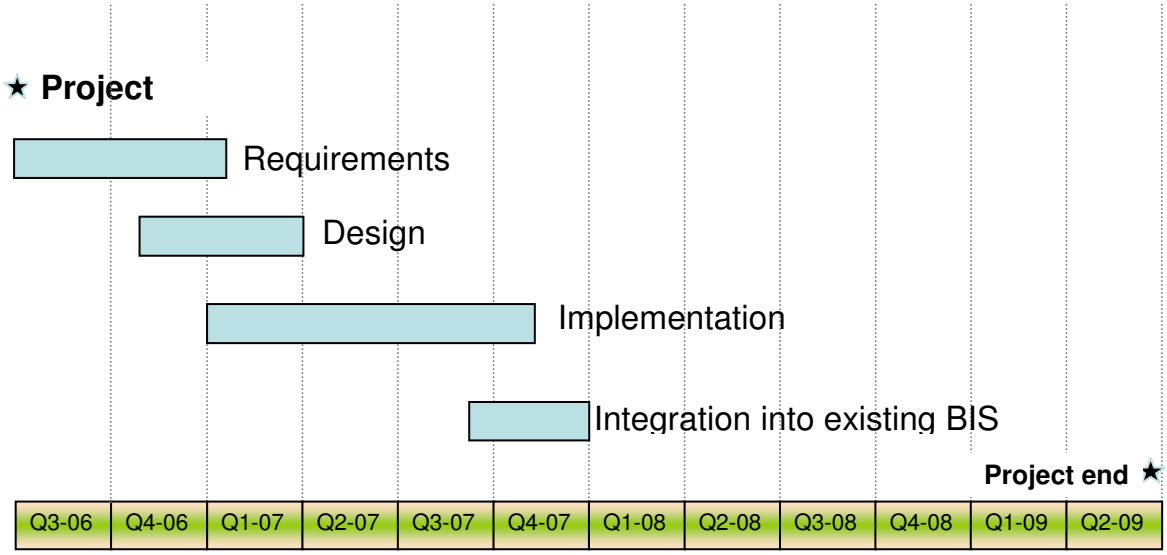


Figure 2: tasks and timeline in Bridge WP2

Background

EPCglobal was created with a concrete target: to develop a universal identification system and an open architecture to provide interoperability in a complex multi-vendor scenario. This universal identification system is based on the allocation of a unique EPC (Electronic Product Code) to every item. As a result, the EPC Network[1] is an architecture proposed for enabling sharing of information about individually identifiable objects among organizations (See Figure 3). Each individual instance of an object can be given a globally unique identifier (unique ID), enabling each object to be tracked worldwide, by means of automatic identification technologies such as Radio-Frequency Identification (RFID), as well as linear barcodes or two-dimensional barcodes. Furthermore, such Auto-ID technologies enable an individual 'life history' of each individual object to be collected efficiently – and this additional data can be linked to the object via the globally unique ID of each object. With a suitable service-oriented architecture, the unique ID can be used both to locate source of information, via lookup services, as well as for extracting relevant information about a particular object from each source, by using the unique ID as a lookup key within a database.

In the EPC Network, the Electronic Product Code (EPC)[2] serves the role of a globally unique ID for objects. In fact, as defined in EPC Tag Data Standards, EPC is not a single identifier scheme but rather a framework for an extensible family of unique identifiers, many of which are aligned with legacy identifiers, extended where necessary with the addition of a serial number, to achieve uniqueness. Each member of the family of unique identifiers is given a unique Uniform Resource Name (URN) prefix. For example, a serialized Global Trade Item Number (GTIN)[3] begins with the prefix 'urn:epc:id:sgtin:' whereas a Serialized Shipping Container Code (SSCC)[4] begins with the prefix 'urn:epc:id:sscc:'. In this way, all EPC identifiers are guaranteed unique, since the URN prefix is unique for each namespace or identifier scheme, while the remainder of the EPC is unique within that namespace or identifier scheme. It should be noted that 'Electronic Product Code' is something of a misnomer, since not all EPC identifiers necessarily indicate the product type.

There are considerable efficiencies to be gained within a supply chain resulting from exchange of more accurate and timely information about flows of goods between trusted trading partners. For example, many retailers are encouraging the adoption of Auto-ID technologies in order to reduce out-of-stocks and to improve replenishment processes. The pharmaceutical industry is considering item-level tagging of pharmaceuticals, together with electronic pedigree mechanisms in efforts to prevent counterfeit products from entering the supply chain. The aerospace sector is considering tagging aircraft parts, in order to automate the gathering of information about faults and maintenance operations, in order to improve maintenance processes, as well as being able to mine the data to identify systematic failure patterns across parts of a similar type or exposed to similar conditions, in order to improve safety and reliability of parts, by making necessary improvements to design and manufacturing processes.

Sharing of data is of course commercially sensitive, especially information about volumes and flows of good and relationships between trading partners, which could be used advantageously by competitor organizations if the necessary security mechanisms and access controls were absent or compromised.

As a result of such concerns, one of the fundamental design principles for the EPC Network is that each company should be able to retain control over the data that they collect or generate within their own organization, i.e. information is decentralized across multiple organizations[5].

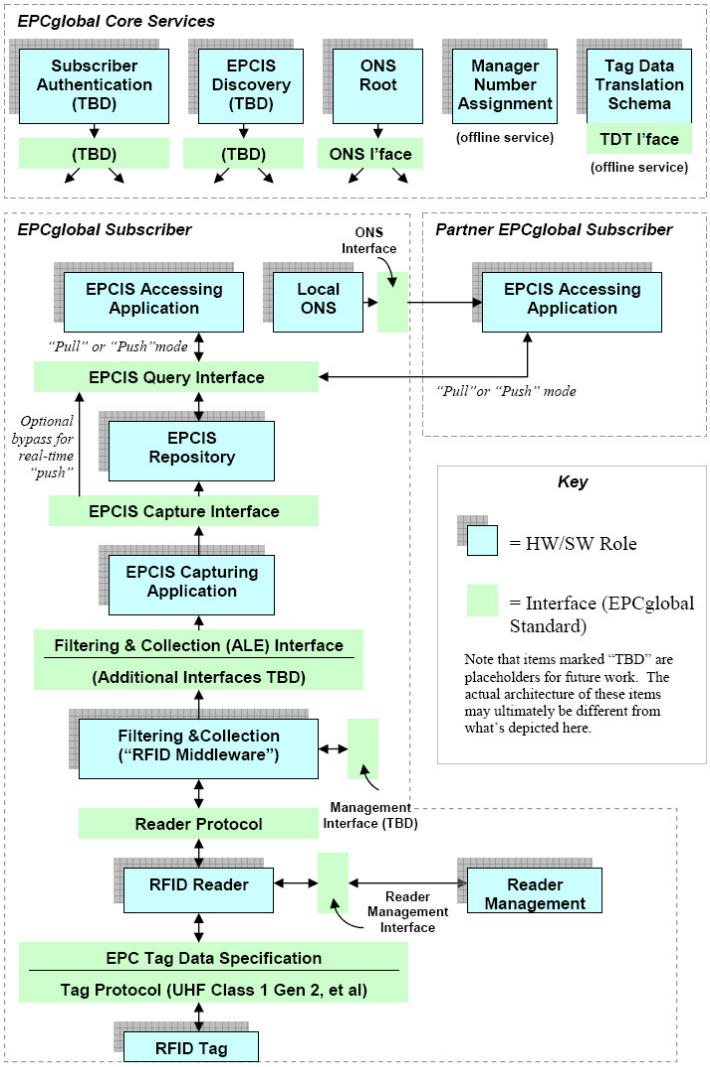


Figure 3: EPCglobal Network Architecture

EPCIS

The EPC Information Services are a *role* defined in EPCglobal Network Architecture Framework [1], which provide for storage and retrieval of filtered and processed information about different events within the supply-chain. The EPCIS offers two interfaces: one for query request and the other one for capture operations. The query interface allows trading partners to query information about any event data stored in the EPCIS-repository together with business context.

However for such a decentralized architecture, since the complete information about an individual object may be fragmented across multiple organizations, there is a need for lookup services for locating all the providers of the fragments of information that constitute the complete supply-chain or lifecycle history for an object.

The EPCglobal Network Architecture Framework document [1] envisages two complementary lookup services: the Object Name Services and the Discovery Services.

Object Name Services

Object Name Services (ONS)[7] provide pointers to authoritative information about an object; this usually means that they provide a pointer to the information services provided by the manufacturer of the object. Multiple types of services can be included in ONS records, including not only EPC Information Services (EPCIS) but also product-specific web pages, web services and other data, such as XML data about products.

The ONS v1.0 standard[7] explains how to query the object name service, given a unique EPC identifier. It should be noted that the ONS lookup mechanism is currently only defined for serialized GTIN EPCs. Furthermore, the granularity of ONS resolution is currently limited to product type, rather than serial-level lookup. i.e. an ONS is not expected to retain distinct records for two objects of the same product type that only differ in their serial numbers – in this situation, ONS would only hold records for the product type. Another point to note is that ONS is currently implemented using the Domain Name System (DNS)[8], using Type 35 Naming Authority Pointer (NAPTR)[9] records to return the information. Queries to ONS are therefore performed by means of a DNS query for a hostname derived from an EPC – and no authentication or authorization is required to perform an ONS query. This is clearly not appropriate for serial-level lookup services for tracking and tracing of objects across the supply chain

Discovery Services

Discovery Services (DS) are envisaged to provide pointers to multiple providers of information across a supply chain, to indicate the addresses of information services of all organizations that hold information about a given EPC – not only the manufacturer. Unlike Object Name Services (ONS), it is expected that most clients querying a Discovery Service will be required to provide authentication credentials – and the amount of information returned in response to their query will be subject to filtering by access control policies based upon the authentication credentials they supply and the business relationship they have with each provider of information that registers records (and associated access control policies) with a Discovery Service.

Discovery Services will need to be designed to accept updates in close to real time from multiple providers of information across the supply chain or lifecycle of an object (including organizations that handle the object beyond the point of sale or delivery, e.g. for repair purposes, maintenance, returns and reverse logistics, as well as recycling, remanufacturing and other end-of-life processes). Because they store serial-level records, they will need to be sufficiently scalable to store large volumes of data, possibly up to trillions of records per year. They will also need to provide for authentication of both information providers (publishers) and those making queries (clients) and accept and enforce access control policies that are defined in a manageable way.

The complementary role of ONS and Discovery Services in relation to multiple EPC Information Services is shown in Figure 4 below:

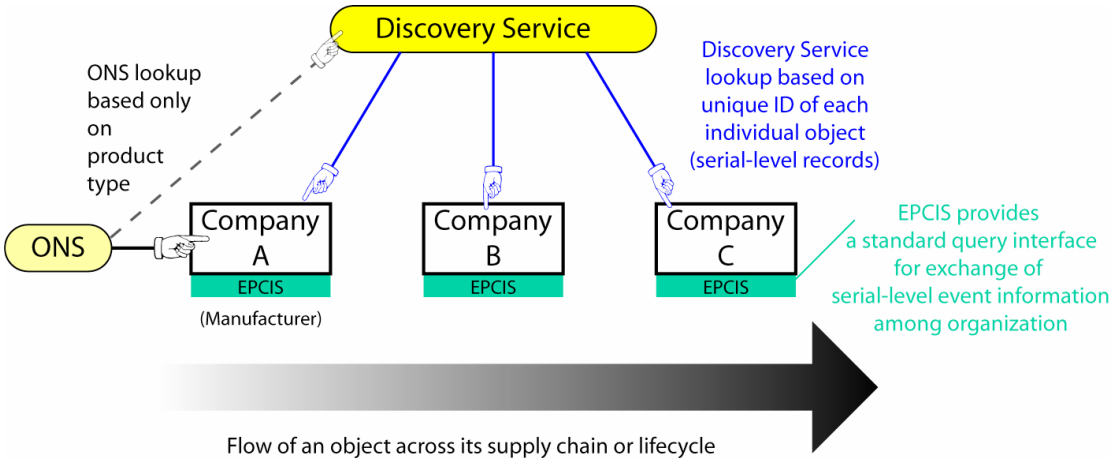


Figure 4 – Complementary roles of ONS and Discovery Services

References

1. *EPCglobal Architecture Framework Version 1.0*. http://www.epcglobalinc.org/standards
2. *EPC - Electronic Product Code*. See EPC Tag Data Standards at http://www.epcglobalinc.org/standards
3. *GTIN - Global Trade Item Number*. http://www.uc-council.org/ean_ucc_system/pdf/GTIN.pdf
4. *SSCC - Serial Shipping Container Code*. http://www.uc-council.org/ean_ucc_system/pdf/SSCC.pdf
5. Vanalstyne, M., E. Brynjolfsson, and S. Madnick, *Why not one big database? - Principles for data ownership.* Decision Support Systems, 1995. **15**(4): p. 267-284.
6. *EPC Information Services (EPCIS) v1.0 standard*, EPCglobal Inc. http://www.epcglobalinc.org/standards
7. *EPCglobal Object Name Service (ONS) v1.0*, EPCglobal Inc. http://www.epcglobalinc.org/standards
8. *Domain Name System (DNS)*. http://www.bind9.net/rfc
9. Mealling, M. and R. Daniel, *The Naming Authority Pointer (NAPTR) DNS Resource Record - RFC 2915*. 2000, IETF - Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2915

Executive Summary

When the description of work was written it was considered that before implementing any prototype of the serial level lookup service, requirements from final users would be needed. For that reason, T2.1 and T2.2 were described and oriented to get those requirements from relevant actors, and thus, the description of such a service.

As the work on those tasks was progressing, the work group realized that another step would be needed to ensure success in the prototyping. That task involved consideration of the different models to be implemented, from a high level point of view, including the design for data models, interfaces and data flows and messages.

Users (i.e. clients) would use a Discovery Service to find information resources about a given EPC – but there were several issues still to be resolved:

- From whom will this new service receive this information?
- Would it have access to these sources of information?
- How should privacy and security be handled?
- Will the service be synchronized to the client?
- Will it take days to get an answer?
- What is the granularity of data stored on the serial level lookup service?
- How will the lookup service store the information considering that the potential number of records can grow rapidly?

There were many other questions without a single answer, and depending on those answers the design of a serial lookup service would vary significantly.

This document starts in section A with the description of the high level design we have decided. It is important to point out that despite being a Bridge design, it includes ideas form external sources. Furthermore, during the discussions within WP2, contact with external companies working on lookup services has been established and through those a wider understanding of the possibilities has been obtained. This document, D2.4 section A, provides the reader with a good understanding of the WP2 proposal and the design that AIDA and AT4 wireless will follow for implementing the prototype in D2.3.

The second section on this deliverable is the result of many discussions held during the months while the task was active. The document begins by presenting all the possibilities we have considered and makes an evaluation of them, in order to justify the choice taken. Different models are presented with different operating mechanisms and message flows, although they aim for the same basic performance objectives.

Finally, the document is completed with section C, which deals with the selected technology to implement the storage component in the serial level lookup service, which is a critical component, whose robustness and reliability of service will impact upon the response times, management of records, interface to low level components of the network, interface to clients making queries, and so on.

Building **R**adio frequency **ID**entification for the **G**lobal **E**nvironment

High level design for Discovery Services
Section A: High-level Design

Authors: University of Cambridge, AT4 wireless, BT Research, SAP Research

15 August 2007

This work has been partly funded by the European Commission contract No: IST-2005-033546

Revision History

Version	Date	Author	Summary of Changes
0.1	Nov 2006 – March 2007	Mark Harrison	Design discussion documents, with comments and feedback from AT4 wireless, BT, SAP, ETH Zurich
0.8	1st April 2007	Mark Harrison	Initial draft
0.9	17th April 2007	Mark Harrison	Further revisions, added UML diagrams
0.95	29th June 2007	Mark Harrison	Revised UML diagrams, included XML schema
0.97	2nd July 2007	Mark Harrison	Included comments from Trevor Burbridge (BT) re Access Control Policies and included discussion of standing queries
1.0	6th July 2007	Mark Harrison	Proof – reading of the document
	26th July 2007	Nicholas Pauvre (GS1 France)	Bridge Internal review Chapter 1.3 Clarification requested about ONS role in EPC Network Some lines are in different font type
1.1	02th August 2007	AT4 wireless, Cambridge	Correction included following review comments

Note

The views expressed in this document are the views of the joint authors and the *Community* is not liable for any use that may be made of the information contained herein.

CONTENTS

1 Introduction

The primary role for 'Discovery Services' is to provide a mechanism to allow computer systems and application software to find the network addresses of information resources that provide more detailed information about an individually identifiable physical object, especially where those resources are distributed across multiple organizations within the object's supply chain or lifecycle history.

Discovery Services are intended to be lightweight and primarily provide links to services such as EPCIS instances where more detailed event information can be retrieved directly via a secondary query. In this section, we compare the purpose of EPCIS with the purpose of Discovery Services and provide an analogy with equivalent roles within the more familiar World Wide Web.

1.1 Purpose of EPCIS

EPC Information Services (EPCIS) allow organizations to provide a standard query interface for retrieval of detailed EPC-related information stored within information systems and databases.

Much of the data provided by EPCIS consists of 'events' such as observations of the object in particular locations within the premises of a company, as well as actions performed on it (e.g. packing, unpacking, shipping, receiving). In future, this information may also include sensor measurements associated with the object or its environment (from which a temperature history may be determined).

The event data at the EPCIS level may be much richer than data provided at the Application Level Events (ALE) level because the EPCIS data model can answer not only the basic questions "what was seen?", "where?", "when?" – but also "why was it there?", "how was it, when it was seen?", because the EPCIS data model allows for a number of additional descriptive data fields to be specified per event, to provide annotations about the additional business context corresponding to each event. These are also known as meta-data fields.

For EPCIS, a query syntax is defined, that allows the client to specify multiple constraints in order to filter the events to return only those matching all specified constraints. The results are then formatted according to a standardized schema and returned to the client.

EPCIS supports both one-time queries, in which a synchronous response is returned in reply to a query, as well as long-running standing queries, where a client wishes to receive all future updates about new events received by the EPCIS repository, which match the client's specified query criteria. In this case, responses are mainly asynchronous and handled via a callback mechanism.

EPCIS-enabled repositories may be provided for each of several geographic sites in which a company operates – or a company may choose to provide a global corporate EPCIS interface, which draws upon data gathered from its multiple geographic sites.

At all times, each company maintains control over its own data and determines who is allowed to access the data, which records they are allowed to access, and how much detail to provide to the client making the query.

EPCIS provides a common 'language protocol' for inter-company information exchange but does not compel anyone to 'speak' – each organization can choose who to communicate

with, and how much to say. A standard query interface and data format only helps organizations understand how to ask the question and understand what is being communicated in the response.

1.2 Purpose of Discovery Services

Discovery Services play a complementary role to EPC Information Services. They are intended to provide the links or addresses, to enable a client to locate sources of information providers and thereby gather more complete information from multiple organizations that have recorded some information about an object during some stage within its lifecycle. In other words, they help a client locate multiple providers that the client can then query (via an EPCIS standard query) for more detailed information.

Discovery Services are not required to hold a replica of each of the detailed events stored within EPC Information Services across an object's individual supply chain or lifecycle. It is sufficient if a Discovery Service provides at least one link to the address of each EPC Information Service that claims to hold some information about a specific EPC or EPC range, even though each EPCIS may hold several events relating to a particular EPC.

Given that information about the flow of objects is commercially sensitive, it is unlikely that companies will provide public or anonymous access to their EPCIS repositories. Therefore, unlike websites that can be crawled or indexed automatically by search engine software, it is likely that a Discovery Service will only link to a particular EPCIS if the owner of that EPC Information Service actively decides to explicitly publish a record to a Discovery Service to inform that Discovery Service that their EPCIS holds more detailed data for a specified EPC.

In highly regulated industry sectors, it is possible that regional, national or supranational regulatory bodies may in future require publishing of records to a particular Discovery Service for various traceability reasons, such as traceability of foods, pharmaceuticals, other safety-critical products, such as aircraft parts or for planning for emergency situations, such as pandemics, where it is essential to know the location of stockpiles of vaccines and medication.

However, normally companies would be free to choose whether or not to use Discovery Services in much the same way that they can opt into being listed in a telephone directory, yellow pages directory or other business directory. There may be benefits to a consortium of trading partners deciding to use a Discovery Service, in order to improve supply chain visibility and transparency (although it should be noted that this is rarely desired by all parties) – or to improve efficiencies, such as being able to do more selective product recalls.

Without Discovery Services, the sharing of information is based upon prior knowledge of the network addresses or URLs for their information services within a cluster of organizations, although an organization may lack the detailed knowledge about which other organizations have information about a specific object. This is much like the exchange of phone numbers on pieces of paper in the absence of any telephone directory or consulting a yellow pages directory that lacks any categorized partitioning of the entries.

Discovery Services can be implemented as directory services - but unlike a telephone directory, they are concerned with highly fragmented sources of data and highly dynamic links between those fragments. The 'trace' or trail of previous custodians for a given object might be completely unique to that particular object - and might never be repeated exactly for other objects of the same type nor fully known in advance when the object begins its journey.

In a sense, Discovery Services can provide an element of robustness or redundancy to the process of gathering information, since their directories can return a list of links to multiple

providers, whereas in their absence, the client would need to locate the manufacturer's EPCIS (e.g. via on ONS query) and then attempt to follow onward links from one organization to the next, if this information was indeed provided by the EPCIS. The major vulnerability of the 'link traversal' approach is that if the EPCIS of any of the intermediate parties were unavailable (e.g. due to temporary power/connectivity outage or permanent cessation of trading), it may become impossible to navigate the missing onward links to downstream parties. If each organization publishes a link to a Discovery Service, then we avoid the risk of a single point of failure within the supply chain, provided of course that each Discovery Service itself has guaranteed availability; in this situation all other organizations remain reachable. These different approaches are illustrated in Figure 1.

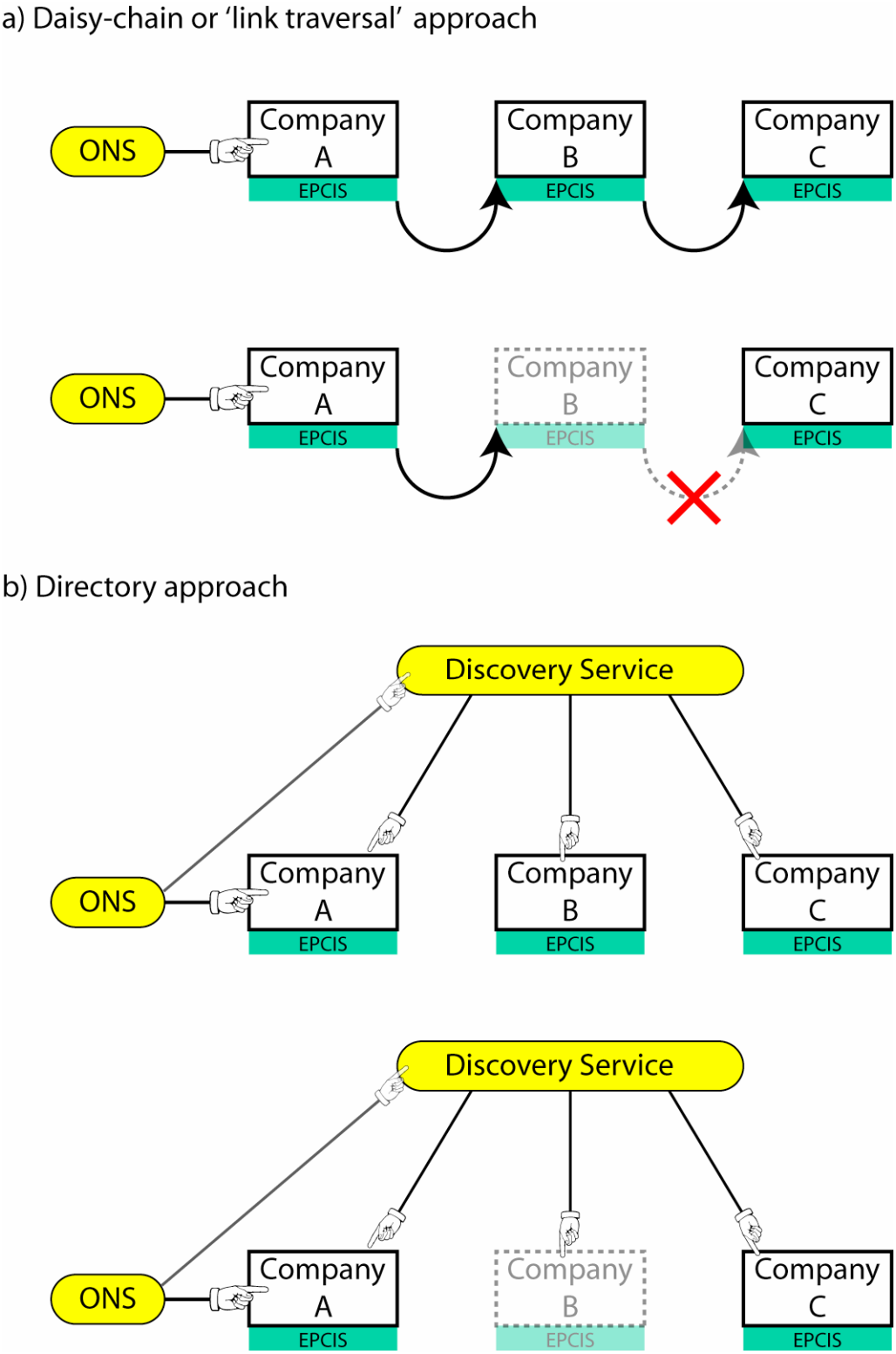


Figure 1. With a daisy-chain or 'link traversal' approach (a), a client follows links from one company to the next but may not be able to track beyond a company whose information service is unavailable. With a directory approach (b), a client able to locate a Discovery Service can directly request the links to multiple information services that provide data for a specific object. Even if one company is unavailable, the link information continues to be provided by a Discovery Service, ensuring that downstream organizations can still be reached even if an intermediate organization is temporarily or permanently unreachable.

In order to gather complete lifecycle information from the fragmented sources, Discovery Services need to provide 1-many links associating each individual serialized object with one or more information providers (or more specifically, the network addresses of those providers' information services).

The total volume of traceability information that Discovery Services may need to store is the product of three factors, namely:

- the number of additional individual objects being tracked per year
- the number of custodians for each individual object
- the retention time for the Discovery Service records associated with a particular object

The retention time for records Discovery Services may vary depending on the type of object and the industry sector to which it belongs:

- For tracking of shipments, the records might only need to be stored while the shipment is in transit and has not yet reached its final customer (e.g. a few days to a few weeks)
- For some objects (e.g. consumer goods/retail sector), the primary interest may be tracking from manufacturer to point of sale (e.g. a few days to a few months)
- In some sectors (e.g. pharmaceuticals), regulatory guidelines may require records to be retained for several years beyond the point of dispensing.
- In other sectors (e.g. aerospace parts), the lifecycle up to the point of delivery is only the initial phase of the lifecycle of the part - and there can be significant interest in tracking the part (and its sequence of custodians and information providers) throughout its active service life, which may be up to 30 years for some parts.

One situation where tracking of an object may become more difficult is if its own identity becomes unreadable for a period of time. Examples of this situation include:

- barcoded components and sub-assemblies being installed within a composite product and subsequently obscured from view
- objects tagged with passive RFID tags being loaded within large metal shipping containers or vehicles during transit, in the situation where the container or vehicle is not equipped with the ability to read its contents
- bulk product with a unique identifier being broken down into a number of smaller objects (with their own unique IDs) for downstream distribution / end-user consumption.
- re-labelling of product with a new identifier

In these situations, it may be appropriate to record aggregation events within the EPCIS of the organization that causes this change of aggregation. It may also be helpful to be able to record an aggregation record also at the Discovery Service layer, to provide for robust end-to-end tracking in the event of non-availability of the EPCIS of the organization that recorded the change of aggregation. Our design for Discovery Services allows organizations to choose to publish an aggregation record and provides some additional optional query parameters, which a client may use to select which aggregation records are of interest. Note that in the case of aggregation records within the Discovery Service, additional parent/child and action information will be provided in addition to the usual `serviceAddress` URL and `serviceType` data. It is not expected that Discovery Services will automatically perform any recursive queries or proxy queries for identities other than the one specified by the client; i.e. the Discovery Service is not required to switch to tracking the parent / children nor have the

'intelligence' to attempt this. Such a role may be handled by tracking models developed in BRIDGE WP3 but are outside the scope of basic Discovery Services in BRIDGE WP2.

1.3 Comparison between the World Wide Web and the EPC Network

Table 1 provides an analogy between the components of the EPC Network and the corresponding components of the World Wide Web.

Purpose	World Wide Web	EPC Network
Primary key for searching for information	keyword	EPC
Provides a list of URL links to sources of more detailed information	Search Engine	Discovery Services
Provides detailed information, usually from one information provider or company	Website	EPC Information Services (EPCIS)
URL to allow computer to connect to a particular information resource	Website address	Address of an EPC Information Service (EPCIS)
Assists the client in retrieving information from directories and various information resources – and may provide visualization of the data (or a derivative of it) in a human-readable format.	Web Browser	Application software
Allow search engines to update and build their directories of links for a specific keyword	Crawling/indexing by search engines	Not applicable – EPC Information Services might not provide any public/anonymous access – and would need to explicitly 'publish' a record to a Discovery Service in order to be found by clients querying that Discovery Service
Allow trails of information to be followed	Human-machine interface (e.g. mouse, keyboard), clicking on hyperlinks	Machine-machine interface
Improve signal/noise ratio, convert data into information, meaning, decisions, actions	Human brain	Human brain + Machine learning, logic, rules, patterns

N.B. In this analogy, there is no element of the world-wide web that plays an equivalent role to that of ONS in the EPC Network.

Basic Concepts

In this section, we introduce the basic concepts and terminology used throughout the remainder of this document. A Glossary of Terms is provided at the end of this document.

EPC – Electronic Product Code (serves as a unique ID for the object and does not necessarily indicate product type (e.g. for SSCC codes))

Client - a user who has some access rights to query a DS for records from within a restricted set of records

Publisher - a user who has some access rights to publish new records to a DS, but where the published records in some way identify the publisher and the supply chain within which those records belong.

User - generic term consisting of both clients and publishers

Discovery Service Record - a single tuple of information that represents a link between an EPC and the URL address of a relevant EPCIS/DS, together with an indication of the ServiceType and additional metadata fields such as (eventTime, recordTime and other meta-data as deemed appropriate). Note that each record is provided by at most one publisher; subsequent custodians of the object may publish additional records that share the same EPC but are distinct records, rather than extensions of an existing record. This approach makes it easier to enforce immutability of records, since each record is published under the authority of a single organization and can be digitally signed by that organization, independently on any other records within a particular Discovery Service.

Trace History - a list of URLs of EPCIS/DS services for the specified object, extracted from all Discovery Service records that share the specified EPC.

Query – the act of requesting information from a Discovery Service, usually by specifying an EPC of interest and providing the user's authentication credentials, and optionally additional constraints. Also refers to the query message sent to a Discovery Service to request information.

Query response – the information provided by a Discovery Service in response to a query. This typically consists of a subset of information extracted from multiple Discovery Service Records all concerned with the specified EPC. At a minimum, the query response provides a set of links to additional EPCIS services and perhaps also to additional Discovery Services.

Publish – the act of creating or adding a new record to a Discovery Service.

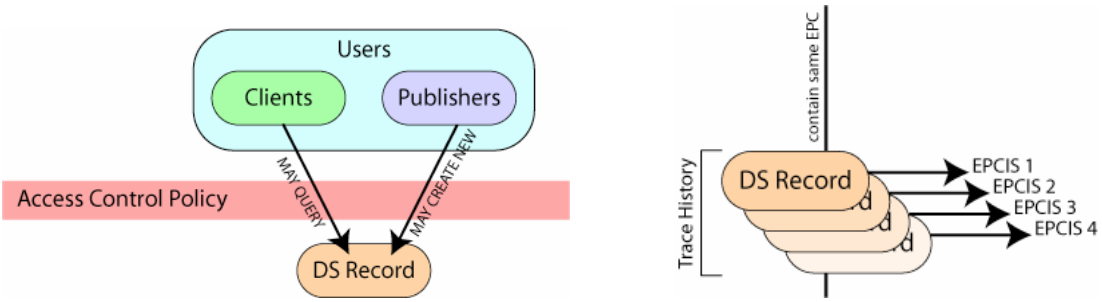


Figure 2 – a graphical representation of the relationship between clients, publishers, records and trace history.

Mutual Authentication - a process by which the authenticity of the identity of the user is verified by the Discovery Service and the authentic identity of the Discovery Service is verified by the user.

Authorization - a process in which the Discovery Service checks a user's entitlement to:

1) interact in any way with the Discovery Service interfaces for query, publishing and subscription (standing queries)
2) membership of particular supply chains or supply chain fragments hosted on that particular Discovery Service
3) membership of particular 'clusters' or 'privileged trading partner groupings' within supply chains or product lifecycles.

Access Control - a mechanism that is used to ensure the privacy of Discovery Services to a appropriate authenticated authorized users.

Access Control Policy – a set of rules that govern whether a specific user is permitted to query a specific Discovery Service record – or whether they are allowed to publish (create) new records within a Discovery Service for a particular object or EPC.

Supply Chain - A supply chain consists of multiple users who are involved in the lifecycle of a product, normally from its point of manufacture (but possibly also including suppliers of components and raw materials) through to its point of sale/dispensation/delivery to customer (and possibly extending beyond this point to include after-sales / in-service repair operators, after-life recovery etc.). The extended supply chain may also consist of other parties (e.g. insurance companies) who never 'touch' the physical part - but nevertheless hold records relevant to it.

Cluster - A subset of one or more organizations within a supply chain that have a trusting business relationship and agree to sharing of information that is more detailed (or more privileged) than the information that is available to ordinary members of a supply chain.

2 Data Model

In this section, we discuss the details of the data fields that may be included within Discovery Service records, as well as the interfaces to the Discovery Services, their methods and input parameters and the resulting information provided in response.

2.1 Interfaces and data formats to be defined

Figure 3 shows the relationship between the interfaces of a Discovery Service and the interfaces of EPC Information Services. Both have an interface though which new events or records can be added (i.e. the EPCIS Capture Interface and the DS Publisher Interface). Both also have a query interface, through which information can be retrieved (i.e. the EPCIS Query Interface and the DS Query Interface).

However, whereas the EPCIS returns a filtered set of EPCIS events in response to a query, our proposal is that it may not be appropriate for a Discovery Service to return an appropriate selection of its internal DS records in their entirety – but rather, it should extract a time-ordered list of the URLs and corresponding ServiceType fields from those records and return these as the response to Discovery Service queries. This is illustrated in Figure 3 below.

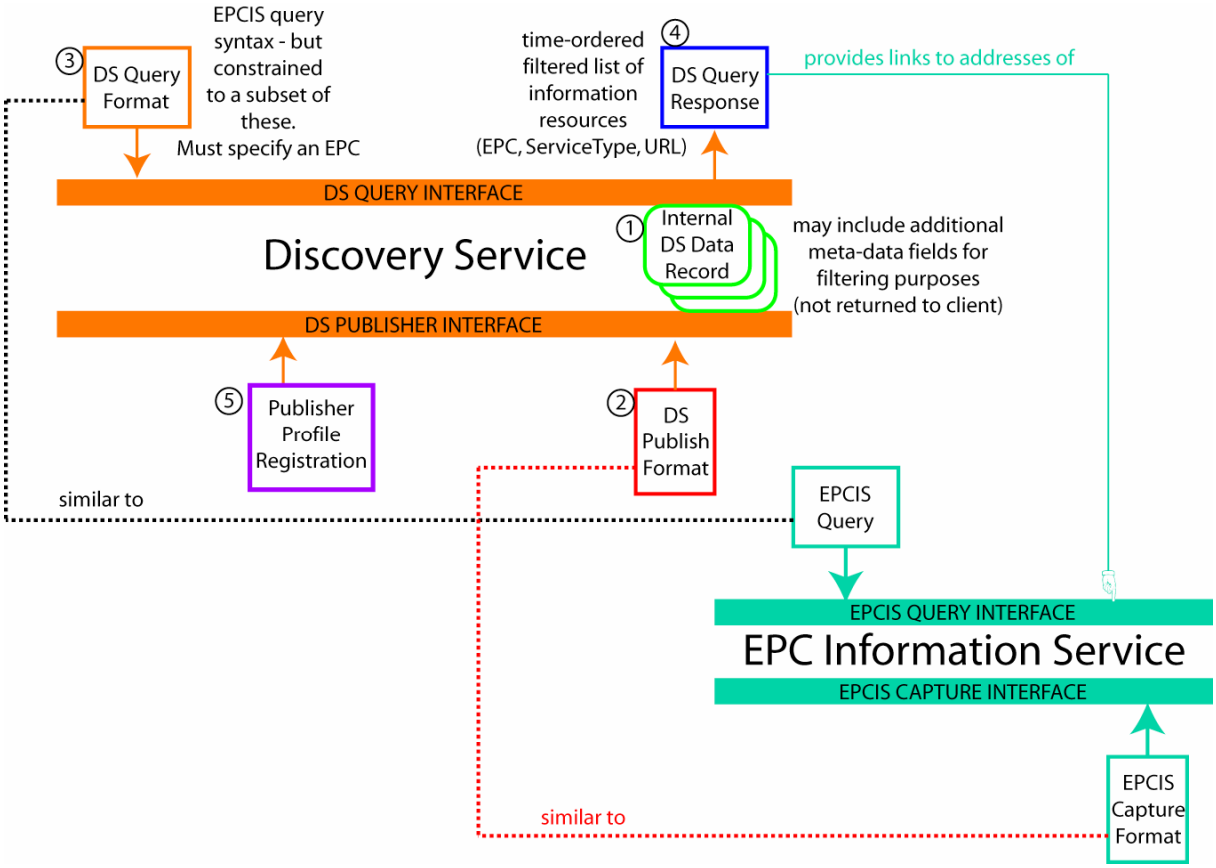


Fig. 3. – Interfaces and data formats to be defined for Discovery Services

From a functional perspective, there is a data class (1) to be defined for the internal record held by the Discovery Service – and additionally three message formats to be defined, (2), (3), (4). It is important to note that these four data structures, in particular (1), (2) and (4) are not identical to each other. Although (1) and (2) may be very similar, the internal data record (1) should also include a timestamp to indicate when the Discovery Service received the record (i.e. recordTime). In addition, a publisher profile (5) is used to store the URL of the

serviceAddress, to avoid embedding this information in each record – and to allow flexibility to change the serviceAddress, should the need arise, as discussed in section 3.2

2.2 Avoiding problems encountered in the world wide web

Two serious problems faced by users of the World Wide Web are broken hyperlinks and phishing or fraudulent addresses.

2.2.1 Broken hyperlinks and fragility of URL addresses

A hyperlink is considered to be broken if the URL address fails to resolve to the expected information resource. Often, an HTTP 404 'Page Not Found' error is displayed. Hyperlink information is very fragile and easily broken unless website developers take due care to ensure that resources are provided with permanently reachable addresses. Some of the reasons for broken hyperlinks include:

- Non-renewal of domain names
- Change of domain name due to company takeovers, consolidation, de-mergers and re-branding
- The link stops working due to a reorganization of the path names within an organizational website (e.g. directory hierarchies)
- Changes to the technology used for server-side delivery of pages (e.g. switching between CGI, ASP, PHP, JSP, etc.)
- Use of filename suffixes (e.g. .htm, .asp, .php. .jsp) and inconsistent use thereof
- Deliberate withdrawal of information
- References to bulletin boards, message lists, blogs etc. in which older messages are archived to different addresses – or deleted
- The address needs to be changed because it has been compromised, e.g. via persistent Denial-of-Service attacks

In the context of Discovery Services, some of the same reasons may also affect the permanence of the URL address for an EPCIS service. In much the same way that it is difficult to correct all the broken hyperlinks on the web, it may be difficult to systematically change the URL address for a particular EPCIS service for all affected records, especially if the URL address is embedded within a digitally signed record provided by the publisher.

2.2.2 Phishing and fraudulent addresses

Phishing is a term widely used to refer to the use of fraudulent web addresses for the purposes of deception, often for the purposes of harvesting information from the user, such as user-IDs and passwords for various accounts, internet banking details, etc. This is much more sinister than broken hyperlinks, since users will often be presented with a website or web page that superficially resembles a familiar page, although any data that the user supplies via a form on that page is transmitted to a potentially malicious user rather than the genuine organization that the user believed that they were interacting with.

In the context of Discovery Services, this highlights two major issues:

- It is important to protect the privacy of information provided by the client and ensure that it is not divulged to malicious parties
- It is important to support verification of records published to a Discovery Services by information providers, in order to exclude malicious parties.

Digital signatures can be used to prove the authorship and authenticity of digitally signed information, since a digital signature depends both upon the information content and upon maintaining secrecy of the private key used when signing the information. It is possible to

verify digital signatures by decrypting the signature with the corresponding public key of the author and comparing the result with the hashed value of the data, using the hashing and encryption algorithms that are specified with the digital signature.

It may be desirable for Discovery Services to accept and verify digitally signed records from publishers – and either to return the original digitally signed record to a client in response to a query, or in the situation where it is not appropriate to return the original record in its entirety, to return an indication of whether the Discovery Service itself was able to verify the authorship and authenticity of the record from which the response is derived.

If digitally signed records are published to a Discovery Service and it is unable to verify the signature, then the record should be rejected and a signature verification exception should be raised via the publisher interface.

2.2.3 Implications for the design of discovery services

Given that Discovery Services may hold vast numbers of records relating to a particular URL address, it may be a good idea to avoid embedding the URL in each of those records. If each of those records instead contains a pointer or immutable cross-reference ID to a single publisher profile record, and the publisher profile record in turn contains the `serviceType` and URL of the `serviceAddress`, then it may be much easier to effectively update all affected records by merely updating the URL of the `serviceAddress` in the publisher profile rather than updating each of the affected records. The decoupling of the URL of the `serviceAddress` from the record also has a further advantage because the record and the publisher profile record can both be digitally signed by the publisher – and in the case where it is necessary to change the address for multiple records, only the publisher profile record needs to be updated with a new signed copy, rather than updating and re-signing all affected records.

Furthermore, in the case that the records are digitally signed, such a change need not alter all the records or invalidate the previous digital signatures, since an immutable cross-reference identifier (the publisher profile ID) could be embedded within the published record, instead of embedding a literal URL address of an EPCIS service. Figure 4 indicates the proposed decoupling of the URL address from the record, instead embedding a permanent immutable publisher profile ID as a cross-reference to the publisher profile record that holds the correct URL of the `serviceAddress` for those records.

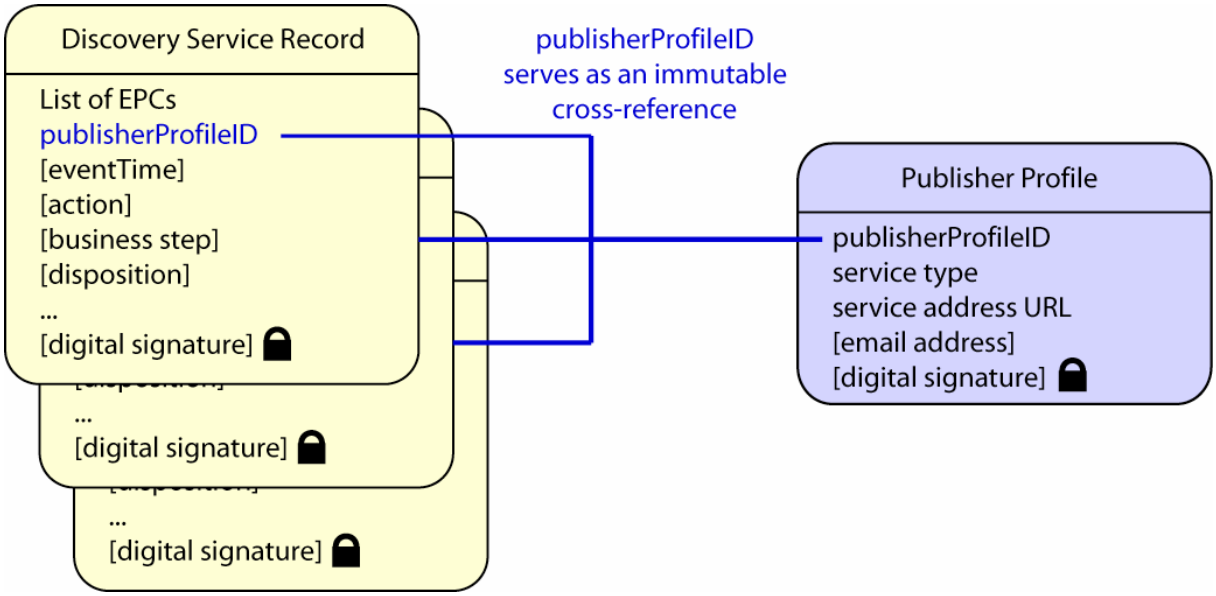


Figure 4 – Decoupling of the URL address from a record published to a Discovery Service. A publisherProfileID serves as an immutable cross-reference for joining to details of service address and type, which may need to be altered later. This decoupling approach removes the need to change digital signatures for each Discovery Service record when the service address URL needs to change.

This decoupling approach is intended to provide flexibility to the publisher. These details should be hidden from the client's query interface. Internally, a Discovery Service is still expected to return the `serviceAddress` URL and `serviceType` and should never return the publisher profile ID to the client. Effectively, the Discovery Service should perform an inner join between each record and the publisher profile record, joining on the publisher profile ID.

It is expected that many records within a Discovery Service may share the same `serviceAddress` URL and that the number of distinct `serviceAddress` URLs will be much smaller than the number of distinct records held within a Discovery Service, so it may be entirely reasonable to maintain a lookup table or indexed map of the associations between a publisher profile ID and the corresponding `serviceAddress` URL.

As discussed in the following section 3.3, in our design for Discovery Services, the records published to a Discovery Service will usually not be returned verbatim to a client's query – but rather, they will be used as the basis for constructing a response, returning a subset of the data fields, such as `serviceAddress` URL and `serviceType`. It is therefore appropriate for a Discovery Service to indicate whether or not it was able to verify the digital signature of signed records from which the response was assembled. This is indicated via a data field named `Status`.

Some publishers may need to specify multiple service addresses – and they may do so by registering multiple publisher profiles with a Discovery Service and allocating one `serviceAddress` URL and `serviceType` to each publisher profile.

2.3 Response from a Discovery Service

In order to minimize the disclosure of commercially sensitive information, the DS Query Response (block 4 on Fig. 3) need NOT return the entire set of internal records (block 1 on Fig. 3) matching the query criteria. Rather, it should extract the `ServiceType` and `ServiceAddress` from the publisher profile corresponding to each record and returns a list of such tuples, sorted chronologically by default.

In our proposal for Discovery Services, the result set returned to a client in response to a query will NOT consist of one or more internal DS records – but rather, will consist of a time-ordered filtered list of information resources, where each information resource consists of the tuple (`ResponseCode, EPC, ServiceType, URL, Status`).

Additional data fields held in internal DS records will generally not be communicated back to the client – but will only be used within the Discovery Service for filtering purposes. The Discovery Service security policies should specify whether the additional data fields may be used for the client for search purposes or additionally returned to the client. It is important that the security policies also cover permitted search criteria in the absence of additional fields being returned to the client since multiple queries could otherwise be used to infer the value of such data fields.

2.4 Publishing a record to a Discovery Service

It is proposed that the DS Publishing format should align closely with a subset of the XML schema proposed for the EPCIS Capture Interface, although it should be understood that the actual schema used for the Discovery Service may be more restrictive in the sense of requiring an EPC data field to be specified – and limiting the event types to a subset of those allowed in the EPCIS data model.

Furthermore, a number of metadata fields of EPCIS events (such as readPoint, businessLocation) are omitted from our proposed schema for Discovery Service events. We also omit the eventTimeTimezoneOffset field, since the eventTime field already fully specifies the timezone in both EPCIS and DS and the eventTimeTimezoneOffset can be retrieved by the client via a subsequent query to an EPCIS, if specifically required.

Note also that Discovery Service sets the value of the recordTime data field as the time of receipt of the record by the Discovery Service. For this reason, the schema for publishing (block 2 of Fig. 3 and section 10.5) will omit the field ‘recordTime’, although the query format (block 1 of Fig. 3 and section 10.8) may support constraints on recordTime in a very similar manner to the way in which the EPCIS query interface supports this.

2.5 Basic Discovery Service Record

We assume that each organization that handles a particular EPC publishes a Discovery Service record in order that the Discovery Service can provide a link to the address of their EPCIS when queried about that EPC by trusted partners. It should be noted that although an individual organization may hold several EPCIS events relating to that EPC, they might only publish a single Discovery Service record in order to establish a link to their EPCIS for this more detailed event information.

Each time an organization publishes a record to a Discovery Service, a new source of link information is added to the Discovery Service. At a minimum, an internal Discovery Service record logically needs to provide the following data fields:

2.5.1 Data fields for a record provided by the publisher

Data Field	*Data Type*	*Description*
`EPC`	Pure-identity EPC or Pure-identity EPC pattern	An EPC or EPC range for the objects in this record.
`ServiceType`	ServiceTypeID	Indicates the type of service provided at the address. Provides the client with information about whether the record links to either an EPCIS or to another Discovery Service. Allowed values are "EPCIS" or "DS". Future extensions are allowed
`ServiceAddress`	URL String	An address for an information resource linked from the Discovery Service

Note that in practice, the ServiceType and ServiceAddress may in fact be stored within a Publisher Profile – and the internal Discovery Service record refers to the ID of a particular Publisher Profile rather than embedding this information within each record.

2.5.2 Additional data fields for a record that are asserted by a Discovery Service

Data Field	*Data Type*	*Description*
`RecordID`	Unique ID	Used internally by the Discovery Service to provide a cross-reference mechanism between a record and the access control policies that govern which clients may receive it as a result of their queries.
`RecordTime`	Time	The timestamp at which the Discovery Service received this record from the publisher. The timestamp should be expressed with resolution of 1 second and be timezone qualified, relative to UTC. i.e. YYYY-MM-DDThh:mm:ssTZD (e.g. "1997-07-16T19:20:30+01:00")
`PublisherID`	String	Identifies which publisher provided this record. This could be a globally unique reference to the publisher's digital certificate – e.g. the certificate authority's ID and the certificate ID or serial number within that certificate authority.

`ServiceTypeID` is a string that should indicate whether the URL corresponds to an EPCIS or a DS. This is analogous to the Service Type field in ONS records – it is merely a helpful clue for the client application. Allowed values of `serviceTypeID` are either "`EPCIS`" or "`DS`". Further allowed values of `serviceTypeID` may be defined in the future.

`recordTime` is the date and time when the record was received by a Discovery Service. This is an internal timestamp recorded by a Discovery Service, which may be used for sorting records into chronological order (especially if no `eventTime` is specified). It may also be used to ensure completeness of responses to standing queries.

The publisher should not specify a `recordTime` – and any value provided by the publisher should be ignored by the Discovery Service, since `recordTime` is an internal timestamp recorded by the Discovery Service upon receipt of the record.

recordTime should be specified in a format consistent with ISO 8601. See for example the formats of timestamp strings in http://www.w3.org/TR/NOTE-datetime

e.g. YYYY-MM-DDThh:mm:ssTZD (eg “1997-07-16T19:20:30+01:00”)

For traceability purposes, a record should also contain a data field, PublisherID – which is not merely extracted from the body of (2) the record that is published to the Discovery Service, but rather, is extracted from the authentication credentials when the publisher authenticates with the Discovery Service before publishing. For example, the publisher might digitally sign a record before publishing it the Discovery Service. In this case, it may be useful for the internal Discovery Service record to store the digital certificate within the internal DS record (1).

Clearly a PublisherID field that has been obtained from a digital signature and verified before inclusion in the Discovery Service is much more trustworthy than if the Publisher ID were merely asserted as an unsigned data field within the publisher record (2).

It may even be appropriate for the response (4) from the Discovery Service to indicate whether or not a particular Publisher ID was verified by that Discovery Service by checking the digital signature when it received the record from the publisher. For this reason, the data field ‘`Status`’ is proposed.

2.5.3 Optional metadata fields

There are two reasons why it may be beneficial for a Discovery Service to support a minimal number of optional metadata fields:

1) To allow the client querying the Discovery Service to receive a more limited number of records by using additional metadata fields to constrain (limit) which records are retrieved according to some criteria.

2) To allow the publisher to use the metadata fields within access control policies as a way of more precisely specifying which records should be made available to which client. e.g. a publisher may choose to only share records about shipping with its customers and only share records about receiving with its suppliers. (See section 9)

2.5.3.1 Optional metadata fields which the publisher may provide:

Data Field	*Data Type*	*Description*
`action`	DSAction	A string from an enumerated list. This is used to indicate how the record corresponds to a particular stage in the lifecycle of the object. For basic records, allowed values are “LINK” (default), “CREATE”, “CLOSE”, “DESTROY”
`businessStep`	BusinessStepID	A vocabulary whose elements denote steps in the business process. e.g. an identifier that denotes “shipping”.
`disposition`	DispositionID	A vocabulary whose elements denote a business state of the object after the event happened. e.g. an identifier that denotes “available for sale” or “received”.
`eventTime`	Time	The timestamp asserted by the publisher for this record. The timestamp should be expressed with a resolution of 1 second and be timezone qualified, relative to UTC. i.e. YYYY-MM-DDThh:mm:ssTZD (e.g. “1997-07-16T19:20:30+01:00”)

`BusinessStepID` is a vocabulary whose elements denote steps in the business process. An example would be an identifier that denotes "shipping". The business step field specifies the business context of an event – i.e. what business process was happening that caused the event to be captured. In EPCIS, the `BusinessStepID` identifiers are usually expressed as URN strings within a standardized namespace (that may be specific to a particular industry sector).

`DispositionID` is a vocabulary whose elements denote a business state of the object after the event happened. An example would be an identifier that denotes "available for sale" or "received". In EPCIS, `DispositionID` identifiers are usually expressed as URN strings within a standardized namespace (that may be specific to a particular industry sector).

`eventTime` is the date and time asserted by the publisher of the record for the appropriate timestamp corresponding to the record. `eventTime` allows some real-world ordering, but cannot be trusted, whereas `recordTime` has the opposite characteristics.

`eventTime` should be specified in a format consistent with ISO 8601. See for example the formats of timestamp strings in http://www.w3.org/TR/NOTE-datetime
e.g. YYYY-MM-DDThh:mm:ssTZD (eg "1997-07-16T19:20:30+01:00")

For basic Discovery Service records, we propose that the action field takes different values from those allowed for EPCIS. We define the following allowed values of the action field:

`CREATE` – used only when an object is physically created for the first time

`LINK` – (default value, assumed if no action is explicitly specified) – indicates a link

`CLOSE` – used to indicate that the chain of custody for the forward logistics has reached its normal final end point (e.g. point of sale / dispensing). (Useful for assisting in detection of duplicate EPC records, due to counterfeiters using discarded packaging to re-insert fake goods into the forwards supply chain). Normally, only genuine reverse-logistics processes should result in new DS records following a DS record with an action marked '`CLOSE`'.

`DESTROY` – used only when the physical object has actually been destroyed. Normally, no further records should follow a DS record with an action marked '`DESTROY`'.

2.5.4 Handling of optional/missing fields

A Discovery Service is not required to store each of these optional metadata fields in its internal records, although it should record a `recordTime` of the time when it received a record from a publisher.

In the case where a client's query to a Discovery Service involves a constraint on a data field that was either not provided by the publisher or not stored / not supported by the Discovery Service, then this constraint shall be ignored for the purposes of filtering the records that are returned as the results to the client's query.

It may be of benefit to publishers and clients of the Discovery Service to provide an interface query method that lists which of these optional metadata fields are supported and stored if published to a Discovery Service. For example, a Discovery Service may provide a method `getSupportedOptionalFields()` that returns an XML response such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_supportedOptionalFields>
    <string>action</string>
    <string>businessStep</string>
    <string>disposition</string>
    <string>eventTime</string>
</ds_supportedOptionalFields>
```

2.5.5 Basic Discovery Service records – UML class diagram

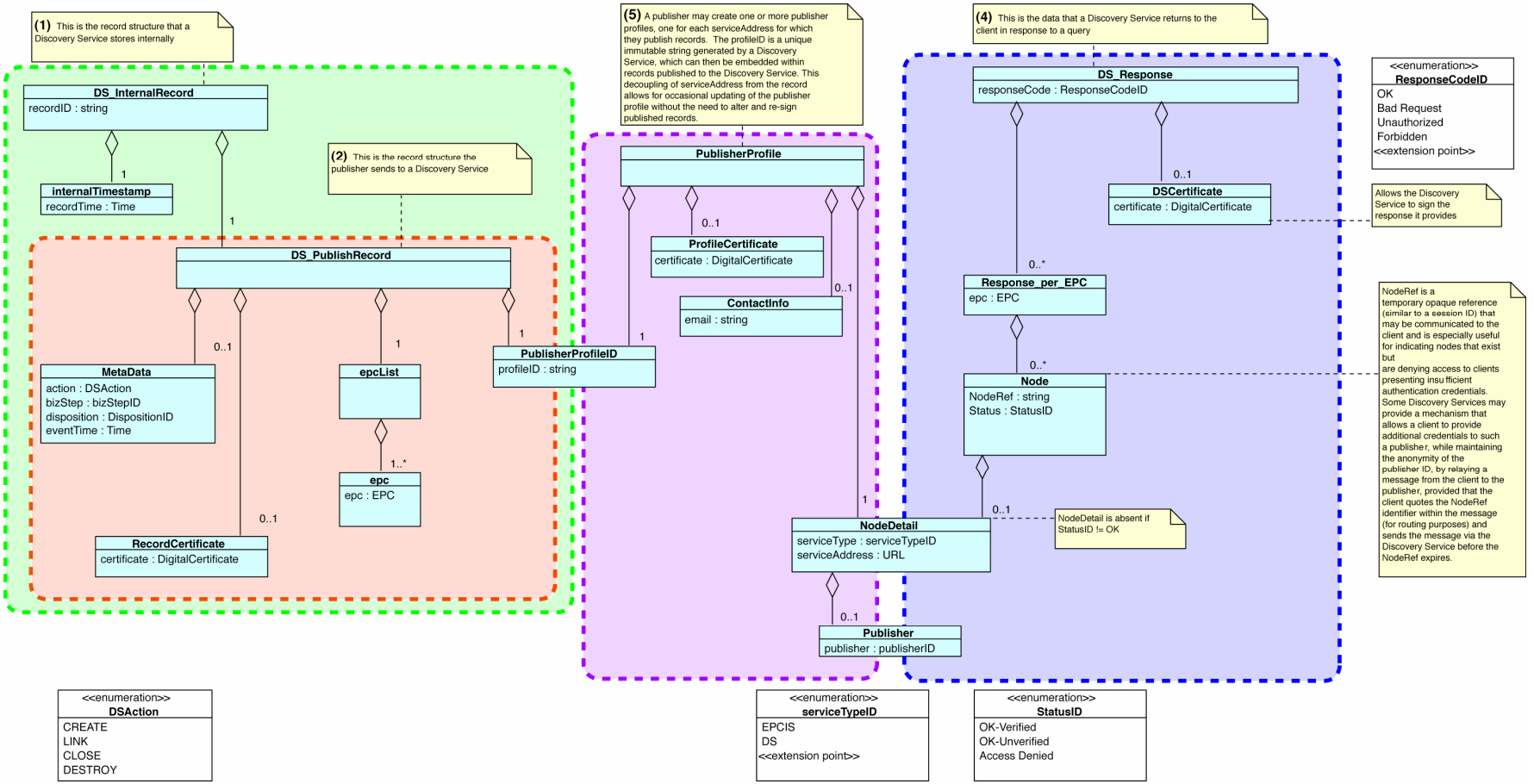


Figure 5 – UML class diagram for basic records within a Discovery Service, showing which data fields contribute to the various message / record formats (1), (2), (4) shown in Figure 3. The numbering and colour-coding of the dashed rounded outlines corresponds to the message formats numbered 1, 2, 4 and 5 in Figure 3. Note that basic records do not provide any details about significant aggregation and disaggregation events. (See Fig. 6 for a UML class diagram for aggregation records)

2.5.5.1 Fields appearing only in a response from the Discovery Service

`Status`	StatusID	Indicates the availability of information and whether the Discovery Service was able to verify the identity of the publisher of this record. • `OK-Verified` indicates that the publisher supplied a digital signature that could be verified • `OK-Unverified` indicates that no digital signature was supplied • `Access Denied` indicates that a node exists but is not providing further information. Note that access control policies should also allow a node to remain silent to particular clients and not even reveal that it exists. Note that a Discovery Service should only store records that are either signed – or records which were digitally signed and for which the signature has been verified; if a record is digitally signed but cannot be verified by a Discovery Service, then this should be rejected as a signature verification exception via the publisher interface.
`NodeRef`	String	A temporary unique ID that can be used to enable a client to send further credentials and an enquiry to a publisher that is initially denying access.

The fields `Status` and `NodeRef` are provided per record in the response from a Discovery Service but are not considered to be part of the internal record held within a Discovery Service, since they depend on the access privileges of the client as set by the publisher of each record – and therefore vary, depending on which client is making the query. They are for information only and cannot be constrained using the usual query parameters described later.

`Status` is used to indicate whether or not a particular information provider (that exists for a given EPC) was willing to provide information to a particular client. It returns a status code, much along the same principles as a web-server returns an HTTP status code. (See reference for HTTP 1.1). We proposed that the status ID should take one of the following three values:

`OK-Verified` The node reveals its address and the Discovery Service verified the Publisher ID by checking the digital signature of the record published by the publisher at the time it was received.

`OK-Unverified` The node reveals its address to the client – but the Discovery Service was not able to verify the Publisher ID using a digital signature

`Access Denied` The node exists but chooses not to reveal its address to this client for this EPC. See NodeRef for a possible negotiation mechanism.

It may be necessary to define additional permitted values of `Status` in a later version of this design.

`NodeRef` is an opaque string that is used as a reference or 'handle' so that the Discovery Service and Client can refer to a particular node that is denying access, without the Discovery Service needing to reveal the identity or URL address of the node that wishes to remain anonymous to the client. This has the benefit of indicating to the client whether a complete set of links has been returned – and provides for future extension features, whereby a client that is initially denied access may send further credentials, an updated digital certificate to replace an expired one, as well as further details of their request to the anonymous node that denies access, by sending the request to the DS and specifying the `NodeRef` as the intended recipient. This only works if the DS is prepared to act as a relay to forward such a client request onwards to the node concerned and if it holds relevant information such as a contact e-mail address for each node or Publisher Profile. To facilitate this optional feature, the Publisher Profile is allowed to include an optional e-mail address as contact information, which a Discovery Service may use to forward such requests to a publisher, but which a Discovery Service should not reveal to any client.

2.6 Aggregation Records within a Discovery Service

The EPCIS data model already provides for an `AggegationEvent` subtype, which can be used to record observations of aggregates as well as explicit changes of aggregation involving an object.

Aggregation is a process in which one or more 'child' objects are aggregated to a 'parent' object. For example the 'child' objects may be the contents and the 'parent' may be a container.

Disaggregation is the reverse process, in which one, several or all 'child' objects are disaggregated from the 'parent' object.

As discussed in Section 1.2, there may be benefit in allowing aggregation records to be held in a Discovery Service, to avoid a situation where it is impossible to track an object further because of unavailability of the EPCIS of the company that recorded the essential `AggregationEvent`.

Our design proposal therefore supports a data model for optional aggregation records within the Discovery Service, although it is for each publisher to choose whether or not to use these.

Figure 6 shows a modified UML diagram for aggregation records.

2.6.1 Aggregation Records – UML class diagram

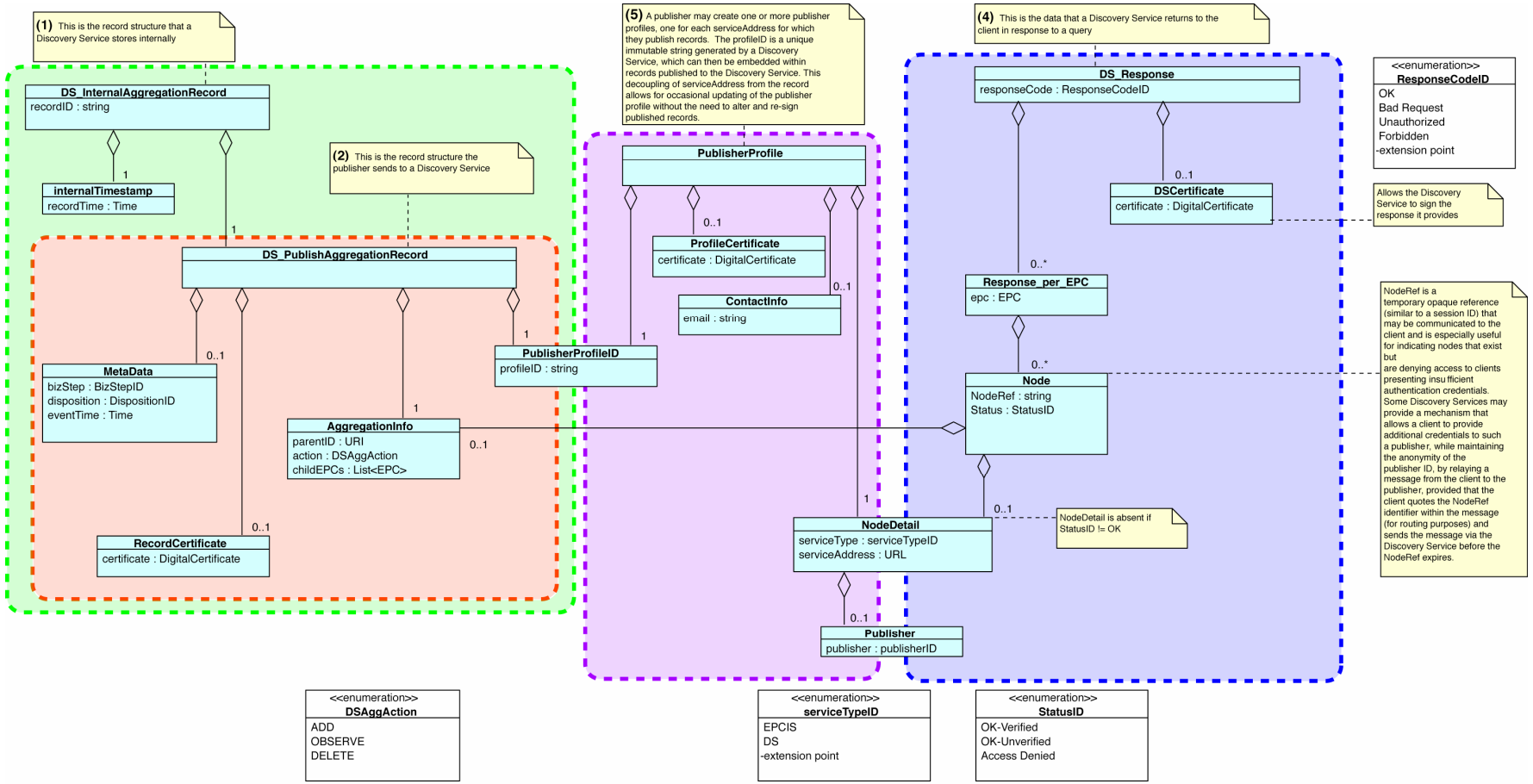


Figure 6 - UML class diagram for aggregation records within a Discovery Service. The numbering and colour-coding of the dashed rounded outlines corresponds to the message formats numbered 1, 2, 4 and 5 in Figure 3.

The permitted values of the `Action` field in our Discovery Service `AggregationRecords` are closely aligned with the values and meanings permitted for `AggregationEvents` within EPCIS, as follows:

`ADD` the EPCs in the `childEPCs` list have been added to the `parentID`. This is used when new children are added to an existing aggregate, as well as when a new aggregate is created for the first time.

`OBSERVE` The event represents neither adding nor removing children from the aggregation. The observation may be incomplete: there may be children that are part of the aggregation but not observed during this event and therefore not included in the `childEPCs` field of the `AggregationRecord`; likewise, the parent identity may not be observed or known during this event and therefore the `parentID` field be omitted from the `AggregationRecord`.

`DELETE` The EPCs named in the `childEPCs` list have been disaggregated from the parent during this event. This includes situations where a subset of children are removed from the aggregation, as well as when the entire aggregation is dismantled. The list of `childEPCs` may be omitted from the `AggregationRecord`, which means that all children have been disaggregated. (This permits disaggregation when the event capture software does not know the identities of all the children.)

It should be noted that unlike basic records in a Discovery Service, the `parentID`, `childEPCs` and `action` fields will be included as part of the response to a query, if they are specified by the publisher, although these fields are subject to any filtering by the publisher's access control policies enforced by the Discovery Service to suppress any EPCs which the client is not authorized to see.

However, both `parentID` and `childEPCs` are optional fields, so it is also permissible for a publisher to publish an aggregation record in which only the `action` field and one EPC within either the `parentID` or `childEPCs` field is specified – and thereby indicate that either an aggregation or a disaggregation event happened (depending on whether `action=ADD` or `action=DELETE`, respectively), without providing details of other EPCs involved in the aggregation/disaggregation event; the client would then need to make a further query directly to the EPCIS of that publisher in order to retrieve the full details via a query for `AggregationEvents` involving that EPC.

3 Publishing to a Discovery Service

3.1 Registration of a publisher profile

As discussed in section 3.2.3, the first step is for the publisher to register a publisher profile with a Discovery Service, in which a `serviceAddress` and `serviceType` are defined. For example, a publisher may register a profile by sending an XML message of the format shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_registerProfile>
    <serviceType>EPCIS</serviceType>
    <serviceAddress>http://www.factorycorp.com/gateway/epcis.wsdl</serviceAddress>
</ds_registerProfile>
```

In response, a Discovery Service should respond with a message that is similar except for the addition of an additional element, `publisherProfileID`, which is a unique permanent opaque string generated by that Discovery Service, which the publisher can cache and later embed within their own records subsequently published to that Discovery Service, whenever the publisher wishes to specify that particular `serviceAddress` for a record. An example of the corresponding response is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_registeredProfileResponse>
    <serviceType>EPCIS</serviceType>
    <serviceAddress>http://www.factorycorp.com/gateway/epcis.wsdl</serviceAddress>
    <publisherProfileID>7D4BA819CE274985F36F</publisherProfileID>
</ds_registeredProfileResponse>
```

(The `publisherProfileID` should be kept private. The publisher may use application software which maintains an internal cache or the lookup table between the `publisherProfileID` and the corresponding `serviceAddress`, for the convenience of the publisher.)

If the publisher ever needs to update the serviceAddress and serviceType for an existing profile, they can post a message specifying both the old and new serviceAddress and serviceTypes. This message should be formatted as below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_updateProfile>
    <previous>
     <serviceType>EPCIS</serviceType>
     <serviceAddress>http://www.factorycorp.com/gateway/epcis.wsdl</serviceAddress>
    </previous>
    <new>
     <serviceType>EPCIS</serviceType>
     <serviceAddress>http://www.newcorp.com/partners/epcis.wsdl</serviceAddress>
    </new>
    <publisherProfileID>7D4BA819CE274985F36F</publisherProfileID>
</ds_updateProfile>
```

Note that it is necessary to specify the previous `serviceAddress` and `serviceType` in addition to the `publisherProfileID`, in order to reduce the risk of unauthorized updates.

3.2 Publishing a basic record to a Discovery Service

Following the successful registration of a small number of one or more publisher profiles, for the `serviceAddress` URLs that the publisher will use, an organization may then publish a new record to a Discovery Service for a single EPC or a list of EPCs. In our design for Discovery Services, EPC pure identity patterns containing wildcards are not accepted within records published to the Discovery Service, although they may be used within query parameters. EPC pure-identity patterns are defined in EPCglobal Tag Data Standards v1.3

An example of an unsigned record published to a Discovery Service is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_publishrecord>
    <ds_header>
        <TTL>P1Y2M3DT10H30M<TTL>
        <ACP></ACP>
    </ds_header>
    <ds_record>
        <publisherProfileID>7D4BA819CE274985F36F</publisherProfileID>
        <action>LINK</action>
        <eventTime>2005-04-03T20:33:31.116-06:00</eventTime>
        <epcList>
            <epc>urn:epc:id:sgtin:0614141.107340.1</epc>
            <epc>urn:epc:id:sgtin:0614141.107340.2</epc>
            <epc>urn:epc:id:sgtin:0614141.107340.7</epc>
            <epc>urn:epc:id:sgtin:0614141.107340.8</epc>
        </epcList>
        <bizStep>urn:epcglobal:epcis:bizstep:fmcg:shipped</bizStep>
    </ds_record>
</ds_publishrecord>
```

The record is asserting that at a particular time, a service holds information about four specified EPCs. The URL `serviceAddress` and `serviceType` are not embedded directly but are specified within the publisher profile corresponding to the specified `publisherProfileID` that is embedded within the record (see previous section 4.2).

A header block may also be provided, in order that the publisher can provide additional information about the record. The header may include information about a time-to-live or retention time, include a reference to an access control policy and may also provide a digital signature for the record.

In the example above, a time-to-live is specified as 1 year, 2 months, 3 days, 10 hours and 30 minutes beyond the time at which the record is published to that Discovery Service.

Upon successful publication, a Discovery Service should respond with an acknowledgment message that specifies the internal recordID that it assigned. An example of this is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_publishedRecordResponse>
    <recordID>2191AB74BE581FC33C</recordID>
</ds_publishedRecordResponse>
```

If a publisher later needs to mark as invalid (void) a previously published record, the publisher may use the method `voidRecord(recordID, reason)`, as defined in Section 10.7. The recordID may also be used by methods that allow extension of the time-to-live value for a specific record.

3.3 Publishing an aggregation record to a Discovery Service

Publishers are not required to publish aggregation records to Discovery Services – and not all Discovery Services are required to support storage of aggregation records. However, if a particular Discovery Service does support aggregation records, a publisher may optionally publish an aggregation record, in order to record important changes of aggregation directly within a Discovery Service, rather than only maintaining the aggregation event within their own EPCIS.

An example of an unsigned aggregation record published to a Discovery Service is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_publishrecord>
    <ds_header>
        <TTL>P1Y2M3DT10H30M<TTL>
        <ACP></ACP>
    </ds_header>
    <ds_aggregationRecord>
        <publisherProfileID>7D4BA819CE274985F36F</publisherProfileID>
        <action>ADD</action>
        <eventTime>2005-04-03T20:33:31.116-06:00</eventTime>
        <bizStep>urn:epcglobal:epcis:bizstep:fmcg:shipped</bizStep>
        <parentID>urn:epc:id:sgtin:0614141.107340.1</parentID>
        <childEPCs>
            <epc>urn:epc:id:sgtin:0614141.107340.2</epc>
            <epc>urn:epc:id:sgtin:0614141.107340.7</epc>
            <epc>urn:epc:id:sgtin:0614141.107340.8</epc>
        </childEPCs>
    </ds_aggregationRecord>
</ds_publishrecord>
```

Note that in this example aggregation record, the publisher asserts that the three specified childEPCs have been aggregated to a parent ID at the time of shipping.

Upon successful publication, a Discovery Service should respond with an acknowledgment message that specifies the internal recordID that it assigned, as explained previously in section 4.2.

4 Querying a Discovery Service

4.1 Query formulation

It is proposed that the DS Query format should closely align with the syntax proposed for EPCIS queries, although it should be understood that all DS queries shall specify an EPC or EPC pure identity pattern in the mandatory `MATCH_anyEPC` query parameter.

The EPCIS query syntax allows a client to constrain the values of one or more data fields of an event to be required to have a particular value, range or to be one of a number of alternatives from a list, in order for that event to be returned as part of the results.

Queries to a Discovery Service for a list of links usually require the unique ID or EPC of the object to be specified.

The simplest query to a Discovery Service is therefore `poll(MATCH_anyEPC=<epc>)`, which returns all links for a particular EPC, which the client is allowed to receive.

However, the client may wish to receive a subset of these records that match particular constraint criteria specified by the client. Further optional query parameters may also be specified to provide additional constraints on the results to be returned.

In this case, the query to a Discovery Service becomes
`poll(MATCH_anyEPC=<epc> [,additional constraints])`.

Note that this approach is subtly different from the EPCIS query interface, in which it is perfectly valid to request all events – or to specify only constraints other than a specific EPC; for example, a client might query an EPCIS for all events with business step = 'shipping' without specifying which objects are of interest; such a query is not allowed in our design for Discovery Services – there is always a mandatory `MATCH_anyEPC` constraint in our Discovery Service design, whereas in EPCIS, even the `MATCH_anyEPC` constraint is optional.

In the case of Aggregation records within a Discovery Service, any records will be considered to match if either the `parentID` or any of the `childrenEPCs` match the EPC specified as the mandatory `MATCH_anyEPC` query parameter. If a client wishes to be specific about whether they want to select only records for which the EPC matches only the parent or only one of the children, they may also include the constraints `MATCH_parent` or `MATCH_childEPC` as additional parameters within the query in addition to the mandatory `MATCH_anyEPC` query parameter.

The following constraints are proposed as useful for filtering of DS records:

Constraint Name	*Constraint Value Type*	*Required ?*	*Meaning*
`MATCH_anyEPC`	List of String	Yes	The result will only include records where the value of any of the `epcList`, `parentID` or `childEPCs` field matches one of the specified values.
`MATCH_parentID`	List of String	No	If specified, the result will only include aggregation records where the value of the `parentID` field matches one of the specified values.
`MATCH_childEPC`	List of String	No	If specified, the result will only include records where the value of any of the `epcList` or `childEPCs` field matches one of the specified values.
`recordType`	List of String	No	If specified, the result will only include records whose type matches one of the types specified in the parameter value. Each element of the parameter value may be one of the following strings: `DSRecord, DSAggregationRecord`
`GE_eventTime`	Time	No	If specified, only records with `eventTime` greater than or equal to the specified value will be recorded in the result. If omitted, records are included regardless of their `eventTime` (unless constrained by the `LT_eventTime` parameter)
`LT_eventTime`	Time	No	If specified, only records with `eventTime` less than the specified value will be recorded in the result. If omitted, records are included regardless of their `eventTime` (unless constrained by the `GE_eventTime` parameter)
`GE_recordTime`	Time	No	If specified, only records with `recordTime` greater than or equal to the specified value will be recorded in the result. If omitted, records are included regardless of their `recordTime` (unless constrained by the `LT_recordTime` parameter)
`LT_recordTime`	Time	No	If specified, only records with `recordTime` less than the specified value will be recorded in the result. If omitted, records are included regardless of their `recordTime` (unless constrained by the `GE_recordTime` parameter)
`EQ_action`	List of String	No	If specified, the result will only include records where the value of the `action` field matches one of the specified values. If omitted, records are included regardless of their `action` field.
`EQ_bizStep`	List of String	No	If specified, the result will only include records where the value of the `bizStep` field matches one of the specified values. If omitted, records are included regardless of their `bizStep` field.
`EQ_disposition`	List of String	No	If specified, the result will only include records where the value of the `disposition` field matches one of the specified values. If omitted, records are included regardless of their `disposition` field.

e.g. if the parameter name is `GE_eventTime` and the parameter value is 2007-02-19T20:29:00.000-0:00 then only those records are returned where their `eventTime` is equal to 19th February, 2007, 20:29 GMT or after this time.

The following query constraint parameters can be used to sort the links into chronological order and also to limit the number of records to be returned:

Constraint Name	*Constraint Value Type*	*Required*	*Meaning*
`orderBy`	String	No	If specified, names a single field that will be used to order the results. The `orderDirection` field specifies whether the ordering is in ascending sequence or descending sequence. Records included in the result that lack the specified field altogether may occur in any position within the result record list. The value of this parameter SHALL be one of: `eventTime` or `recordTime`. If omitted, no order is specified. The implementation MAY order the results in any order it chooses, and that order MAY differ even when the same query is executed twice on the same data.
`orderDirection`	String	No	If specified and `orderBy` is also specified, specifies whether the results are ordered in ascending or descending sequence according to the key specified by `orderBy`. The value of this parameter must one of `ASC` (for ascending order) or `DESC` (for descending order). If omitted, defaults to `DESC`.
`recordCountLimit`	Int	No	If specified, the results will only include the first N records that match the other criteria, where N is the value of this parameter. The ordering specified by the `orderBy` and `orderDirection` parameters determine the meaning of "first" for this purpose. If omitted, all events matching the specified criteria will be included in the results.

4.1.1 Effect of specifying multiple constraints

Note that where multiple constraint parameters are specified in a query to a DS, there is a logical AND between them – i.e. all specified constraints must be satisfied (except those that are ignored because either a particular Discovery Service does not support those fields – or a publisher chose not to supply those fields).

Note also that `EQ_action`, `EQ_bizStep` and `EQ_disposition` may each take a list of alternative values and the constraint is then satisfied if the value of the corresponding field in the DS record matches any one of the values in the list of alternative values for that field. i.e. there is a logical OR within a given constraint parameter, if multiple alternative values are specified.

Discovery Services may support a REST query interface – or they may accept a query specified via an XML message formatted as in the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_query>
    <MATCH_anyEPC>
        <epc>urn:epc:id:sgtin:0614141.107340.1</epc>
    </MATCH_anyEPC>
</ds_query>
```

5 Response from a Discovery Service

An example of an unsigned response from a Discovery Service is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_queryResponse>
    <epcElement>
        <epc>urn:epc:id:sgtin:0614141.107340.1</epc>
        <nodelist>
            <node>
                <noderef>586822F278C1C92DD40F53298FD2602582D2FA72</noderef>
                <status>OK-Verified</status>
                <serviceType>EPCIS</serviceType>
                <serviceAddress>http://www.factorycorp.com/gateway/epcis.wsdl</serviceAddress>
                <publisherID>
                    <GLN>0614141000007</GLN>
                </publisherID>
            </node>
            <node>
                <noderef>758EC127AD9A6A6FB459EBE0210AB7C380062A7B</noderef>
                <status>Access Denied</status>
            </node>
        </nodelist>
    </epcElement>
    <epcElement>
        <epc>urn:epc:id:sgtin:0614141.107340.2</epc>
        <nodelist>
            <node>
                <noderef>586822F278C1C92DD40F53298FD2602582D2FA87</noderef>
                <status>OK-Verified</status>
                <serviceType>EPCIS</serviceType>
                <serviceAddress>http://www.factorycorp.com/gateway/epcis.wsdl</serviceAddress>
                <publisherID>
                    <GLN>0614141000007</GLN>
                </publisherID>
            </node>
            <node noderef="758EC127AD9A6A6FB459EBE0210AB7C380062A45">
                <status>Access Denied</status>
            </node>
        </nodelist>
    </epcElement>
</ds_queryResponse>
```

The response indicates that the Discovery Service knows of two nodes for the specified EPC but only the first node reveals its identity and provides any address information. It indicates that is an EPCIS interface – and the status indicates that the Discovery Service was able to verify the authenticity of the record from the publisher via a digital signature. The publisher ID, in this case a GLN (and perhaps other information) is included.
For the second node, access is denied. However, the noderef is an opaque handle which indicates to the client that it has not received a complete set of links – and (if the Discovery Service supports this), the client may submit further information (or more comprehensive / up-to-date credentials) to the operator of the service for the second node, using the noderef as a reference number, to refer to the node that the client wishes to contact for further information. Upon receipt of such a request, the Discovery Service may forward the request to the operator of the node, perhaps as an e-mail message to a human administrator.
Note that this is a potentially useful value-added feature, but which requires additional security measures to prevent it from being abused (e.g. generating spam messages to annoy administrators etc.).

Note also that the response from a Discovery Service may provide a set of links for a number of EPCs, especially if a list of EPCs is provided as input to the query parameters MATCH_anyEPC, MATCH_parentID or MATCH_childEPC. The element `<epcElement>` is used as a container for the links provided for each EPC, to avoid any ambiguity about which links correspond to which EPC.

Below is an example of a query response that includes aggregation records

```
<?xml version="1.0" encoding="UTF-8"?>
<ds_response>
    <epcElement>
        <epc>urn:epc:id:sgtin:0614141.107340.4</epc>
        <nodelist>
            <node>
                <noderef>586822F278C1C92DD40F53298FD2602582D2FA87</noderef>
                <status>OK-Verified</status>
                <serviceType>EPCIS</serviceType>
                <serviceAddress>http://www.factorycorp.com/gateway/epcis.wsdl</serviceAddress>
                <publisherID>
                    <GLN>0614141000007</GLN>
                </publisherID>
            </node>
            <node>
                <noderef>3758EC127AD9A6A6FB459EBE0210B7C380062A45</noderef>
                <status>OK-Verified</status>
                <serviceType>EPCIS</serviceType>
                <serviceAddress>http://www.distcorp.com/partners/epcis.wsdl</serviceAddress>
                <publisherID>
                    <GLN>0537289000003</GLN>
                </publisherID>
                <aggregationInfo>
                    <action>ADD</action>
                    <parentID>urn:epc:id:sgtin:0614141.107340.1</parentID>
                    <childEPCs>
                       <epc>urn:epc:id:sgtin:0614141.107340.4</epc>
                    </childEPCs>
                </aggregationInfo>
            </node>
        </nodelist>
    </epcElement>
</ds_response>
```

In this example, the information from the second node is that the EPC has been aggregated within a parent object, whose ID is specified.

6 Interface methods

This section summarizes the methods available for interacting with a Discovery Service. These are divided into Publisher methods (which a company uses to publish records to a Discovery Service) and Query methods (used by a client for retrieving data from a Discovery Service).

6.1 Methods available to both publishers and clients (query interface)

The methods below are for information purposes and are generally available to clients and publishers. The corresponding XML schema (XSD) are defined in Section 10.11.

`getStandardVersion()`	identifies which version of a (future) Discovery Services standard is implemented
`getVendorVersion()`	identifies a vendor-specific number of the version and revision number for this implementation
`getSupportedOptionalFields()`	provides a list of meta-data fields supported by the data model of this implementation.
`getSupportsAggregation()`	returns true if the publishing and querying of aggregation records is supported within this implementation of Discovery Services; returns false if aggregation records are not supported.
`getSupportedMessaging()`	provides a list of message transport services supported by this implementation of Discovery Services – indicates allowed options to help a client specify the `dest` URI parameter for a standing query (see section 8.1)
`getCurrentTime()`	returns the current internal timestamp of the specific Discovery Service. The timestamp should be expressed with a resolution of 1 second and be timezone qualified, relative to UTC. i.e. YYYY-MM-DDThh:mm:ssTZD (e.g. “1997-07-16T19:20:30+01:00”)

6.2 Publisher methods

This section should be read in conjunction with section 4, which explains how the publisher first registers a publisher profile, then publishes a number of records referring to that profile. Sections 10.1-10.6 define corresponding XML schema (XSD).

`registerProfile(record)`	publishes a new publisher profile to a Discovery Service. The profile is an XML message conforming to the schema `RegisterProfile.xsd`
`updateProfile(record)`	informs the Discovery Service of a change to an existing publisher profile. The profile update is an XML message conforming to the schema `UpdateProfile.xsd`
`publish(record)`	publishes a new record to a Discovery Service. The record is an XML message conforming to the schema `PublishRecord.xsd`

6.3 Query methods

This section should be read in conjunction with section 5, which explains how the query mechanism uses parameters to select which link information is retrieved.

`poll(querySpecification)`	retrieves link information from a Discovery Service subject to the constraint criteria specified in the list `querySpecification`. The querySpecification takes the form of pairs of a constraint parameter name and a value (which may itself be a list of alternative values, any of which represent a match). The constraint `MATCH_anyEPC` is mandatory and must always be included.

Section 10.8 defines XML schema (XSD) for sending a query to a Discovery Service.

The response from our Discovery Service shall conform to the schema `QueryResponse.xsd` defined in section 10.9

7 Support for standing queries

It is important for event-driven service-oriented-architectures to support standing queries, in order to allow a client to register their interests in particular query criteria and be automatically and promptly notified with any future updates matching those criteria, which are subsequently received by the service at any time in the future. For example, a manufacturer may wish to register a standing query with a Discovery Service, specifying as query criteria a list of the EPCs shipped by the manufacturer, in order to be automatically notified of their progress across the supply chain, as downstream organizations publish records to the Discovery Service, usually to acknowledge that they have handled those objects.

Both the Application Level Events (ALE) and EPC Information Services (EPCIS) standard interfaces support registration of standing queries, with some subtle differences between them. Application Level Events 1.0 allows the boundary conditions (i.e. start/stop conditions or triggers) of an Event Cycle specification to be set, to determine the collection period for the reporting of ALE events – these may be triggered by external events or specified as being periodic in time. EPC Information Services 1.0 takes a slightly different approach and allows for a time-based subscription control, following a similar approach and syntax to the scheduling of processes on POSIX operating systems via `cron` and `crontab`.

The latter approach, of time-based subscription controls would appear to be most appropriate for Discovery Services. This provides the client with flexibility to suggest a schedule for when new records should be delivered from the Discovery Service to the client, either batched to be received at particular times of the day – or if the client specifies the trigger URI `'urn:bridge-project.eu:ds:triggers:onPublication'`, this should be interpreted as a request for new records to be sent to the client preferably without any delay (i.e. no time-based batching of responses).

7.1 Methods for supporting standing queries

The subscription models specified in ALE and EPCIS both support the methods `subscribe()`, `unsubscribe()`, and `poll()`. The explanation below indicates how these methods may be applied to Discovery Services to support standing queries.

`poll(querySpec: querySpecification) : queryResults`

The poll method is used to perform an immediate one-off query of a Discovery Service. The query specification is formatted according to `Query.xsd` defined in Section 10.8 and the results are usually returned synchronously using the same message transport and formatted according to `QueryResponse.xsd` defined in Section 10.9.

`subscribe(querySpecification, dest: URI, controls: SubscriptionControls, subscriptionID: string): boolean success`

The subscribe method is used to create a subscription to a standing query. Like the poll method, the query is specified according to the format `Query.xsd` defined in Section 10.8.

The second parameter, `dest` is a URI that indicates a messaging protocol and address to which the results should be sent. For example, a URI that begins jms: indicates that Java Message Service (JMS) is to be used.

The third parameter, `controls` is used to specify the time schedule for when results should be batched together and sent to the messaging service. The format, `SubscriptionControls` is defined below – and aligned with the crontab-like specification used in EPCIS v1.0

The fourth parameter, `subscriptionID` is a string that is used to unambiguously refer to an individual subscription created by a particular client. i.e. the `subscriptionID`

combined with the client ID should be unique for each subscription running within a particular Discovery Service implementation.
A return value of true indicates that a subscription has successfully been created. A return value of false indicates that creation of a subscription was unsuccessful, most likely because an existing `subscriptionID` was supplied.

`unsubscribe(subscriptionID: String) : boolean success`

This method allows a client to cancel a subscription that is no longer of interest to the client.
A return value of true indicates that a subscription has successfully been cancelled. A return value of false indicates that cancellation of a subscription was unsuccessful, most likely because an invalid `subscriptionID` was supplied.

`getSubscriptionIDs() : list of String`

This method returns to the client a list of all the subscriptionID values for the standing queries subscribed to by the client

`getSubscriptionByID(subscriptionID: String)`

This method returns to the client a message that contains the parameters originally supplied for a particular `subscriptionID` via the `subscribe()` command. The format of this message is defined in section 10.10 This method is only provided for convenience, in case the client failed to store these details in a local cache when using the `subscribe()` method.

7.2 Subscription controls

Subscription controls can be specified to return responses to standing queries either on a periodic basis or based upon particular trigger conditions.

The fields for a Subscription Control instance are defined below – and XML schema are provided in section 10.10

Argument	Type	Description
`schedule`	`QuerySchedule`	(Optional) Defines a periodc schedule on which the query is to be executed. Exactly one argument of either schedule or trigger must be specified – or else a `SubscriptionControlsException` should be raised.
`trigger`	`URI`	(Optional) Specifies a triggering event known to a Discovery Service, which serves to trigger execution of the standing query. Exactly one argument of either schedule or trigger must be specified – or else a `SubscriptionControlsException` should be raised.
`initialRecordTime`	`Time`	(Optional) Specifies a time used to constrain which records are considered when processing the standing query when it is executed for the first time. If omitted, the time defaults to the time at which the subscription was created.
`reportIfEmpty`	`boolean`	If true, a response is always sent to the subscriber when the query is executed. If false, a response is only sent to the subscriber if it contains a non-empty set of addresses.

7.3 Special value of trigger URI

The trigger URI `'urn:bridge-project.eu:ds:triggers:onPublication'`, this should be interpreted as a request for new records to be sent to the client preferably without any delay (i.e. no time-based batching of responses).

7.4 Schedule

The schedule is specified five optional values, which specify the minute, hour, day of week, month and day of month when a standing query should be executed, although implementations of Discovery Services are free to execute the query within a narrow time range similar to the value specified, if this enables better load balancing.

The five numeric values in the list are specified as follows:

Argument	Type	Description
`minute`	`String`	(Optional) Specifies that the query time must have a matching minute value. The range for this parameter is 0 through 59, inclusive.
`hour`	`String`	(Optional) Specifies that the query time must have a matching hour value. The range for this parameter is 0 through 23, inclusive, with 0 denoting the hour that begins at midnight, and 23 denoting the hour that ends at midnight.
`dayOfMonth`	`String`	(Optional) Specifies that the query time must have a matching day of month value. The range for this parameter is 1 through 31, inclusive. (Values of 29, 30, and 31 will only match during months that have at least that many days.)
`month`	`String`	(Optional) Specifies that the query time must have a matching month value. The range for this parameter is 1 through 12, inclusive.
`dayOfWeek`	`String`	(Optional) Specifies that the query time must have a matching day of week value. The range for this parameter is 1 through 7, inclusive, with 1 denoting Monday, 2 denoting Tuesday, and so forth, up to 7 denoting Sunday.

Note that all values are optional. If no values are specified, the schedule shall instead be interpreted as corresponding to the special trigger URI, `'urn:bridge-project.eu:ds:triggers:onPublication'` and responses will be sent as soon as possible after new records are received by a Discovery Service, where they match the query specified.

XML schema for the subscription controls are provided in section 10.10

7.5 Push vs Pull

Standing queries are intended to provide a client subscribers with updated information corresponding to records received by a Discovery Service at a future time, without requiring further interaction with the Discovery Service itself. Depending on the choice of messaging service specified via the `dest` URI parameter, the updates may be pushed via some messaging protocols to a 'listener' that expects to receive them. For other messaging services, the updates are pushed into a dedicated queue for a particular client and the client may need to periodically poll the message queue to retrieve any updates, much like a POP e-mail client periodically polls a remote POP mailbox – although this is clearly a different interaction mode than polling of a Discovery Service, since the polling is of the underlying message transport layer.

8 Access Controls

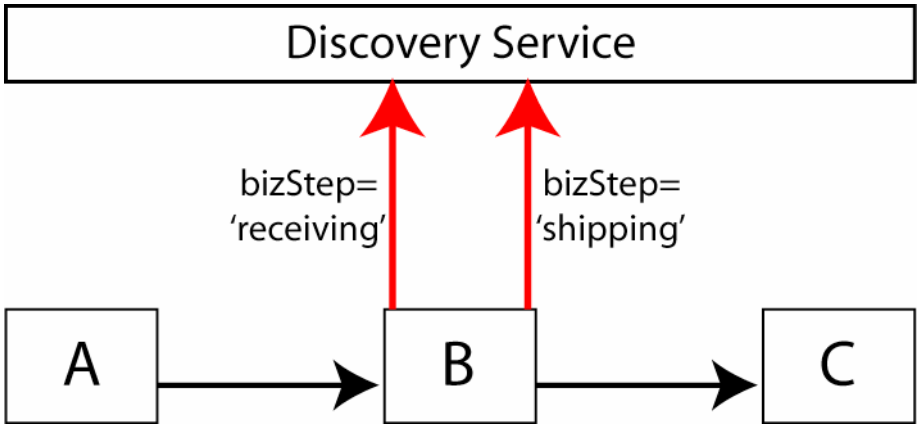
Access control policies may be specified in order to grant or restrict visibility of that record to particular clients, groups of clients or roles.

Access control policies may be specified by both the operator of a Discovery Service and by the publisher of a Discovery Service record. The Discovery Service policy may specify default security arrangements that will apply to all publishers and cannot be over-ridden by the publisher. For example, such Discovery Service policies may specify that a regulatory body has read access for all records relating to particular products.

Access controls relate to the reading of DS records, along with the writing, updating and deletion of such records. Access controls may also cover the definition of security policies to allow delegation of access right management.

The access control decision may be made over any information submitted in the client request. This includes the body of the request (specifying for example the EPC range or attribute filters such as bizStep), along with header information carrying the client credentials. Such credentials will include the authenticated identity of the client along with assertions such as roles or groups to which the client belongs.

Consider as an example, the simple supply chain below:



publisher B may decide to grant read access to A for records with bizStep = 'receiving' and may grant read access to C for records with bizStep = 'shipping'.

In this example, publisher B published two separate records to the Discovery Service, in order to keep their upstream supply chain separate from their downstream supply chain.

However, client A may be able to read records published by C (or vice versa), depending on the access controls that they set for the records that they publish.

One of the challenges in developing access control policies is scalability. Supposing that there are N organizations within a particular supply chain, there could theoretically be N x (N-1) distinct policies, although the number of Discovery Service records scales approximately linearly with N.

In reality, many organizations may have very little visibility across the entire supply chain – and may only have visibility of one company upstream and one company downstream. In this situation, a more scaleable approach to the number of access control policies can be achieved if the policies can also express whether the rights granted to the supplier or customer are also sharable with other parties further upstream / downstream, as appropriate. In this way,

propagating access control policies between companies that are adjacent to each other within the supply chain can potentially be combined together logically to evaluate the appropriate policy between any two organizations within the supply chain. The EPCglobal Architecture Review Committee has recently outlined such an approach.

Access control policies for Discovery Services will be developed more fully by BRIDGE WP4, task 4.5.2 – and their implementation described in task 4.5.3.

In order to ensure scalability of access control policies, a single policy may be applied to multiple records. In addition to the roles of client and publisher, a separate security manager role will be defined, together with a policy management interface, allowing the security manager to specify the policies to be applied both to the records published by that organization and to queries made by clients within that organization, in order to specify a confidentiality policy, to prevent their query from being divulged to unintended parties.

9 XML Schema

This section provides XML schema that define the structure of the messages and responses for interaction with a Discovery Service.

9.1 Registering a publisher profile

RegisterProfile.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

   <xsd:include schemaLocation="./ServiceTypeIDList.xsd"/>

   <xsd:complexType name="RegisterProfile">
      <xsd:sequence>
         <xsd:element name="serviceType" type="ds:ServiceTypeIDList" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="serviceAddress" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_registerProfile" type="ds:RegisterProfile"/>
</xsd:schema>
```

9.2 Response to registering a profile

RegisteredProfileResponse.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

   <xsd:include schemaLocation="./ServiceTypeIDList.xsd"/>

   <xsd:complexType name="RegisteredProfileResponse">
      <xsd:sequence>
         <xsd:element name="serviceType" type="ds:ServiceTypeIDList" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="serviceAddress" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="publisherProfileID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_registeredProfileResponse" type="ds:RegisteredProfileResponse"/>
</xsd:schema>
```

9.3 Updating a publisher profile

UpdateProfile.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

   <xsd:include schemaLocation="./ServiceTypeIDList.xsd"/>

   <xsd:complexType name="Profile">
      <xsd:sequence>
         <xsd:element name="serviceType" type="ds:ServiceTypeIDList" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="serviceAddress" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="UpdateProfile">
      <xsd:sequence>
         <xsd:element name="old" type="ds:Profile" minOccurs="1" maxOccurs="1">
         <xsd:element name="new" type="ds:Profile" minOccurs="1" maxOccurs="1">
         <xsd:element name="publisherProfileID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_updateProfile" type="ds:UpdateProfile"/>
</xsd:schema>
```

9.4 Response to updating a publisher profile

UpdatedProfileResponse.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

   <xsd:include schemaLocation="./ServiceTypeIDList.xsd"/>

   <xsd:complexType name="UpdatedProfileResponse">
      <xsd:sequence>
         <xsd:element name="serviceType" type="ds:ServiceTypeIDList" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="serviceAddress" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="publisherProfileID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_updatedProfileResponse" type="ds:UpdatedProfileResponse"/>
</xsd:schema>
```

9.5 Publishing basic records or aggregation records

PublishRecord.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

    <xsd:include schemaLocation="./DSActionList.xsd"/>

    <xsd:complexType name="EPCList">
        <xsd:sequence>
            <xsd:element name="epc" type="xsd:anyURI" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Record">
        <xsd:sequence>
            <xsd:element name="publisherProfileID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="epcList" type="ds:EPCList" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="action" type="ds:DSActionList" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="bizStep" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="disposition" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="eventTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="AggregationRecord">
        <xsd:sequence>
            <xsd:element name="publisherProfileID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="parentID" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="childEPCs" type="ds:EPCList" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="action" type="ds:DSActionList" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="bizStep" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="disposition" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="eventTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Header">
        <xsd:sequence>
            <xsd:element name="TTL" type="xsd:duration" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="ACP" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="PublishRecord">
        <xsd:sequence>
            <xsd:element name="ds_header" type="ds:Header" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="ds_record" type="ds:Record" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="ds_aggregationRecord" type="ds:AggregationRecord" minOccurs="0"
                  maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="ds_publishRecord" type="ds:PublishRecord"/>
</xsd:schema>
```

9.6 Response to publishing of records

PublishedRecordResponse.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:complexType name="PublishRecord">
      <xsd:sequence>
         <xsd:element name="recordID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_publishedRecordResponse" type="ds:PublishRecord"/>

</xsd:schema>
```

9.7 Voiding of published records

VoidRecord.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

   <xsd:complexType name="VoidRecord">
      <xsd:sequence>
         <xsd:element name="recordID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="reason" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_voidRecord" type="ds:VoidRecord"/>
</xsd:schema>
```

VoidedRecordResponse.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

   <xsd:complexType name="VoidedRecordResponse">
      <xsd:sequence>
         <xsd:element name="recordID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="voidRecordException" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_voidedRecordResponse" type="ds:VoidedRecordResponse"/>
</xsd:schema>
```

9.8 Specifying a query

Query.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:include schemaLocation="./DSAnyActionList.xsd"/>
   <xsd:include schemaLocation="./RecordTypeList.xsd"/>
   <xsd:include schemaLocation="./DSOrderByList.xsd"/>
   <xsd:include schemaLocation="./DSOrderDirList.xsd"/>

   <xsd:complexType name="EPCList">
      <xsd:sequence>
         <xsd:element name="epc" type="xsd:anyURI" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="Query">
      <xsd:sequence>
         <xsd:element name="MATCH_anyEPC" type="ds:EPCList" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="MATCH_parentID" type="ds:EPCList" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="MATCH_childEPCs" type="ds:EPCList" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="recordType" type="ds:RecordTypeList" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="GE_eventTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="LT_eventTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="GE_recordTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="LT_recordTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="EQ_action" type="ds:AnyActionList" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="EQ_bizStep" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="EQ_disposition" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="orderBy" type="ds:OrderByList" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="orderDirection" type="ds:OrderDirList" minOccurs="0" maxOccurs="1"/>
         <xsd:element name="recordCountLimit" type="xsd:integer" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="ds_query" type="ds:Query"/>
</xsd:schema>
```

9.9 Response to a query

QueryResponse.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

    <xsd:include schemaLocation="./DSActionList.xsd"/>
    <xsd:include schemaLocation="./StatusIDList.xsd"/>
    <xsd:include schemaLocation="./ServiceTypeIDList.xsd"/>

    <xsd:complexType name="EPCList">
        <xsd:sequence>
            <xsd:element name="epc" type="xsd:anyURI" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="AggregationInfoType">
        <xsd:sequence>
            <xsd:element name="parentID" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="childEPCs" type="ds:EPCList" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="action" type="ds:DSActionList" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="PublisherType">
        <xsd:sequence>
            <xsd:element name="GLN" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="NodeType">
        <xsd:sequence>
            <xsd:element name="noderef" type="xsd:string" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="status" type="ds:StatusIDList" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="serviceType" type="ds:ServiceTypeIDList" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="serviceAddress" type="xsd:anyURI" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="aggregationInfo" type="ds:AggregationInfoType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="publisher" type="ds:PublisherType" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="NodeListType">
        <xsd:sequence>
            <xsd:element name="node" type="ds:NodeType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="epcElement">
        <xsd:sequence>
            <xsd:element name="epc" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="nodelist" type="ds:NodeListType" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="QueryResponse">
        <xsd:sequence>
            <xsd:element name="responseCode" type="ds:ResponseCodeID" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="epcElement" type="ds:epcElement" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="ds_response" type="ds:QueryResponse"/>
</xsd:schema>
```

9.10 Schema for standing queries

StandingQueryMethods.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

  <xsd:include schemaLocation="./DSAnyActionList.xsd"/>
  <xsd:include schemaLocation="./RecordTypeList.xsd"/>
  <xsd:include schemaLocation="./DSOrderByList.xsd"/>
  <xsd:include schemaLocation="./DSOrderDirList.xsd"/>
  <xsd:include schemaLocation="./Query.xsd"/>
  <xsd:include schemaLocation="./QueryResponse.xsd"/>

  <xsd:complexType name="SubscriptionControls">
     <xsd:sequence>
      <xsd:element name="schedule" type="ds:QuerySchedule" minOccurs="0"/>
      <xsd:element name="trigger" type="xsd:anyURI" minOccurs="0"/>
      <xsd:element name="initialRecordTime" type="xsd:dateTime" minOccurs="0"/>
      <xsd:element name="reportIfEmpty" type="xsd:boolean"/>
      <xsd:element name="extension" type="ds:SubscriptionControlsExtensionType" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="SubscriptionControlsExtensionType">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <xsd:complexType name="QuerySchedule">
    <xsd:sequence>
       <xsd:element name="second" type="xsd:string" minOccurs="0"/>
       <xsd:element name="minute" type="xsd:string" minOccurs="0"/>
       <xsd:element name="hour" type="xsd:string" minOccurs="0"/>
       <xsd:element name="dayOfMonth" type="xsd:string" minOccurs="0"/>
       <xsd:element name="month" type="xsd:string" minOccurs="0"/>
       <xsd:element name="dayOfWeek" type="xsd:string" minOccurs="0"/>
      <xsd:element name="extension" type="ds:QueryScheduleExtensionType" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="QueryScheduleExtensionType">
    <xsd:sequence>
      <xsd:any namespace="##local" processContents="lax" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

   <xsd:complexType name="Subscribe">
       <xsd:sequence>
        <xsd:element name="querySpecification" type="ds:Query"/>
        <xsd:element name="dest" type="xsd:anyURI"/>
        <xsd:element name="controls" type="ds:SubscriptionControls"/>
        <xsd:element name="subscriptionID" type="xsd:string"/>
     </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="subscribe" type="ds:Subscribe"/>

  <xsd:element name="SubscribeResponse" type="xsd:boolean"/>

  <xsd:complexType name="Unsubscribe">
     <xsd:sequence>
        <xsd:element name="subscriptionID" type="xsd:string"/>
     </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="unsubscribe" type="ds:Unsubscribe"/>

  <xsd:element name="UnsubscribeResponse" type="xsd:boolean"/>

 <xsd:element name="getSubscriptionIDs" type="ds:emptyParams"/>

 <xsd:element name="GetSubscriptionIDsResponse" type="ds:ArrayOfString"/>


  <xsd:complexType name="GetSubscriptionByID">
       <xsd:sequence>
        <xsd:element name="subscriptionID" type="xsd:string"/>
     </xsd:sequence>
  </xsd:complexType>
```

```
 <xsd:element name="getSubscriptionByID" type="ds:GetSubscriptionByID" />

  <xsd:complexType name="GetSubscriptionByIDResponse">
       <xsd:sequence>
        <xsd:element name="querySpecification" type="ds:Query"/>
        <xsd:element name="dest" type="xsd:anyURI"/>
        <xsd:element name="controls" type="ds:SubscriptionControls"/>
        <xsd:element name="subscriptionID" type="xsd:string"/>
      </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="GetSubscriptionByIDResponse" type="ds:GetSubscriptionByIDResponse" />

  <xsd:complexType name="Poll">
      <xsd:sequence>
         <xsd:element name="querySpecification" type="ds:Query"/>
      </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="poll" type="ds:Poll"/>

  <xsd:element name="pollQueryResponse" type="ds:QueryResponse"/>

</xsd:schema>
```

9.11 Schema for general interface methods

GeneralMethods.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ds="urn:bridge-project.eu:ds:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="0.1">

    <xsd:complexType name="ds:EmptyParams"/>

    <xsd:complexType name="ArrayOfString">
        <xsd:sequence>
           <xsd:element name="string" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="ds_getStandardVersion" type="ds:EmptyParams"/>

    <xsd:element name="ds_standardVersionResponse" type="xsd:string"/>

    <xsd:element name="ds_getVendorVersion" type="ds:EmptyParams"/>

    <xsd:element name="ds_vendorVersionResponse" type="xsd:string"/>

    <xsd:element name="ds_getSupportedOptionalFields” type="ds:EmptyParams"/>

    <xsd:element name="ds_supportedOptionalFieldsResponse" type="ds:ArrayOfString"/>

    <xsd:element name="ds_getSupportsAggregation” type="ds:EmptyParams"/>

    <xsd:element name="ds_supportedOptionalFieldsResponse" type="xsd:boolean"/>

    <xsd:element name="ds_getSupportedMessaging” type="ds:EmptyParams"/>

    <xsd:element name="ds_supportedMessagingResponse" type="ds:ArrayOfString"/>

    <xsd:element name="ds_getCurrentTime” type="ds:EmptyParams"/>

    <xsd:element name="ds_getCurrentTimeResponse" type="xsd:dateTime"/>

</xsd:schema>
```

9.12 Auxiliary schema for constrained enumerated lists

DSActionList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="DSActionList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="CREATE"/>
         <xsd:enumeration value="LINK"/>
         <xsd:enumeration value="CLOSE"/>
         <xsd:enumeration value="DESTROY"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

DSAggActionList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="DSAggActionList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="ADD"/>
         <xsd:enumeration value="OBSERVE"/>
         <xsd:enumeration value="DELETE"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

DSAnyActionList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="DSAnyActionList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="CREATE"/>
         <xsd:enumeration value="LINK"/>
         <xsd:enumeration value="CLOSE"/>
         <xsd:enumeration value="DESTROY"/>
         <xsd:enumeration value="ADD"/>
         <xsd:enumeration value="OBSERVE"/>
         <xsd:enumeration value="DELETE"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

DSOrderByList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="DSOrderByList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="eventTime"/>
         <xsd:enumeration value="recordTime"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

DSOrderDirList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="DSOrderDirList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="ASC"/>
         <xsd:enumeration value="DESC"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

RecordTypeList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="RecordTypeList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="DS_Record"/>
         <xsd:enumeration value="DS_AggregationRecord"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

ResponseCodeList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="ResponseCodeList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="OK"/>
         <xsd:enumeration value="Bad Request"/>
         <xsd:enumeration value="Unauthorized"/>
         <xsd:enumeration value="Forbidden"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

ServiceTypeIDList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="ServiceTypeIDList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="EPCIS"/>
         <xsd:enumeration value="DS"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

StatusIDList.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated from annotated java  -->
<xsd:schema targetNamespace="urn:bridge-project.eu:ds:xsd:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ds="urn:bridge-project.eu:ds:xsd:1"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="0.1">

   <xsd:simpleType name="StatusIDList">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="OK-Verified"/>
         <xsd:enumeration value="OK-Unverified"/>
         <xsd:enumeration value="Access Denied"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

10 Glossary of Terms

Client	An organization making a query to an EPCIS or Discovery Service
Custodian	An organization that physically handles an individual object at some time during its supply chain or lifecycle.
Discovery Service	A mechanism (not yet standardized) for finding providers of information about an object, if the object's EPC or unique identifier is known
EPC – abbreviation of 'Electronic Product Code'	A framework for globally unique identifiers for use with Auto-ID technologies. Use of a globally unique identifier allows each object to be tracked individually and for each organization to store information about each individual object
EPCIS – EPC Information Services	A standard interface for accessing detailed serial-level information about an object (usually retrieved from within a single organization)
Historical Trace	A historical (or upstream) trace attempts to find all previous information providers.
Information Provider / Source / Node	An organization that holds some detailed information about an individual object. This potentially includes custodians (who handle the physical object), as well as non-custodians, such as insurance companies, who nevertheless hold individual records for objects, such as warranty details.
Publisher	An organization that publishes a new record to a Discovery Service
(Discovery Services) Record	A data packet, that contains a number of data fields, and is used to indicate a relationship between
Track	The act of finding the current (or last recorded) information provider for an object
Trace	The act of finding several information providers for an object.

11 References

EPCglobal Network Architecture Framework document
See http://www.epcglobalinc.org/standards

EPC Information Services (EPCIS) v1.0
See http://www.epcglobalinc.org/standards

EPCglobal Tag Data Standards v1.3
See http://www.epcglobalinc.org/standards

Application Level Events (ALE) v1.0
See http://www.epcglobalinc.org/standards

Object Name Service (ONS) v1.0
See http://www.epcglobalinc.org/standards

HTTP 1.1 (including status codes)
http://www.ietf.org/rfc/rfc2616.txt

REST – REpresentation State Transfer
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

XSD XML Schema
http://www.w3.org/XML/Schema

Time and data formats
http://www.w3.org/TR/NOTE-datetime

Building **R**adio frequency **ID**entification for the **G**lobal **E**nvironment

High level design for Discovery Services

Section B: Analysis of Discovery Service Models for RFID

Authors: University of Cambridge, AT4 wireless, BT Research, SAP Research

15 August 2007

This work has been partly funded by the European Commission contract No: IST-2005-033546

Revision History

Version	Date	Author	Summary of Changes
	January 25, 2007	Trevor Burbridge (BT)	Initial document: Security Considerations of Alternative Discovery Service Designs
	May 29, 2007	Trevor Burbridge (BT)	Included section on taxonomy of DS models and preliminary evaluation
	June 7, 2007	Trevor Burbridge (BT)	Revised version
	June, 2007	Mark Harrison (Cambridge)	Additions to all sections
	June 29, 2007	Oliver Kasten (SAP)	Additions and improvements to all sections. Integration into BRIDGE deliverable template.
	July 2, 2007	Cosmin Condea (SAP)	Modifications to Section 2, format changes
	July 4, 2007	Oliver Kasten (SAP)	Added Sect. 1, Introduction.
	July 4, 2007	Mark Harrison (Cambridge)	Added intro to one-off vs. standing queries
	July 5, 2007	Trevor Burbridge (BT)	Improvements to sections on discovery from unknown parties and on resource confidentiality
	July 6, 2007	Oliver Kasten (SAP)	Improvements to all sections, final editing.
	July 6, 2007	Mark Harrison (Cambridge)	Proof-reading of document
	July 26, 2007	Nicholas Pauvre (GS1 France)	Bridge Internal Review
	August 2, 2007	AT4 wireless	Inclusion of comments from internal review

Note

The views expressed in this document are the views of the joint authors and the *Community* is not liable for any use that may be made of the information contained herein.

TABLE OF CONTENTS

1. Introduction

In this section we will briefly repeat our basic assumptions and requirements for completeness of this document and for the reader's convenience. For an introduction to the components of the EPC Network that are relevant in this context and to basic Discovery Service concepts, please refer to D2.4, Section A, High-level design.

1.1. Assumptions

We assume for the purposes of this discussion that both the client and resource trust the Discovery Service with which they are interacting. They expect that the Discovery Service will release information in accordance with its public design and not act in any other interest. Our discussion is based on the following assumptions:

1.1.1. Connectivity, Availability

- EPCIS instances are generally connected to the Internet and are generally reliable.
- EPCIS instances may have downtimes (typically in the order of hours, e.g., for maintenance)
- The volume of client queries (to an EPCIS) is expected to be an order of magnitude lower compared to the volume of event updates (to an EPCIS). Event updates include the creation of new EPCs as well as read events of the same EPC in several locations and organizations
- The address of an EPCIS may change infrequently (e.g., domain name change after company being bought, restructuring of a company's IT infrastructure)

1.1.2. Trust and Confidentiality

- The provider of a Discovery Service is expected to be trustworthy and to act in the interest of resources (i.e., EPCIS instances).

1.2. Requirements

Below is a list of the most important requirements. For the full list of requirements please refer to D2.1, Section C.

- Client Queries must be treated confidentially by the Discovery Service
- Discovery Service records (typically, EPC number and resource references) must be treated confidentially by the discovery service.
- Latency times should be minimized
- The Query Response must be complete, that is, it must contain all answers by resources that have willingly chosen to provide an answer.

2. RFID and Discovery Services

A Discovery Service provides a method for the establishment of contact between a client and a resource. Discovery Services appear in some form in any Service Oriented Architecture but vary in the exact requirements they are designed to meet. While there are a number of different approaches to establish communication between a client and a resource, we shall outline some of the main ones and discuss their suitability for RFID.

2.1. Background and Scope

The RFID architecture we consider and which we try to find a suitable Discovery Service for is the EPCglobal Network [12]. Its central purpose is to enable inter-organizational sharing of information about individually identifiable objects. In the EPCglobal Network, the Electronic Product Code (EPC) serves the role of a globally unique ID for objects, The client is an application or service that requires data associated with an EPC-identifiable object whereas the resources are the repositories that contain item-level data (which is typically but not necessarily collected using RFID technology). In the EPCglobal Network, these repositories are the so-called EPC Information Services (EPCIS). Discovery Services that we discuss throughout this document focus on detection of item-level information services that actually share item-level data between multiple organizations. Note that in the EPCglobal Network other kinds of discovery, possibly at different layers in the architecture, could be present – for example, the discovery of RFID readers. Since in our case there is no single owner of the resources and client applications, discovery becomes particularly challenging. In such a situation, the Discovery Service can act as a trusted *intermediary* to establish selected contact between clients and resources.

The Discovery process forms part of a larger communication, whose ultimate aim is to allow resources to serve the needs of clients. The communication between the client, Discovery Service and resource can be considered in three phases:

1) **Setup**. During this phase the client and the resource engage with a Discovery Service to register their interests or capabilities and negotiate security rights.
2) **Discovery**. The discovery phase provides either the client or resource with sufficient information about the other party to initiate the service phase. For the purposes of this document the discovery phase is considered to start when the client attempts to discover resources that have already been publicised or vice-versa.
3) **Service Fulfilment**. During the service fulfilment phase the resources are engaged to meet the ultimate demands of the clients. The Service Fulfilment phase is considered to start when the resource becomes aware of the client request and is able to meet it.

2.2. Queries and Data

Before proceeding with the taxonomy of Discovery Services, let us add more specifics in regards to an RFID architecture. We start from a couple of simplifying assumptions on which, as we will see later, our Discovery Services models will be based on. The first one is that clients can specify either a full query including the EPC number and other parameters or specify only the EPC number. The EPC number represents the query key. The second assumption is that resources can either publish to the Discovery Service solely their EPC numbers they hold or send the full events, that is, business steps, state, etc., pertaining to their EPC-identifiable objects. This implies that the data stored on the Discovery Service / intermediary level can take one of the following four variants:

1) Tuples of the form (EPC number, resource reference). For example, the resource reference may indicate the URL used to the access an EPCIS repository

2) Actual data, fully replicated from all resources / EPCIS instances. This includes the EPC number and resource reference.
3) Keys in the client queries, i.e., the EPC numbers of interest to the client
4) Full client queries.

2.3. Modes of interaction

This section discusses two modes of interaction between a client and information resources, namely one-off queries and standing queries.

2.3.1. One-off queries

In a one-off query, a client wishes to perform a specific query only once and gather current and historical information that is already available at the time the query is issued. The information may be fragmented across multiple resources and the intermediary assists with the gathering of that information, either by forwarding the client's query to multiple relevant resources - or by providing the client with a list of relevant resources, allowing the client to contact each resource in turn.

One-off queries are suitable when the client is not interested in future activity of the specified EPC and does not wish to be informed about new information provided by existing resources, nor about the future arrival of new resources that may provide additional information about the specified EPC.

2.3.2. Standing queries

Standing queries allow a client to register a persistent ongoing interest in a particular EPC, possibly further qualified by additional query parameters. In this situation, a client may wish to be informed about either new information provided by existing resources, as well as about the future arrival of new resources that may provide information about the specified EPC.

For a standing query, the intermediary and/or resource are required to maintain state information about the standing query subscriptions registered by each client. This typically consists of the following information per subscription:

- client ID and callback address (i.e. how and where to send future information),
- query details (in order to send only relevant information),
- timestamp of the last time when an update is sent to the client from the intermediary or resource (note that this timestamp is updated with each successive update that is sent - and should use the same internal clock that is used for recording the recordTime within an EPCIS event or within a Discovery Service record).

Many readers of this document may be familiar with existing resource update mechanisms, such as Really Simple Syndication (RSS) feeds on websites and weblogs, through which a client can quickly be alerted to new content that has recently been added. The EPCglobal standards, EPC Information Services (EPCIS) v1.0 and Application Level Events (ALE) v1.0 already support standing queries / filter requests via a publish-and-subscribe mechanism. However, this document is concerned with mechanisms that allow a client to discover providers of information about an EPC, for which the client may have no prior knowledge of their existence, nor an existing business relationship, in some cases.

This document is therefore less concerned about standing queries subscribed to particular known resources (which is already handled by EPCIS 1.0) - and more concerned with standing queries subscribed to the intermediary, which allow new resources to make themselves known to subscribing clients at a future time when a new resource publishes a record to the intermediary to indicate that they also have information about a particular EPC.

In the discussion of the eight models, we consider their suitability for these two modes of interaction - and in the discussion of two possible implementations, these modes of interaction are considered to be implemented via a hybrid of two distinct models, one model for each mode of interaction.

3. Discovery-Service Taxonomy

In this section we will present a taxonomy for discovery services, resulting in eight basic models. Before discussing these models in more detail, we will present the initial selection criteria assessing the suitability of those models for RFID systems. Based on these criteria we will dismiss some of the models as unsuitable for our needs. The remaining models, and potential implementation options, will be discussed in the next section.

3.1. Taxonomy

We first consider four communication models that have an explicit Discovery phase. In these models, the Service phase is not considered since communication will occur strictly and directly between the client and resource without further engagement of the Discovery Service. Thus, the Service phase has no impact on the Discovery Service design. These four communication models can be categorised according to two criteria. The first criterion is whether the communication during the Discovery Phase is request/reply or publish/subscribe. The second criterion is the direction of communication flow during the Discovery Phase: either the client information flows towards the resource or the resource information is passed towards the clients. It can be noted that request/reply and publish/subscribe communication technologies often employ a mix of both paradigms. For example, in a message queue network the client may connect to the edge message server to request their waiting messages. However, looking at the communication pattern helps to discuss the available options. Finally, the four abovementioned models are:

- **Directory of Resources**. The resources publish their availability in a directory service. For the domain of RFID, this means key-value pairs of EPC numbers and EPCIS addresses. This directory may be a single well-known repository, or consist of a network of federated directory stores with a method of distributing data and routing queries.
- **Directory of Clients.** The clients can register their static interests in a directory. In the case of RFID, this represents the EPC numbers. The resources can then query which clients may be usefully served and initiate interaction with the client.
- **Notification of Resources**. The available resources may multicast their availability onto a communication medium to which the clients listen directly. The communication medium may be a network technology (such as IP multicast), or an overlay network (for example a Discovery Service). As previously mentioned, for RFID, the availability of resources means key-value pairs of EPC numbers and EPCIS addresses.
- **Notification of Clients.** The clients multicast their resource needs onto a communication medium which resources directly listen to. Again, for RFID, these needs are represented by the EPC numbers of interest. Resources able to serve the client respond and establish communication with the clients.

	Request/Response	Publish/Subscribe
Client is querying; Resource is publishing; Client may be unknown	Directory of Resources	Notification-of-Resources
Client is publishing; Resource is querying; Resource may be unknown	Directory of Clients	Notification of Clients

Table 1: Classification of Discovery Service Models. Approaches that include an explicit Discovery Phase.

The above Discovery Service models from Table 1 provide an explicit Discovery phase after which the clients and resources engage directly to fulfil service. Information communicated in

the Discovery Phase is restricted to identifying and locating clients and resources, respectively.

There is an alternative approach where the Discovery phase is omitted. Instead, a communication network, or federated repositories, is provided to route the service between the clients and resources. In effect, the Discovery phase is directly combined with the Service Fulfilment. Again, these models can be classified along the same lines: first after the type of communication, that is, request/reply or publish subscribe and second after the information flow, that is, from client to resource of vice-versa. In the two models where the information flow is Resource to Client, this occurs in response to Client information recorded during the setup phase. No further communication is required to complete Service Fulfilment. In the two models where the communication flow is Client to Resource, a return communication is expected to complete the Service Fulfilment. This return communication may be returned through the Discovery Service, or through another network, such as directly over an IP network. The four models are first briefly described and then presented in Table 2.

- **Meta Resource**. This approach links all resources into a federated system that can be queried at a single point, typically using the same interface specification as the subordinate resources. The full data set of each resource is replicated to the Discovery Service. The single point of access is expected to be known (e.g., through configuration of clients, effectively making the Discovery phase obsolete).
- **Meta Client**. All client requests are available for search which is performed by resources via a single point of access. The Discovery Service stores full client queries (as opposed to query keys, i.e., EPC numbers only)
- **Notification of Events**. The resources publish all available data onto a communication medium. The published data is then routed to the clients. This data is sufficient to meet the clients' demands and thus clients do not need to establish interactive communication with the resources. Resources and clients are effectively decoupled. The routing of events to interested clients can be considered the Discovery phase in this model. This model is used when the resources produce information. It would not make sense for non-information resources such as available processor time in a GRID architecture.
- **Query Propagation**. The clients broadcast their exact requests onto a communication medium. Any resources that are able to meet the client demands respond by providing the required serial-level data. The Discovery Service stores key-value pairs, containing of EPC numbers and references to resources such as EPCIS.

	Request/Response	Publish/Subscribe
Resource to Client	Meta Resource	Notification of Events
Client to Resource	Meta Client	Query Propagation

Table 2: Classification Discovery Service Models. An approach where the Discovery Phase is omitted

The communication diagrams used in the following sections show the Client, Resource and Intermediary (e.g., directory service) communication patterns during the three phases. The numbered arcs show the order of communications. For the first four Discovery Service models, these Discovery-phase communications are sufficient to establish a relationship between the Client and Resource for a subsequent Service Fulfilment phase.

3.2. Initial Selection Criteria

Before describing the eight basic discovery-service models in greater detail, we will briefly discuss the criteria to select a smaller number of models and to dismiss the rest as unsuitable for our needs. The selected models are discussed in greater detail in the next

section. This initial discussion is conducted in two areas: interaction mode and trust in the Discovery Service.

3.2.1. Interaction Mode and Transience of Connectivity

Many of the questionnaire responses and interview results from the Requirements phase of WP2 suggest that quick response times are a key requirement for a Discovery Service. It also follows that predictable response time is required so that client applications can make informed decisions on whether any resources are available at the time of (or soon after) their query.

3.2.2. Data Ownership and Trust

Many respondents to both the WP2 and WP4 surveys suggested that data ownership was a key concern. Most were reluctant to share more than the necessary minimum information with the Discovery Service, or at least suggested that such additional sharing should be optional. Thus we reject any model that requires the EPCIS owner to share detailed information with the Discovery Service without first gaining details of which clients require the detailed access and being able to refuse or negotiate this access.

In the next sections we shall discuss these eight models in greater detail, assessing where the technology is currently used, and discussing their suitability within RFID architectures.

3.3. Directory of Resources

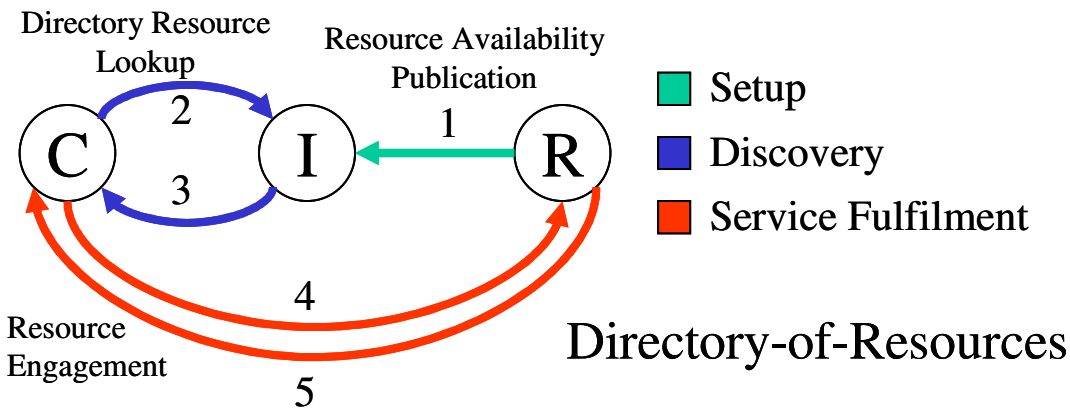


Figure 1: Directory-of-Resources

Figure 1 depicts the Directory of Resources model. The concrete steps are:

1. A resource publishes its availability into the intermediary. In the case of an RFID architecture, this means publishing key-value pairs of EPC numbers and reference resources.
2. A client queries the intermediary to receive the identity of relevant resources. For the domain of RFID, the query contains only the EPC number of interest.
3. The identities (i.e., reference resources) of relevant and legitimately-accessible resources for the EPC number of interest are released to the client.
4. The client separately and directly queries each received resource for detailed information. The full query is now released.
5. A resource answers with detailed information as requested by the client.

Directory Services are extremely common to advertise clients of the availability of resources. For example, Web Services use UDDI (Universal Description, Discovery and Integration) to advertise their availability to client applications and other Web Services. Many middleware platforms for software integration also use directories to publish available services to distributed program components (for example JINI, RMI, CORBA). Directory services are also commonly used for the registration of devices. This approach can be seen in local machine Registry services, or directories for networked 'plug and play' devices.

Many of the directory services already mentioned are often constrained in terms of the network reach and number of parties involved. One well-known global directory service for the registration of device IP addresses and domain names is the Domain Name System (DNS). ENUM, a suite of protocols to unify the telephone numbering system with the internet addressing, provides a similar directory for the registration of telephony numbers. Other global directory services can be seen in content download networks. These networks use a distributed directory to index available content, such as music files.

It can be seen from the above diagram that the Discovery phase should be able to be performed with very low latency for the client. Once resources have been discovered, they may be engaged multiple subsequent times for delivery of service without involving the Discovery Service during each interaction. The client has good visibility over each communication and can take remedial action when individual resources fail to respond during the Service Fulfilment phase. Of course, the access control policies asserted by some resources may prevent a particular client from having any visibility of a particular resource.

The Directory-of-Recourses model allows the client to control the interaction with the directory and subsequent interactions with resources. It can thus control the exact query issued to each of the potentially multiple resource returned by the directory individually and adapt the query if desired.

The client can authenticate the directory before constructing the query, and in some secure directories (such as DNS-Sec) can check the integrity of the directory record. When a client queries a Directory of Resources, the resource is not notified of the identity or existence of the client until the client makes a separate decision to contact that particular resource directly. The client can then apply intelligence to decide which resources it wishes to interact with. For example, a client would have an opportunity to make an innocuous query to each EPCIS resource, such as a request for its current standard version number, then check the response, address of the EPCIS resource and perhaps even verify the signature and signer of a digitally signed response, all before the client is required to divulge to the resource which EPCs are of interest to it. This leads to increased client confidentiality. Of course, a Directory of Resources would need to accept, store and enforce access control policies asserted by the resources, in order to allow the resources to control which records should be revealed to which clients.

There are two principle problems with applying existing Directory Service technology to the domain of RFID. The first problem is that most directories assume that the information is either public, or visible within a controlled user group. For the publication of EPCIS resources this will not be suitable, as data owners will desire more fine-grained controls over who can see the information on the directory. The reason for this is that the availability of an EPC Information Service to meet a request about an EPC is itself sensitive business information (since it discloses the fact that the EPC has been observed and potentially allowing to conclude which company has handled the associated item). This property is unusual in directories of resources. Indeed, most serve published data that is supposed to be found. The sensitivity of the data therefore represents a significant difference between EPC Discovery Services and established Directories of Resources, such as UDDI.

The second problem is also related to security. Global scale directories such as those used for distributed content networks or the DNS, along with other directory technologies such as LDAP (Lightweight Directory Access Protocol), use the key for the directory entry to determine where in the distributed directory that entry should be stored. This may be unacceptable for an RFID network since the directory provider will control access to, and have visibility over all resources for a particular range of EPCs. Suggestions to encrypt the information may not be sufficient since the directory provider may still block communications. This static division of the EPC resource space can also allow attacks against particular companies or product ranges through Denial-of-Service attacks against the directory resources responsible for selected EPC ranges.

3.4. Directory of Clients

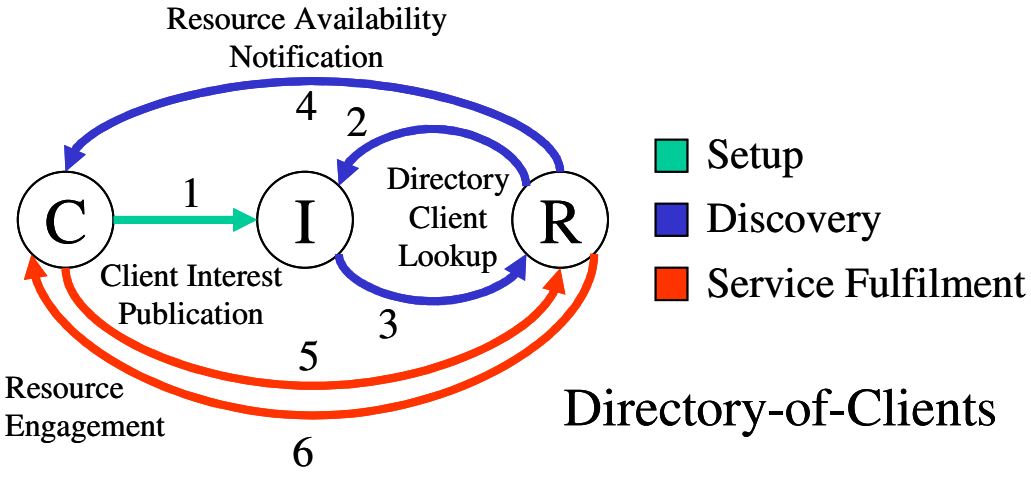


Figure 2: Directory-of-Clients Model

Figure 2 depicts the Directory-of-Clients model. The concrete steps are:

1. The client publishes its interest in certain EPC numbers (i.e., query keys) to the intermediary.
2. A resource looks up the keys in the intermediary to identity client queries it can serve.
3. The intermediary replies to the resource with the list of keys.
4. The resource notifies the client of its identity together with the fact that it has become aware of the client interest and holds relevant information on the requested keys . The notification from the resource to the client may be given directly (as shown in the figure) or may travel through the intermediary.
5. The client separately and directly queries each relevant resource for detailed information. The full query is now released.
6. A resource answers with detailed information as requested by the client.

In contrast to the Directory of Resources, we can take the approach of using the directory to store the key of client queries. This is useful if client queries have a long lifetime compared to the information stored within the resources (or if resources are only intermittently connected), thus reducing the churn on the directory. In the Directory-of-Clients model (cf. Figure 2) a resource notifies the client before the client engages the resource for Service Fulfilment. While in a general model the resource could initiate the Service Fulfilment, we note that in the current EPCglobal design the client must engage the EPCIS with a query or subscription to a standing query.

For RFID systems we expect the resources (i.e., EPCIS instances) to be permanently connected to the network and that the RFID event data stored in the resources typically does not expire. Therefore the characteristics of the Directory-of-Clients model do not come into effect in RFID systems and there is little motivation to support intermittent connectivity of resources or to support short-lived information. Rather, we expect most client queries to be one-off (as opposed to being long-lived, standing queries). Most supply chain scenarios will require an immediate (or at least time bounded) response to one-off queries. With the Directory-of-Client model, a client has no visibility or control of when resources connect to the Directory of Clients and thus when the query will be answered. In order to reduce the average response times, resources need to contact the directory more frequently, thus increasing the load of the directory as well as network traffic. For the above reasons, the Directory-of-Clients model is not suitable for the main mode of operation in RFID discovery services. On the other hand, this model seems to be well suited for long standing client

requests, for example, for the Notification-of-Events that the client expects to occur in the future. It could therefore be used in combination with another model in order to serve both one-off and standing queries.

3.5. Notification of Resources

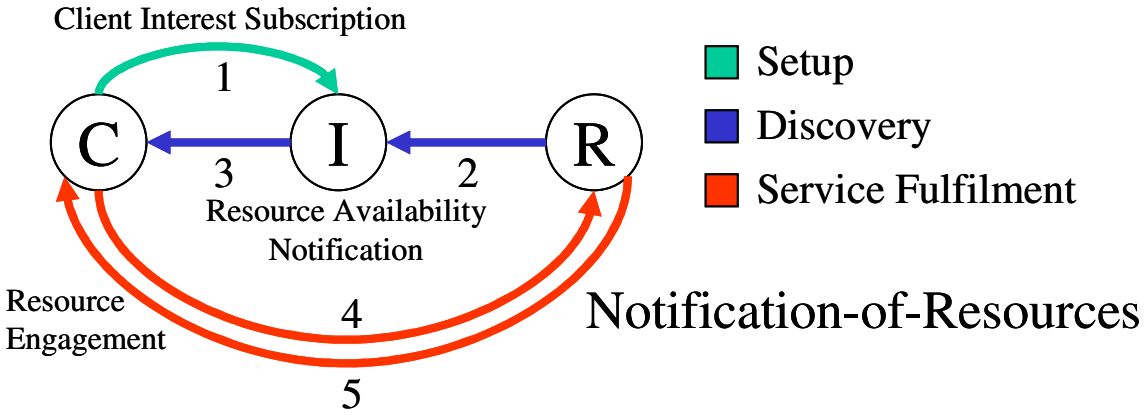


Figure 3: Notification-of-Resources Model

Figure 3 depicts the Notification-of-Resources model. The concrete steps are:

1. The client registers its interest in certain EPC numbers (i.e., query keys) with the intermediary.
2. A resource broadcasts its availability (i.e., its identity) and the set of EPC numbers it holds further information about into the communication network.
3. The communication network notifies the client of the relevant resources for its expressed interest.
4. The client directly queries each relevant resource for detailed information. The full query is now released.
5. A resource answers with detailed information as requested by the client.

In this model, resources, such as the EPC Information Services, broadcast the availability of EPC information to the intermediary, which is then forwarded to registered clients. Forwarding of the notification would typically be restricted by the publisher (i.e., the EPCIS) in the form of security controls, by the clients in the form of a subscription filter, or both. This technology is widely available as Message Oriented Middleware often used in the field of Enterprise Application Integration. These systems are typically limited in their coverage and scope, although some global examples are found such as Usenet news. Academic research has also attempted to address globally scalable publish/subscribe systems (see, for example, [1, 2, 5, 6, 7]). More localised resource broadcast can occur using network technologies such as a physical bus (e.g. USB), network multicast/broadcast (such as IP multicast), or radio broadcast (for example local services advertising over wi-fi).

This model is often suggested for location-based services to preserve the privacy of clients. A client may listen to the service broadcasts and choose whether to engage based on the service offered and any authentication checks the client wishes to perform.

Security of such systems can take two approaches. The first is to transmit to a group communication channel to which membership is restricted. The second approach is to not restrict the propagation of the message, but to encrypt it so that only selected parties can understand it. This latter approach requires the message to be transmitted with a group key. This key must be continuously managed as clients join and leave the security group, making it less suitable for dynamic audiences. In the first approach, security policies must be distributed into the network to control the flow of information.

We can consider that this communication model differs from the directory service approach in one key manner. In the directory service, a client is expected to perform infrequent connections to the Discovery Service and perform a query over historical resource information. In this model the client is assumed to be continuously connected to the network and respond in an event-driven manner to new EPC resources.

Of course each model can be extended to incorporate the characteristics of the other. For example, a directory service can include a subscription interface along with a request-reply communication interface. Similarly archival services can subscribe to the real-time Notification-of-Resources to provide a historical query capability. In this manner a Directory-of-Resources capability can be built over an underlying Notification-of-Resources model.

3.6. Notification of Clients

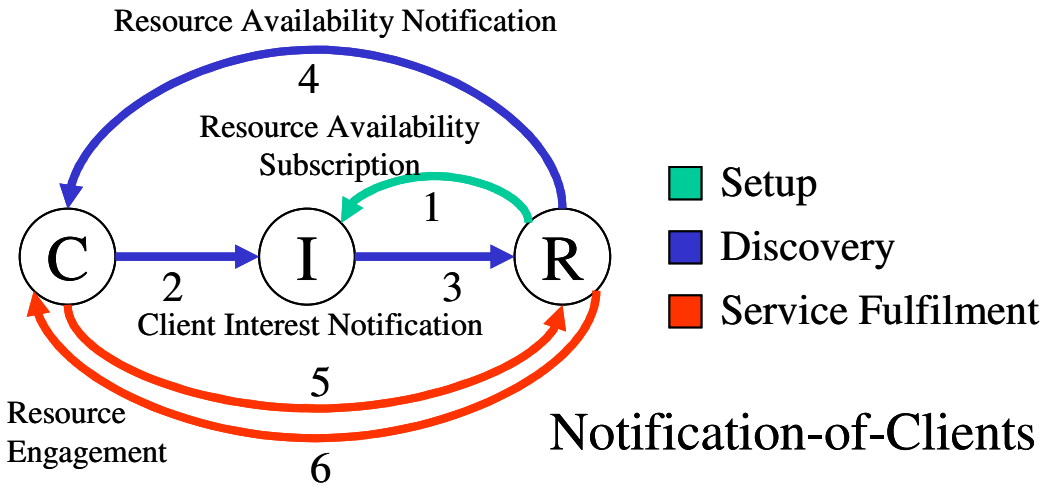


Figure 4: Notification-of-Clients Model

Figure 4 depicts the Notification-of-Clients model. The concrete steps are:

1. A resource publishes its availability (i.e. resource reference) and the set of EPC numbers it holds further information about into the communication network.
2. The client broadcasts its identity and its interest in an EPC numbers (or set of numbers) into the communication network.
3. The intermediary propagates the client notification to those resources which have published a matching EPC number in step 1.
4. The resource notifies the client of its identity together with the concrete EPC number it holds relevant information about. Multiple resources may reply to a client request.
5. The client directly queries each relevant resource for detailed information. The full query is now released.
6. A resource answers with detailed information as requested by the client.

This model uses the same technology base as the Notification-of-Resources, namely a publish/subscribe network. It differs because the client initiates the communication by publishing its resource requirements (cf. Figure 4). This is similar to a query to a Directory Service, except that the request is relayed to the resource itself (instead of being handled autonomously by the Directory Service). Resources that are both willing and able to assist the client may then establish communication with the client in order to serve future specific resource requests.

When compared to the Directory-of-Resources model, the client loses a degree of control. It must first announce some intention to engage with a resource to an unknown audience. If resources fail to respond the client will remain ignored and cannot take remedial action. Thus for unreliable resources and networks a Directory-of-Resources approach may be preferable (under the assumption that such a Directory is more reliable than the resources themselves). In contrast, the resource gains some form of control since it can choose which clients to engage dynamically instead of attempting to set static policies for the release of its directory or resource availability information.

Security again can take two approaches. Either the routing of the client request is limited by security policies, or the message itself can be encrypted with a group key.

The resources, as receivers of the communication, are expected to be continuously connected to the network to receive client demands. However, this model, when compared to the Notification-of-Resources model allows clients to be more transient in their connection, although the Directory of Resources also allows this transient client behaviour. It may also be preferable in terms of network traffic if there are fewer client requests than resource update events.

3.7. Meta Resource

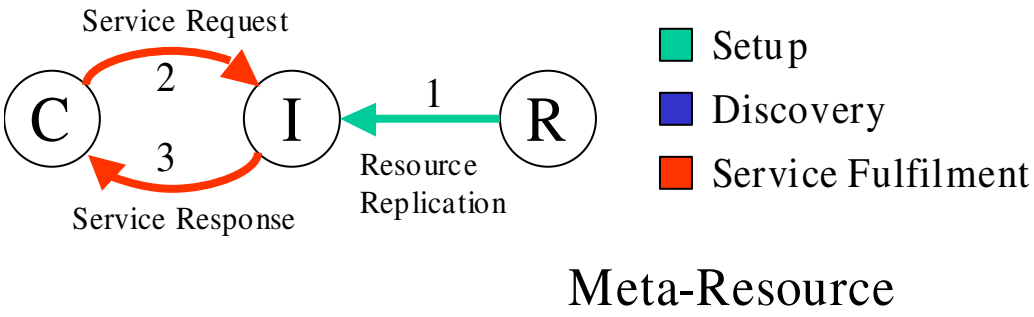


Figure 5: Meta-Resource Model

Figure 5 depicts the Meta-Resource model. The concrete steps are:

1. Each resource replicates its entire information set into a central intermediary (e.g., a data warehouse), which combines information from multiple resources. The intermediary stores the full data set of all resources connected to it.
2. The client queries the data warehouse for detailed information. The full query is specified.
3. The data-warehouse provides the client with the detailed information it solicited.

This model, depicted in Figure 5, combines multiple resources into a single physical resource. This can take the approach of data-warehousing for information resources, where all data is replicated to a single repository as shown in the diagram above. This model is popular for data consolidation within a single enterprise.

In the case of the EPCglobal network, data of multiple companies would need to be replicated into a central repository to serve all client requests. This implies a very high degree of trust in the intermediary. It would also require the intermediary to handle very high volumes of continuous updates from resources and store the associated volumes of data. For the above reasons the Meta-Resource is not suitable for RFID discovery services.

Alternatively, the Meta-Resource model may be combined with the Query Propagation model (see below). In this hybrid approach some resources may only provide a link instead of replicating the resource data. The intermediary propagates the query to the relevant resources. The resources then answer the query through the intermediary, which then aggregates them into a single reply to the client. In this hybrid approach the interface presented to the client remains consistent with the Meta-Resource model; the actual mechanics are transparent to the client. Aggregation of replies is optional in the Query-Propagation model, as discussed later.

3.8. Meta Client

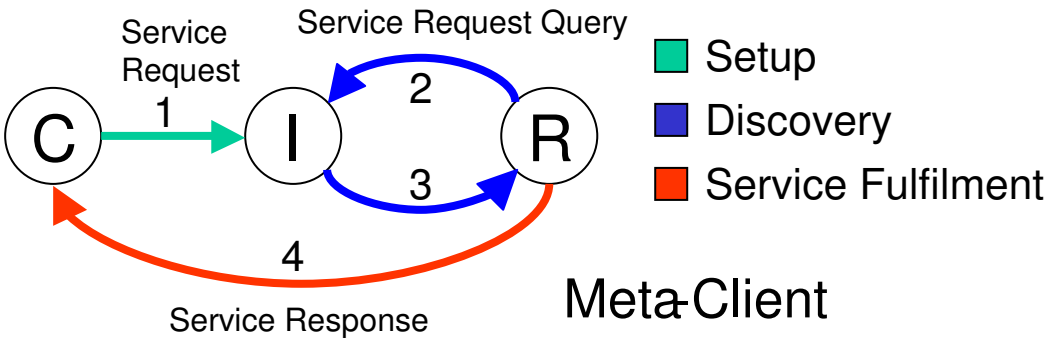


Figure 6: Meta-Client Model

Figure 6 depicts the Meta Client model. The concrete steps are:

1. The client registers the full query with the intermediary.
2. The resource looks up the available queries in order to identify which client queries it can serve.
3. The intermediary replies with the set of stored queries.
4. The resource directly contacts the client with the complete query result.

Whereas in the Directory-of-Clients model clients register broad interests (i.e., EPC numbers of interest), in the Meta-Client model (cf. Figure 6) clients register their fully specified queries. The resources, which may be intermittently connected, fetch these specific requests and can decide to act upon them.

This model is unsuitable as the main mode of interaction between clients and resources, that is, for one-off queries that are expected to be answered immediately. As in the Directory-of-Clients model, also in this model clients have no control of when and how often resources poll for client requests and thus become aware of a client request. The problem is intensified as different resources are likely to have individual polling schedules and thus the client cannot predict when it has received the complete response of all resources. On the other hand, this model seems to be well suited for long standing client requests, for example, for the Notification-of-Events that the client expects to occur in the future.

3.9. Notification of Events

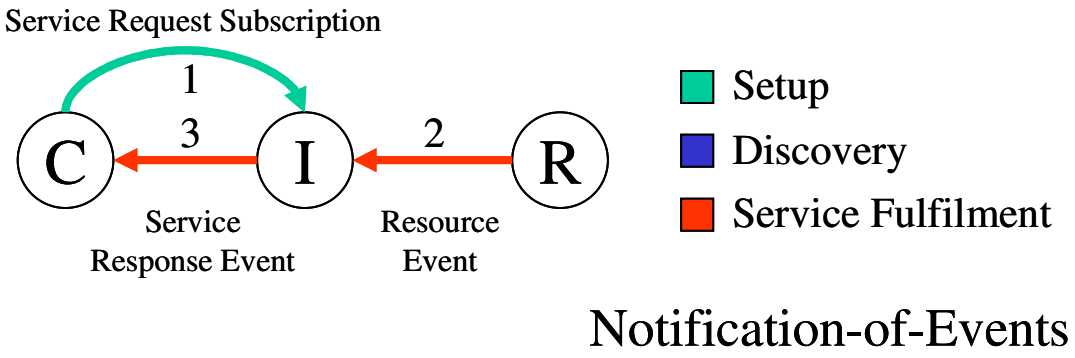


Figure 7: Notification-of-Events Model

Figure 7 depicts the Notification-of-Events model. The concrete steps are:

1. The client registers its specific and full query with the communication network. The intermediary thus stores full client queries (as opposed to query keys, i.e., EPC numbers only).
2. Each resource replicates its entire information set into the communication medium / a central repository.
3. The relevant and legitimately-accessible information is routed to clients according to their expressed interests.

Using the same technology base as the Notification-of-Resources models and Notification-of-Clients models, this approach (cf. Figure 7) differs by the fact that resources publish the information they hold onto the communication medium (either periodically or only once on becoming available). The information is then routed to interested clients, which have previously registered their interest with the intermediary. Direct interaction between the client and resource is not required. This model is common in information dissemination where we need to inform a large number of permanently connected receivers who may wish to respond immediately to an event.

The intermediary may also archive historical events and replay them on client request [9]. These archives support infrequently connected clients and are used to analyse long-term patterns in the event data. This model therefore is often used for the collection of sensor information. Such event archives may be considered as providing a hybrid with the Meta-Resource model discussed previously.

This model may be inappropriate for inter-organisational RFID systems due to the loss of fine-grained control by resources over the data. Manageable security policies are long-lived and thus control only broadly which receivers are allowed to receive what data. Dynamic changes to these policies, of policies that define security access based upon many attributes of the data are considered non-scalable.

Also such a model implies a higher degree of trust in the communication network, since much more information is being released.

3.10. Query Propagation

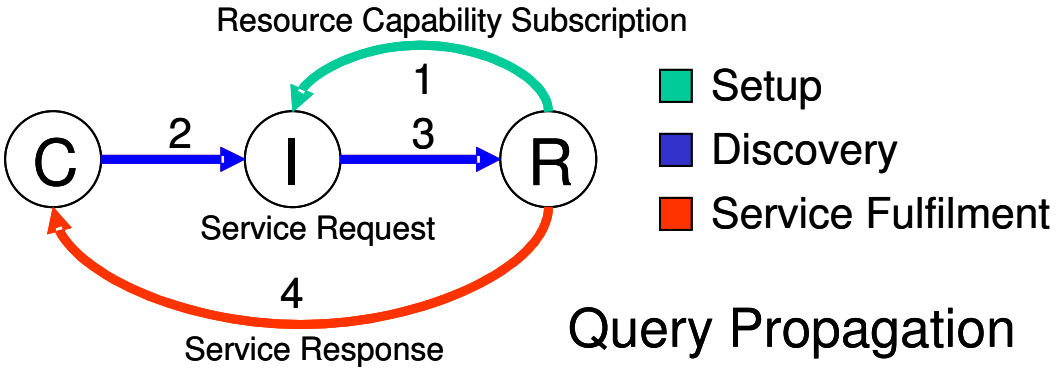


Figure 8: Query-Propagation Model

Figure 8 depicts the Query-Propagation model. The concrete steps are:

1. The resource publishes / registers key-value pairs containing EPC numbers and resource references to the intermediary.
2. A client issues a full query into the communication network.
3. The client query is selectively propagated to only those resources that are known to hold relevant information.
4. A resource answers at its own will with detailed information as requested by the client. The response from the resource to the client may be given directly (as shown in the figure) or travel through the intermediary.

Similar to the Notification-of-Clients model, Figure 8 depicts the Query-Propagation model in which the client releases to the intermediary a full resource query, which is sufficiently detailed to be answered by resources directly. The intermediary propagates the query to those resources that hold relevant information to answer the query. As in the Directory-of-Resources model, the intermediary stores the information (i.e., EPC numbers) about who has relevant data for a query. This information is released by the resources as part of the setup phase, which can be seen as the subscription for client queries.

In this model, the intermediary directly propagates client queries to relevant resources, allowing them to reply without delay. The model also supports clients to only transiently connect during Service Fulfilment. Information providers are (as in most of the other models) required to publish the EPC numbers they hold more information about, however, these numbers are never directly revealed to clients. Rather, resources stay in full control over what data they release to which clients. Access control to the resources' data is performed by the resources themselves. Also, resources can directly log all attempts (successful or failed) to access their data. As in any other model storing data on behalf of resources, this model requires trust in the intermediary not to reveal any of this data.

In this model, some form of access control to the intermediary (e.g., based on credentials passed with the client request) is still required to protect against, for example, malicious clients using the propagation functionality to overload resources (i.e., in denial-of-service attacks). Access control in Query Propagation serves a different function to that found in the Directory of Resources, where access control is used to restrict the release of resource information.

3.11. Summary and Conclusion

In the previous section we have outlined for completeness all possible communication models according to our taxonomy characteristics. We have dismissed several of these models as unsuitable for further in-depth investigation. As stated earlier, the selection was made on two criteria: interaction mode and trust in the Discovery Service and network. The table below summarizes our selections.

Model	Client Trust	Resource Trust	Response Latency	Status
Directory of Resources	Good	Concern	Good	Candidate
Directory of Clients	Concern	Good	Poor	Reject
Notification of Resources	Good	Concern	Good	Candidate
Notification of Clients	Concern	Good	Concern	Candidate
Meta Resource	Good	Poor	Good	Reject
Meta Client	Concern	Good	Poor	Reject
Notification of Events	Good	Poor	Good	Reject
Query Propagation	Concern	Good	Concern	Candidate

3.11.1. Interaction Mode and Transience of Connectivity

Interaction models that require the EPCIS to poll for potential clients are rejected for the main mode of operation, that is, for one-off queries. These are the Directory-of-Clients and Meta-Client models, because they are suited more to transient resources than transient clients.

3.11.2. Data Ownership and Trust

We rejected any model that requires the EPCIS owner to share detailed information with the Discovery Service without first gaining details of which clients require the detailed access and being able to refuse or negotiate this access. For these reasons we also reject the Meta-Resource and Notification-of-Events models. The following table summarizes the types of data that needs to be stored at the intermediary in order for it being able to complete discovery. Note that this data is not necessarily shared with the client or resource, respectively.

Model	Type of data stored at intermediary to complete discovery
Directory of Clients	clients' query keys (i.e., EPCs)
Notification of Resources	clients' query keys (i.e., EPCs)
Meta Client	full client queries
Notification of Events	full client queries
Notification of Clients	resource keys (i.e., EPCs) and resource refrences
Directory of Resources	resource keys (i.e., EPCs) and resource refrences
Query Propagation	resource keys (i.e., EPCs) and resource refrences
Meta Resource	n/a (no discovery phase, since all data of all resources replicated)

In the following section we present two RFID Discovery-Service design alternatives, which are built based on the candidate models selected in this section.

4. Selected Discovery Service Designs

The initial selection from the previous section leaves four models remaining for consideration and design development. The Directory-of-Resources and Notification-of-Resources models vary only in whether the client is predominantly request/reply or publish/subscribe. From the questionnaire and interview results we see that clients will require operation in both of these modes. Thus for the remainder of this document we consider these two models to be combined to provide a Directory Service with a notification capability for publish/subscribe operation. Henceforth we shall refer to this solely as the Directory Service design.

The remaining two models involve the client sending requests via an intermediate network to the permanently connected resources. In the Notification-of-Clients model, the client sends the minimum credentials and statement of interest to start a negotiation over finer grained access to the resource at a later time. In the Query Propagation model, the detailed resource access request is transmitted directly without the former negotiation phase. We can consider that these two models are not entirely disjoint, since in the former case, some indication of the subsequent access requests will be included. For an RFID system, this indication may consist of the EPC numbers that subsequent requests will include in their request for detailed EPCIS events. In this section we will discuss these models together under the category of Query Propagation design (with a more general client request being considered as a sub-design).

In the remainder of this section we present and discuss the two alternative Discovery Service designs. In the discussion we will further elaborate on the criteria that we used for the selection of the initial models. Furthermore, we will take into account various aspects that we have omitted in the initial discussion, such as scalability, performance, and resilience.

4.1. RFID Directory Service

The Directory Service design discussed in this section combines the Directory-of-Resources and Notification-of-Resources models, which are repeated below for the reader's convenience.

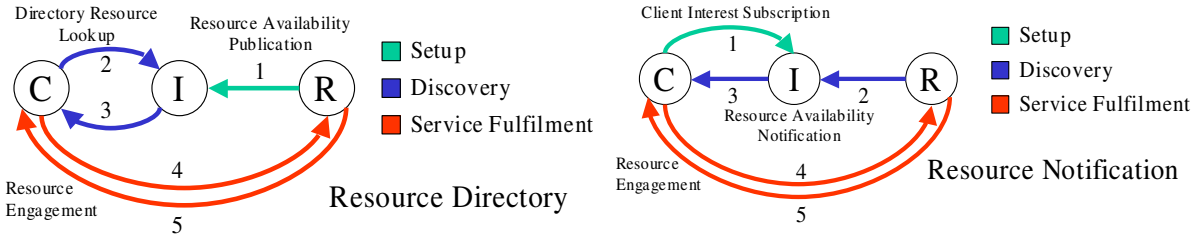


Figure 9 shows the operation of a directory-based Discovery Service within an EPC RFID Architecture. The resources, consisting typically of EPC Information Services (EPCIS), publish selected information to one or more Discovery Services. The information may be replicated across multiple Discovery Services for dissemination to different business communities, or for redundancy. Replication of information and federation of Discovery Services are not discussed here.

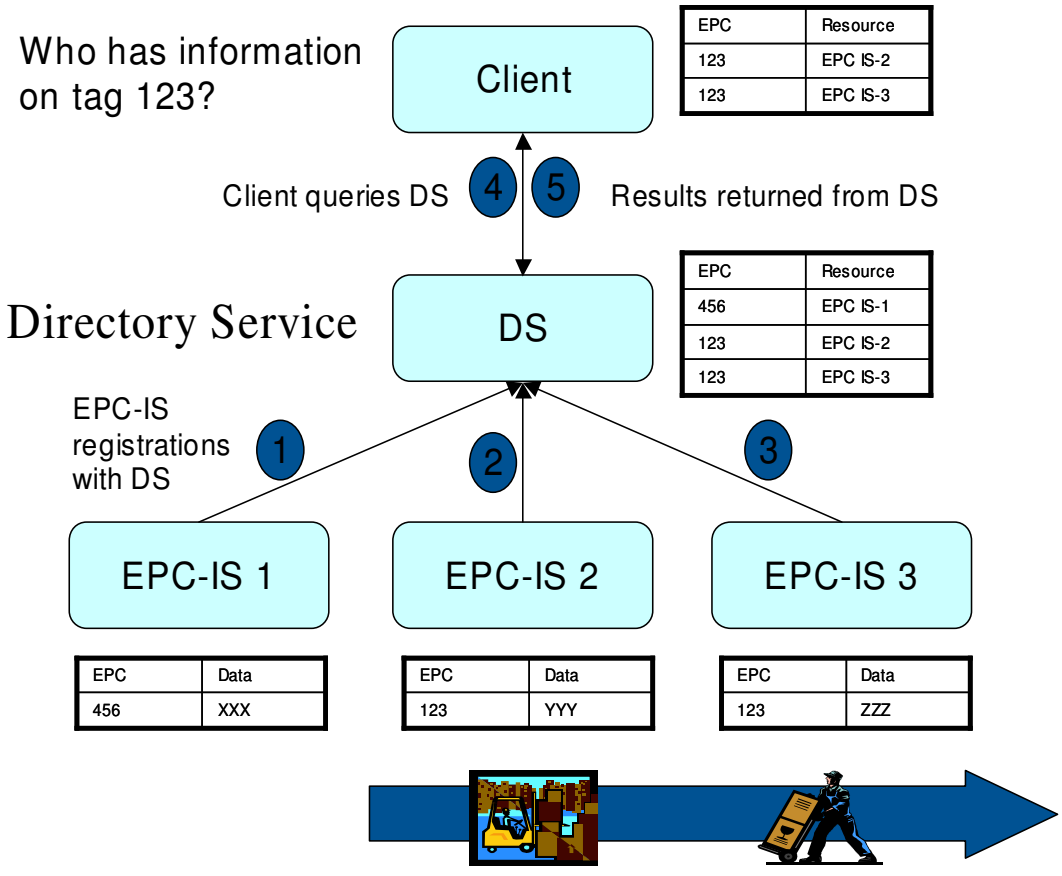


Figure 9: Operation of a directory-based Discovery Service within an EPC RFID architecture.

Querying DS

The client may query the Discovery Service using a query interface, similar to that specified in the EPCIS standard, but including at least an EPC (or EPC list/range). This may consist of a one-off query, or a standing query resulting in a subscription to the Discovery Service. In the latter case, any new publication of records to the Discovery Service will be compared against the list of current subscriptions. Where client subscriptions match (and subject to security constraints) these new records are notified to the waiting clients (or delegated to message queues).

Any client query can be answered immediately (subject to security negotiations) by the Discovery Service without recourse to additional parties. The EPCIS resources do not have to be available at the instant of the client query. This allows for mobile or intermittently connected EPCIS resources, although the advantage is unclear as the EPCIS must be available at a later time to respond to the client's EPCIS query. However, this is still an advantage over a total lack of Discovery Services, where end-to-end traceability/completeness might be severely reduced if one intermediate EPCIS within the supply chain fails to respond and is on the critical path for providing an onward link in an approach where each EPCIS links to the next in the chain. It also means that a Discovery Service is not required to maintain the client session while communicating with onward systems. This can aid scalability and resistance to denial-of-service attacks and reduce the latency of the response to the client.

The minimal information published to a Discovery Service must be the EPC, that is, the unique identification number associated with the RFID tag, along with a link that can be used to communicate with the resource. For a web-service interface to an EPCIS this would consist of the location of the Web-Service interface description.

Querying EPCIS

In response to a client query, a list of EPC numbers and associated links to EPCIS instances are returned to the client. Subsequently, the client directly contacts one or multiple of the returned EPCIS instances. This process can be a burden for the client when several EPCIS instances need to be contacted. If some EPCIS instances are slow to reply (due to network traffic or server load) receiving the complete reply to a query can take significant time. Therefore, it may be required to issuing the individual request in parallel which increases the software complexity as connection state needs to be maintained. On the other hand (and as discussed previously), the client is in full control of its requests and may choose to alter the query in subsequent requests or may choose to abort the request altogether. This is particularly useful if all or part of the answer has been returned to the client in a request to a previous resource in the list of returned resources. If the complete set of replies by all resources is not required, this mode can reduce the query speed compared to, for example, the Query-Propagation model where always the full set of replies is returned.

Data ownership consideration

Access to Discovery Service data records must be permitted by both the operator of the Discovery Service, and the owner of the resource records. Thus the client must have a trust relationship with both the Discovery Service and the EPC resources that publish records to the DS. Clients that are unknown to a resource are unlikely to be granted access to the DS records about that resource. A mechanism is required either to introduce potential clients to resources, or to establish trust through intermediate parties. For example, a resource may trust clients that have obtained membership of a particular group or federation. The DS may implement such a mechanism, or rely on wider trust establishment mechanisms such as those being developed for Web Services (e.g., Liberty Alliance [10] and WS-Federation [11]).

Once records are retrieved from the DS, the client has the reciprocal problem of whether the EPC resource (e.g., EPCIS) is trusted. The EPC resource also has to trust the client to release the EPC trace data. If sufficient relationships do not already exist as a result of

gaining access to the Discovery Service, then the client and EPC resource will need to negotiate a further trust relationship."

Advanced Queries

Although in the 'pure' directory service model only the resource refrence must be published, optionally additional information may be included, to provide business context. Such information may allow more intelligent matching of resources through the use of more expressive client queries. For example, instead of just fetching all resources that have information on a particular EPC, the client may restrict the set of results by specifying the type of event, for example, the business step (encoded in the field bizStep in EPCIS v1.0) for which it is interested. To allow more intelligent matching of client queries at the Discovery Service level, resources need to reveal more than the minimal set of data to the Discovery Service. Sharing additional data over the minimal set is often considered a trust problem by information providers. Either the publisher of the record to the Discovery Service, or the Discovery Service itself may supply additional meta-data. For example the publisher of the record may provide an event time, while the Discovery Service may maintain the time at which the record was submitted.

Enforcing Security Policies

The directory-based Discovery Service must maintain a number of security policies. Its own policies will specify who can publish information to the Discovery Service, and what information they may/must publish (e.g., which EPCs and mandatory additional fields such as a signature). The Discovery Service may also broadly specify the clients that are allowed to use the service. Additionally it is expected that the Directory Service will delegate the ability to define security policies to the publishers of the Discovery Service records. This will allow the publishers to maintain fine-grained control over which clients can see which records (and individual record attributes).

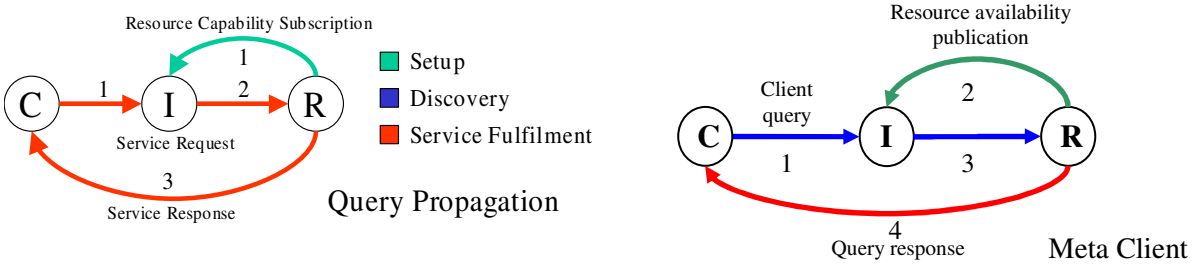
In this manner, the Discovery Service acts as a trusted broker between the client and the EPCIS resource. The resource will not find out about the client interests until the point at which the client initiates direct communication with the EPCIS. The, client similarly, will not discover the EPCIS resources unless the security policies allow them to do so.

As mentioned before, the necessity to protect the sensitive data records that are stored in the Discovery Service from unauthorised client access requires enforcing access control on the level on the Discovery Service. This means that access control is effectively duplicated, once in the Discovery Service and once again for the subsequent query in the EPCIS. Note that this duplication of access control does not provide additional security; it is merely to protect the replicated data. (In fact, the Discovery Service is an additional target for attackers.)

One issue of duplicated access control may be that EPCIS instances must carefully maintain the consistency of access control policies for itself and for the Discovery Service. If a company decides to modify its access control policies, immediate propagation to the DS is also required. In order for EPCIS instances to maintain their policies, a standard for describing the semantics and maintaining access control policies need to be part of the DS. Such a standard increases the complexity of the EPCIS and DS software as well as that of the standardisation process.

4.2. RFID Query Relay

The Query-relay directory service design discussed in this section combines the Query-Propagation and Meta-Client models. The graphical representations of the interaction models from the previous section are repeated below for the reader's convenience.



The key idea of this design is to not reveal any of the resources' data to clients (i.e., EPC numbers and resource references stored in the intermediary) but to instead propagate client queries to the resources. Thus, the main mode of operation of this discovery-service design employs the Query-Propagation model. That is, the Query Relay forwards client queries to waiting resources (i.e., typically EPCIS instances), which can then answer the query directly. Alternatively, the Query Relay also supports subscriptions of client queries, following the Meta-Client model. That is, the query may take the form of a standing query for which a subscription is maintained (and stored) by the intermediary. Standing queries are forwarded to resources as soon as they register new events matching the query; the standing query then travels piggy-backed on the resource's availability publication (see step 3 in the figure depicting the Meta-Client model above).

The client query itself can take two forms, (1) a resource query to identify relevant resources for specified EPC numbers or (2) a full query directly returning the desired query result. The resource query (as previously described in the Notification-of-Clients model) is the preferred query mode for clients with a long-lasting interest in certain products and which expect further queries on the same EPC number. In this case, clients may chose to maintain a local cache of EPC numbers and the associated resources already identified rather than repeatedly placing resource queries to the DS. This approach requires the client to subsequently post the full query to the identified resources. From a client perspective, this mode resembles the Directory-Service design. (It may even be indistinguishable from it if the replies travel through and are aggregated by the intermediary, an option which we will discuss below). The full query (as previously described in the Query-Propagation model) is useful for one-off queries, that is, when clients expect no need for further queries on the same EPC number. An example could be a lost and found application for finding lost reusable assets.

In this model, the intermediary may be realized as a single server. Or it may be realized as a network of federated servers, for example, for load balancing. The latter approach is common in publish-subscribe overlay networks [cf. 1 - 7]. As with the Directory-Service model discussed in the previous section, many such networks may exist to serve the needs of different communities, and ECPIS resources may be registered to receive requests over multiple networks.

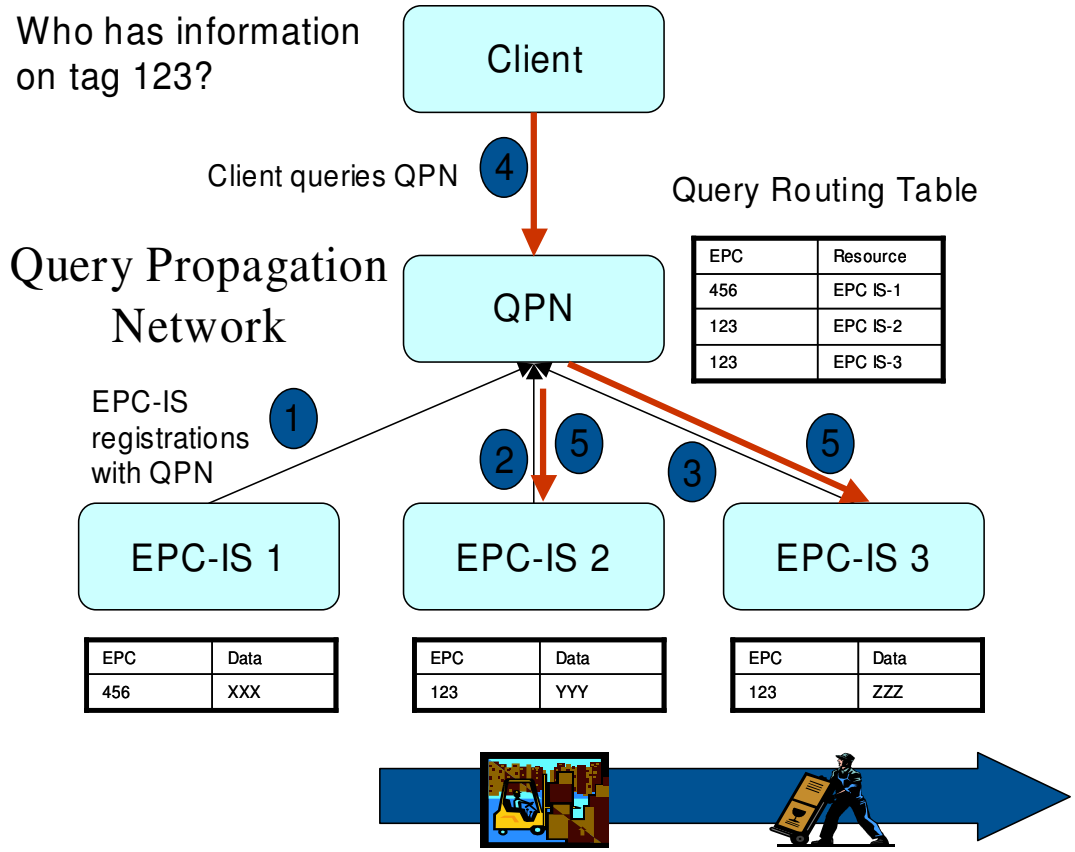


Figure 10: Operation of a Query Relay within an EPC RFID architecture.

Figure 10 shows the basic operation of the Query Relay. ECPIS resources register to receive client queries at one or more Query Relays. This registration specifies the EPC numbers (i.e., keys) the EPCIS resource holds information about plus a reference to the resource. This may optionally include further routing data (i.e., secondary keys), for example, the EPC event type (i.e., bizStep) to allow more selective routing of client requests. This is desirable to reduce network traffic and to relieve EPCIS resources from queries that they cannot answer.

Relaying Queries

Relaying the client's query has the advantage that EPCIS resources retain full data ownership, that is

(1) Resources stay in full control of which of their data is released to whom because only they control the access to their data. No data from resources is revealed to clients directly.
(2) Resources can log all (successful and failed) attempts to access their data.
(3) Resources can deny access to certain clients without making the client aware of it, following the lines that access control policies are typically considered sensitive data.

Relaying queries on behalf of the client also relives the client from having to connect to and access the relevant resources directly. Without increasing the software complexity of the client, the Query Relay could implement dynamic strategies for parallel querying multiple resources concurrently and for to cope with the idiosyncrasies of the network, intermitted disconnects, as well as with slow and unresponsive clients. There is a tradeoff between the client's convenience and the client's ability to control the query process, for example, to modify and abort the query when partial results have been received (as is possible in the Directory Service design).

Routing Replies

There are two principal alternatives of routing the query response back to the client. Firstly, a queried EPCIS can establish direct connection to the client in order to convey the query response, bypassing the discovery service. For example, the responses may be routed over a VPN or the Internet to the client's IP address. This approach is shown in the figures depicting the Query-Propagation and Meta-Client models above. In the second alternative, the query response is routed through the Query Relay. This may be achieved in two ways. The first approach is to maintain session state in the Query Relay for each client request. This state is used to direct the query responses back to the client. The session state can be dropped when the last response has passed through the Query Relay. A second approach is to include the client's return address in the resources' response. The Query Relay can then use the address to pass the query response back to the client without having to keep session state.

Routing the reply via the Query Relay allows the consolidation of the responses from multiple EPCIS resources. It also allows decoupling of client and resources, for example, to hide the client's network address from EPCIS instances and vice versa. It also allows the consolidated responses of multiple resources to be returned in a single (possibly synchronous) reply to the client. On the other hand, maintaining session state adds complexity to the implementation of the Query Relay and may be disadvantageous in terms of scalability.

Routing the replies through the Query Relay without session state but only based on the client address instead complicates consolidation of replies since the discovery service cannot determine how many (if any) responses are expected. Sending the reply directly from the resource to the client and bypassing the discovery service, however, has the advantage that no additional data is revealed.

Receiving Replies

If query responses are returned to the client without consolidation in the Query Relay (either if sent directly from resources to the clients or if routed through a stateless Query Relay), clients need to receive and combine the potentially multiple replies. Receiving and combining replies in the client is an iterative task, involving two trivial steps: (1) accepting a reply message from the network and (2) consolidating it with all previously received messages. However, there is a fundamental problem with the timeliness property of the system. If the client has no indication of how many replies to expect, it cannot decide the termination of the query, that is, if and when all replies of resources *willing to reply* have received by the client or if particularly "slow" replies are still underway.

Note the fundamental difference between slow replies and replies withheld by resources. Withheld replies are not meant to be delivered to the client, nor is the information that a reply has been withheld in the first place. (If that information was to be given to the client, the resource can trivially do so by returning a special reply.) For slow replies, however, all parties have an interest in communicating this information.

In principle a client would have to wait forever for replies that still might come. A partial solution is to return to the client the number of actual resources the query has been forwarded to by the Query Relay. This number can be trivially provided by the Query Relay in the return communication with the client. The client can now decide termination if and only if all resources reply "in time". However, the client can still not decide termination if there are withheld replies since they are distinguishable from slow replies. Moreover this approach may be considered harmful as the client knows the actual number (albeit not the identity) of resources holding more information on a particular EPC even though some resources may have declined to interact with the client.

A common and pragmatic solution approach is to use timeout intervals after placing the initial query. This approach assumes that after the timeout all replies have been received. Though easy and efficient to implement, the client faces the danger of disregarding potentially important but slow replies.

Client Confidentiality

In the Query-Relay design, clients either release queries containing EPC queries only or full queries. Therefore the intermediary must be trusted to manage the confidentiality of the client requests. Typically more than the relevant resources will gain visibility of the client query, that is, more than are able to respond meaningfully.

Enforcing Security Policies

Like the directory model, the Query Relay will implement security policies. These policies will determine which EPCIS resources are allowed to register with the Query Relay for which EPCs and other associated routing data. They will also determine which clients are allowed to submit queries to the Query Relay, and may optionally constrain the parameters of the query or the response messages that are allowed.

Access control policies for the Query Relay serve a different function compared to the Directory-Service model. In the latter they are used to protect the confidentiality of the resource, that is, to protect access to the resources' sensitive data (EPCs and resource references) from unauthorized and potentially harmful clients. To serve this function they may need to be very fine-grained. In the Query Relay, access control policies are intended to protect the systems from denial-of-service attacks and to reduce the query traffic reaching an EPCIS. This may be achieved with relatively light-weight access control mechanisms.

5. Analysis of Design Candidates

Having presented and discussed two candidate designs in the last section, in this section we will discuss further topics of interest in the design and implementation of discovery services for RFID. We will frequently revert to the previous discussions to illustrate additional points and suggest implementation options as well as future work.

5.1. Security and Trust

The different communication models critically affect the underlying security properties of the system and the steps that must be taken. In this section we discuss a number of security considerations and how these might affect the choice of communication model and the final design of a Discovery Service. There are a variety of security requirements that may be considered when designing a Discovery Service.

Client Confidentiality

Client confidentiality refers to the characteristic of the system to not reveal sensitive client information to third-parties. In the case of RFID discovery services, the client query is typically considered confidential client information. Revealing the client queries to potential attackers would allow them to analyse the queries and contained EPC numbers. Such an analysis could allow them to gain insight on the physical objects (identified by the EPC number in the query) handled within the client's organization and the business steps performed within the client's organization.

The client of the Discovery Service submits credentials (such as their identity or role) along with their expressed interest (EPC query) to the Discovery Service. The Discovery Service may also have visibility over lower layer network addresses that the client uses.

The Directory Service acts as a trusted broker between the client and the resource. If the resource's access control policy on the Directory Service is not fulfilled, no information is exchanged between the client and resource. If the access control policy is fulfilled then the network address of the EPCIS is released to the client to then initiate contact with the EPCIS. No further information need be released (unless desired and specified by the resource's access control policy).

In the Query Propagation model the client credentials and EPC interests are released to all resources that have expressed (and been allowed) to register an interest in receiving such communications, basically by pretending to possess information on a certain EPC number. The Publishers who receive such client information can be grouped into four categories:

1) Those who hold no legitimate information about an EPC
2) Those that have no information relevant to the client, but do hold other legitimate information on that EPC
3) Those who hold information relevant to the client but choose not to respond
4) Those who hold relevant information and respond to the client

To improve client confidentiality we may wish to restrict those resources that fall into category 2 by allowing finer-grained registrations of the EPCIS resources to the network. Additionally the network, in conjunction with clients, should attempt to police those resources that fall into category 1. It should be noted that a resource in category 1 is not necessarily malicious, but may instead be managing the scalability of its registration with the Query Propagation network through subscribing to a range of EPCs where it may not hold information on every EPC in the range. From the consideration of client confidentiality such behaviour should be discouraged. Discovery Services may even need to make use of auxiliary data analysis tools to intelligently check the plausibility of the records asserted by a publisher in order to detect and prevent such behaviour at the time when records are

published. For example, it might check that a resource owner cannot claim to have had custody of an EPC during a time period that overlaps with the period when another resource owner claimed to have custody of the same object.

A pragmatic and easy approach to improve the client confidentiality of Query Relays is to allow clients to use blacklist or whitelists. Blacklist can be used to prevent the forwarding of client queries to the known set of competitors and dubious resources. This approach is limited since typically not all of them are known. Whitelists, on the other hand, can be used to restrict forwarding the query to the set of trusted business partners. The drawback of this approach is that unknown yet trustworthy resources that have relevant information regarding the query could not be found. Blacklists and whitelists could be sent with a particular client query or be stored and maintained by the client in the Query Relay.

Resource Confidentiality

The EPCIS resource submits credentials (such as their identity) to the Discovery Service, along with information selected from the EPCIS repository (such as EPCs recorded). The Discovery Service also has visibility over the resources' network addresses. Any access policies submitted by the resource are also confidential information.

In the Directory Service model, the release of resource information to clients should be controlled through access policies. These policies will cover both the desires of the Discovery Service, and the resources. Each publisher of resource records to the DS should be able to create policies that control access to their own records. The DS-controlled policies specify the overall behaviour of the DS service irrespective of the data records and will over-ride any publisher policies. For example, the DS may specify that a regulator is permitted to see any published records within an EPC range. Publishers should be aware of such policies before releasing data to the DS. The resource confidentiality may be compromised if the resource's security policies are not fine-grained enough, or are not kept up-to-date with the resource's actual desires. Thus scalability and management of the security policies should be a prime consideration for a Directory Service. In the Directory Service model it is hard for a resource to receive requests for permission to access its DS data records from unknown clients since the availability and contact address for the resource will be protected by the access policy. Methods to address this problem include releasing a temporary network address for the resource or other point at which negotiation for access can take place.

In the Query Propagation network, the resource is able to decide without delegation to the Discovery Service, whether to engage with the client. It will have access to the most complete and up-to-date security policies, and may also include dynamic information (such as resource load) in the decision. This model may also work better when unsolicited client communications are encouraged.

Information Integrity

Any information stored within the Discovery Service should not be able to be compromised. Directory records and routing tables should only be erased or modified according to security policies. It is expected that the resource publishing this information will retain the right to modify or delete this information, although this may be over-ridden by Discovery Service policies (for example to maintain a journal for regulated supply chains).

Information integrity is not directly addressed by the presented DS designs. Additional mechanisms to enforce information integrity need to be employed. For example, in the Directory Service model, where resource information is returned to the client, such information may be digitally signed in order to authenticate the origin. Similarly for the Query Propagation model, the client query may also carry the client signature.

Service Availability

The Discovery Service should be designed to be resilient to Denial-of-Service attacks, such as network bandwidth denial. Furthermore the design should not compromise the clients or resources in terms of such attacks.

The Directory Service is involved in the initial EPCIS address lookup and then not further involved in the interaction between the client and EPCIS resources. Attacks on the availability of the Discovery Service will not prevent clients communicating with EPCIS repositories. Clients may also easily change their network addresses without significant repercussions. The EPCIS address is only released to clients fulfilling the access policy restrictions, helping prevent attacks on the EPCIS. Where the address of the EPCIS interface is compromised and subject to Denial-of-Service attacks, the Directory records may be updated to reference a new EPCIS location. To be able to make such changes efficiently, this suggests a need for any model to be able to decouple the current (and possibly mutable) EPCIS address of a resource from its immutable resource ID, rather than embedding the EPCIS address within the record. This has further implications for records that are digitally signed by the resource owner.

In the Query-Relay design, if clients typically rely on full query mode (i.e., the Query-Propagation model) they are particularly dependent on the availability of the discovery service. In this case the client's local cache (if one is maintained at all) would probably be incomplete since it would be based on previous detailed queries.

Attack Scenarios

Attackers could misuse a Query-Relay discovery service as relay for fake queries in order to launch denial-of-service (DoS) attacks on EPCIS repositories. For example, attackers may construct queries to load EPCIS instances that handle particular products. Potential countermeasures include authenticating clients before allowing them to use the discovery service and enforcing maximum rates of queries per client and per resource.

An attack scenario relevant for both discovery service designs is the registration of inexistent EPCIS instances addresses for already assigned EPC numbers. This attack could increase the network load and delay the query response since connections to inexistent services are typically only dropped after a timeout and several repeated connection attempts. For the same reasons this attack could also increase the load to query actual resources for clients of the Directory-Service discovery service and for the Query-Relay discovery service itself. For the Query Relay only the discovery service itself is affected since clients are only notified after successfully contacting resources. As a countermeasure, the Query Relay and each client of Directory Service could identify resources that fail repeatedly when making queries and remove them from the resource cache (or blacklist them). In variation of this scenario, an attacker could register arbitrary service addresses for assigned EPC numbers in order to launch DoS attacks on them. A potential countermeasure is the authentication of resources before allowing them to register EPC numbers and resource references.

Another attack scenario relevant for both discovery service designs is the impersonation of other clients by malicious clients. By posting a query with an assumed identity of a known client, an attacker could lead EPCIS instances (or the DS itself) to deduce that a particular item has been observed by the real client or at a particular location. Such attacks may be used, for example, to disrupt legal or trigger unjustified anti-counterfeiting investigations (as discussed in BRIDGE's work package 5). Countermeasures include authentication of DS clients.

Interworking with NATs and Firewalls

The Discovery Service Design should be able to work in harmony with network middleboxes such as Network Address Translators and Firewalls. Failure to allow clients to operate behind such middleboxes will compromise the ability to use the Discovery Service. NATs use

information on the outbound communication (from client to Discovery Service) to route the response(s) back to the client. Stateful firewalls also match returning traffic with outbound addresses. Responses coming from unexpected network addresses will pose problems for companies operating private address spaces or firewalls. A solution is to provide a client proxy with a public network address and allow incoming traffic from unknown addresses to this proxy. The traffic can then be routed to the client after inspecting the response (e.g. using a session identifier).

This problem is simplified if responses are returned directly in response to the client request. Thus the Directory Service model, or a Query Propagation model where the responses are routed via the Query Propagation network may be preferred. It should also be noted that any solution using a message transport network may suffer a similar problem since the network address of the response will be that of the message router.

Management of Access Control Policies

Key to all designs is the ability for the Publisher to manage the permissions for clients to access the Discovery Service and reach the EPCIS repositories. We should not forget that a similar set of access control restrictions are also required on the EPCIS since in any model communication may be routed directly to the EPCIS from external clients. Another set of access control permissions also exists to manage which resources may register with the Discovery Service.

In the Directory Service model the resource requires the ability to provide detailed policies to be enforced by the Discovery Service so that the Discovery Service may act as a trusted broker to handle client requests without contacting the resource. These policies may be considered to be a subset of the policies required by the EPCIS as any client with permissions to access the EPCIS should also have access to the Discovery Service. Thus, although the policy state held by the Discovery Service is considerable, the management problem can be considered to reduce to be an incremental cost above that required for managing the EPCIS access policy and it may even be possible to develop a common framework for expressing and enforcing access control policies, which can be applied at both the EPCIS and Discovery Service layers.

In comparison the Query Propagation network uses the policies stored at the EPCIS resource. In addition, resource policies may be pushed into the network to reduce the load on the resource, although these policies may be much less granular or less specific to individual records.

Both the Directory Service and Query Propagation models also have Discovery Service security policies. The control over which clients may use the Discovery Service is similar in both cases. Failure of security policies to restrict the registration of the resource in each case is slightly different. A particular concern is an impostor resource or ‘honey pot’ whose primary purpose is to harvest information about client queries. In the Directory Service, the address of such a ‘honey pot’ EPCIS resource location will be released to the client. In the Query Propagation model this may also be true, but additionally the client request is visible to the ‘honey pot’ EPCIS resource. In the Directory Service model the client has the opportunity to apply an additional level of scrutiny before contacting the EPCIS resource directly.

5.2. Network Performance and Resilience of Design Candidates

The choice of communication model also critically affects many issues around the service performance, network load, resilience, and scalability of the design. In this section we discuss some considerations that should be taken into account when designing the Discovery Service. There are a variety of scalability and performance factors that may be considered when designing a Discovery Service.

Persistent state on the Discovery Service

We can consider simply the state that the Discovery Service is required to store in long term memory, yet be able to retrieve for processing and response for immediate queries from clients. This state includes the Directory Service EPC data records, along with security policies. It also may include long-term subscriptions to new Discovery Service records. In the Query Propagation network, again the security policies and routing tables are long-lived. In the Query Propagation network the client subscriptions will be maintained by the EPCIS resource and not burden the Discovery Service.

Management

The persistent data on the Discovery Service requires management. Client subscriptions should be self-managing with automated removal of redundant state information. Discovery Services may also provide automated retention management for the Discovery Service EPC records or routing table entries. For security policies tools should be provided to manage security across multiple Discovery Services, EPCIS instances and other resources.

Transient/Session State on the Discovery Service

The use of the Discovery Service by a client should result in minimal state information being retained for the minimal amount of time. Ideally, the Discovery Service should handle the client request without further communication with onward services. Such onward communication necessitates the persistence of state for managing the transaction and the return communication. This should be considered if opting for a Query Propagation model that routes the EPCIS resource responses.

Transaction Duration, Transparency & Predictability

The client should receive a response to its query with minimal latency. The client should be able to manage its communication with the Discovery Service and other services. This typically means that each communication should be short and predictable. Ideally the client should be able to detect failed communications in a timely manner, and retry only the part of the communication that failed.

In the Directory Service model the client is in the optimum position of controlling a series of one-to-one communications with the Discovery Service and EPCIS resources. In the Query Propagation network the EPCIS resources are decoupled from the client (for the request communication). Thus the client must wait for a time interval to ensure that all EPCIS resources have an opportunity to respond. It this time interval is too short, the client will not know which EPCIS resources have failed to respond in order to selectively repeat the communication.

Caching

Some consideration should be given to the ability for the Discovery Service and the client to cache information for the improvement of performance. In the Directory model the Discovery Service already acts as a cache for resource availability information. Once retrieved from the Discovery Service such information may also be cached by the client for repeated later connections to the resource. In the Query Propagation model client requests must be routed to the end resources since there is insufficient information within the Query Propagation network to serve the request. Any responses obtained by the client may be cached for later direct communication with the resources, for example if the Query Propagation Network fails. However, in the Query Propagation model where fine-grained queries are submitted to obtain service from the EPCIS resources, many resources will remain undiscovered for later queries about the same EPC if the initial query was too specific, unless the EPCIS resource always returns a response to identify itself as a potential resource for that EPC, even if it has no events to provide for that specific query.

Discovery-Service Processing Load
The Discovery Service should be able to handle many simultaneous client communications. This means that each request should be computed quickly and with minimal computational effort.

This processing load is affected by the matching of the client request to the Directory records or network routing tables. It is also affected by the application of security policies. The provision of additional attributes in the Directory Service records should be carefully considered. Not only will these attributes increase the complexity of the Directory search, but also lead to a proliferation of finer-grained security policies if such attributes are restricted to limited clients.

In the Query Propagation network a similar problem exists with finer-grained routing tables including additional attributes. Although these allow the filtering of client request traffic and reduce network bandwidth and load on the EPCIS resource, the load on the network router will increase.

6. Conclusions

In this document we have discussed a taxonomy of communication models that may be considered for RFID Discovery Services and wide area RFID information networks. We have selected some of the more promising models and discussed the benefits and research challenges associated with each design. In particular the choice should be considered carefully for its impact on security, performance and scalability.

Although the Directory Service model is a traditional well-proven approach to this type of problem, its application in RFID architectures poses some unique challenges, particularly on the confidentiality of EPCIS resources. Key challenges are the delegated control and the scalable expression, evaluation and enforcement of security policies.

In contrast the Query Propagation model is perhaps a less obvious candidate, but such routing networks have seen widespread use, for example in peer-to-peer content retrieval systems. Such networks face similar challenges both within and outside an RFID context. Honey Pots are often used to gather client information and the injection of false information into these networks is a widespread problem. Thus, the challenge is around the security of resource registration and policing resource behaviour.

7. Bibliography

[1] Carzaniga, A.; Rosenblum, D. S. & Wolf, A. L. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, 2000, 219-227

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems, 19(3):332–383, 2001.

[3] Astley, M.; Auerbach, J.; Bhola, S.; Buttner, G.; Kaplan, M.; Miller, K.; Robert Saccone, J.; Strom, R.; Sturman, D. C.; Ward, M. J. & Zhao, Y. Achieving Scalability and Throughput in a Publish/Subscribe System. Technical Report RC23103. *IBM Research Division Thomas J. Watson Research Center,* 2004

[4] Gero Mühl, Ludger Fiege, and Peter R. Pietzuch. *Distributed Event-Based Systems*. Springer, August 2006.

[5] IBM. Gryphon: Publish/subscribe over public networks. Technical report, IBM T. J. Watson Research Center, 2001.

[6] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[7] W. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the 1997 Australian UNIX Users Group*, Brisbane, Australia, Sept. 1997.

[8] http://www.enum.org/, information request on 2nd of July 2007

[9] Large-Scale Content-Based Publish/Subscribe Systems, Ph. D. Thesis, G. Mühl, T.U. Darmstadt, 2002. http://elib.tu-darmstadt.de/diss/000274/

[10] http://www.projectliberty.org/

[11] http://www.ibm.com/developerworks/library/specification/ws-fed/

[12] EPCglobal Inc. The EPCglobal Architecture Framework, July 2005.

Building **R**adio frequency **ID**entification for the **G**lobal **E**nvironment

High level design for Discovery Services Section C: Scalability of Different Approaches

Authors: University of Cambridge, AT4 wireless, BT Research, SAP Research

15 August 2007

This work has been partly funded by the European Commission contract No: IST-2005-033546

Revision History

Version	Date	Author	Summary of Changes
0.9	26th February 2007	AT4 wireless	Scalability of different data storage implementations with comments and feedback from Cambridge
0.95	29th June 2007	AT4 wireless	Cover page updated
1.0	6th July 2007	Mark Harrison (Cambridge)	Proof-reading of document
	26th July 2007	Nicholas Pauvre (GS1 France)	Internal Bridge review
1.1	5th August 2007	AT4 wireless	Inclusion of comments from internal review

Note

The views expressed in this document are the views of the joint authors and the *Community* is not liable for any use that may be made of the information contained herein.

1. Objective

The objective of this report is to study the different options to implement the data storage of the Discovery Service from the point of view of the scalability of the system. Some other features that can be useful to evaluate are included as well.

Scalability means the ability to handle growing amounts of data and operations between them. This definition includes the capacity of the system to evolve in order to manage that growth.

2. Background

A critical component of the Discovery Service is the data storage component, which stores information about the list of EPCIS instances that have information about a particular EPC serial number. Potentially, the data storage component could include a list of EPCIS for every serial product, meaning that the amount of data to be stored by the Discovery Service can be immense.

During the design task, some options have been proposed to implement this component, i.e LDAP, DHT and Search Engines. Their characteristics should be compared with the requirements for Discovery Services, so that the most promising could be selected for the discovery service prototype.

3. Data storage alternatives

The data storage options have been classified in three architectures. First, the open hierarchy architecture represents those models which are oriented to store highly hierarchical data. Second, the peer to peer architecture is characterized by the participation of multiple nodes that collaborate in some way to store and retrieve the information. Finally the hosted service is based in the existence of a web search engine which deals with the storage and recovery of the information.

3.1. Open Hierarchy

Open hierarchy architecture includes those models that store the information in a tree. Some features of these models are:

- The content is organized into a hierarchy.
- The query operations have to pass through the root (although they can be cached at lower levels of the hierarchy).
- Sub trees of the hierarchy can be distributed between different servers.

Two options of open hierarchy architectures are considered and described in more detail, namely LDAP and DNS.

3.1.1. LDAP (Lightweight Directory Access Protocol)

LDAP is a networking protocol for querying and modifying directory services running over TCP/IP. LDAP is based on the X.500 standard, but is significantly simpler and more readily adapted to meet custom needs.

The LDAP protocol is based around the definition of a directory. A directory is a set of information with similar attributes organized in a logical and hierarchical manner. An LDAP information directory is a type of database, but it is not a relational database. Databases are usually designed to perform many changes of their data and LDAP directories are heavily optimized for read performance, so it is particularly useful for storing information that you wish to read from many locations.

LDAP servers can replicate either some or all of their data via push or pull methods, allowing dat to be pushed to remote servers, to increase security or efficiency. The replication technology is built-in and easy to configure.

Each LDAP server can hold a sub tree of the hierarchy starting from a specific entry. Servers can store references to other servers too. In this way a query to a server can result in a reference to other server (referrals) or even the server can contact the other server and return its results to the client (chaining).

LDAP allows for secure delegation of read 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.

Integration of Discovery Service data

The integration of the Discovery Service records into the data structure of the LDAP is based on the decomposition of the pure-identity URN of the EPC serialized ID into the LDAP tree. Fields of this URN like Company Prefix, Item Reference or Serial Number can be used to distribute the EPC serialized IDs among the tree structure.

Sub trees representing sets of companies (divided according to the company prefix) can be separate in different servers to distribute the system. It can use the chaining feature of LDAP to homogenise the external view of the system and return only the final list of EPCISs.

To mitigate the bottleneck of the root, some servers can store copies of the root server content using the replication option feature. Load-balancing solutions can be used to distribute the operations between the servers.

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.

3.1.2. DNS (Domain Name Service)

DNS stores and associates information with domain names, and translates domain names into IP addresses. This enables the use of easy to remember names instead of their IP addresses.

The domain name space is structured as a tree of domain names whose leaves or nodes have resource records that have information related to the domain name. The tree is divided into zones of connected nodes that depend on an authoritative DNS server or nameserver. These zones may change depending on the function of the nameservers.

A resolver is in charge of the lookup of the information associated with nodes. It sends DNS requests to communicate with the nameservers and receives DNS responses. Usually it is necessary to communicate with several servers to find the needed information.

A domain name usually consists of several parts separated by dots. In a query each part is interpreted from right to left using an iterative search procedure. At each iteration, the

program queries the corresponding DNS server to provide a pointer to the next server that has to be consulted.

This search procedure has the problem that it produces a high load in the collective root servers, since every search starts by querying one of these servers, producing a bottleneck.

A way to prevent this load in the servers is to include a cache that will store the response of the server during a given time to live (TTL). The resolver will consult this cache instead of querying the server unless the responses in the cache are not useful or have expired. With this cache mechanism, the resolver does not need to contact the server for the same information several times.

Changes in data in DNS do not always take effect immediately, so it may happen that different users can access different versions of the data at the same time until the change takes effect in all the systems.

To improve the security of the protocol, Domain Name System Security extensions (DNSSEC) add some security features to the DNS protocol, like origin authentication and integrity of data and authenticated denial of existence, but the issue is that can reveal the complete list of zone names.

Integration of Discovery Service data

The integration of the Discovery Service records into the data structure of the DNS is based on the decomposition of the pure-identity URN of the EPC serial ID into the DNS tree. Fields of this URN like Company Prefix, Item Reference or Serial Number can be used to distribute the EPC serialized IDs among the tree structure.

Sub trees representing sets of companies can be separated into different servers to distribute the system.

To mitigate the bottleneck of the root, some servers can manage copies of the root server content using replication features. Load-balancing solutions can be used to distribute the operations between the servers.

To increase security, the DNS Security extensions (DNSSEC) can be used to integrate security characteristics to the system, but some features like fine grained access control still have to be developed.

3.2. Peer to Peer

This architecture facilitates the collaboration between the nodes to store and retrieve the information. Below is a list of some features of this architecture:

- Organization of the nodes can change dynamically to accommodate variations in the number of nodes and improve the performance and scalability of the system.
- Permits multiple starting points to make operations with the data.

DHT architecture is considered and described in more detail.

3.2.1. DHT (Distributed Hash Tables)

Distributed Hash Tables are decentralized distributed systems that distribute the management of a key table between the participating nodes. Each node will maintain a routing table with a list of its neighbours, 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 have continuous node arrivals and failures by constructing a structured overlay network.

The key is usually a 160-bit key that can be obtained from a SHA1 hash of more complex data like filenames. The typical operations are put(key, data) to store data into the DHT and get(key) to recover data. These operations can be sent to any node of the DHT structure and it will propagate that operation to the appropriate node.

One of the characteristics of this model is the 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 the nodes are organized in a way that a node only needs to coordinate with a few other nodes in the system (usually $\Theta(\log n)$ 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 DHT systems can implement replication features to avoid that the failure of a node, which would make it impossible to access a portion of the data stored by the system. To do so, each portion of data can be replicated in some cluster nodes, and the system is in charge of maintaining the coherence of the replicas in a dynamic way: When a node joins the cluster, it can store replicas of some partitions of the distributed hash table.

DHT systems implement features like load balancing by distributing the operations among a cluster of nodes, ensure data integrity by keeping up-to-date replicas of each portion of data and maintain performance optimizing the management of the data stored in the system.

Integration of Discovery Service data

The integration of Discovery Service records into the data structure of the DHT is based on the use of the pure-identity URN of the EPC serialized ID as the key for the Hash table. The DHT algorithms deal with the partition of the key space between the nodes.

If the hash function selected to generate the key space returns the same hash key for two different URN codes, there will be false positives results for queries on these URN codes.

There is no bottleneck in the system because any operation can be initiated in any node, and the failure of one node can be mitigated with the existence of replications of its data in other nodes.

Security options like authentication or fine grained access control have to be implemented into the system.

3.3. Hosted Service

The hosted service architecture uses functionalities of search engines to manage the data to be stored. Some characteristics of this architecture are:

- The number of attributes to be indexed can be increased easily.
- User can use filters with combinations of attributes of the nodes in the search.
- Search engine can cache EPCIS data to optimize the response time.

Search Engine architecture is considered and described in more detail

3.3.1. Search Engine

A search engine is an information retrieval system prepared to find information stored in a particular system. The search is produced when it is asked for data that contains or is related

to specific criteria. The search produces a list of results that match the criteria. It is usually sorted according to a measure of how much the results are related to the criteria.

Search engines store information about the entries of data. All this information is analyzed to know how that entry must be indexed in a database that will be used in search operations. Internally, the most used structure to store the information is the inverted index. The inverted index is a structure that stores a mapping from words to their locations in a document or a set of documents, allowing full text search over the document. When a query is done over a set of words, the list of results is the intersection of documents that appears in the inverted index for each searched word. This means that the model is optimized to respond to search queries that include combinations of the record fields.

When a user of a search engine makes a query, usually giving a keyword, the engine looks for it in the index database to produce the results. The utility of the search engine is measured with the relevance of the results it gives back.

Integration of Discovery Service data

The integration of the Discovery Service records into the data structure of the search engine is based on the indexing of each record. When the EPCIS submits a new record, the search engine stores the index of the pure-identity URN of the EPC serialized ID.

A query on the pure-identity URN will return the URL of the EPCIS instances that have registered events about it. To use other attributes in the query it is only needed to index them in the search engine.

The bottleneck caused by the single entry point to the search engine can be mitigated by replicating it (with clusters, for example). Load-balancing solutions can be used to distribute the operations between the servers.

Security options like authentication or fine grained access control have to be implemented into the system.

4. Comparison

This section describes some criteria used in the comparison of the different implementations for the data storage component of a Discovery Service.

4.1. Criteria

The criteria used have been grouped into three categories: Scalability criteria, data integration criteria and other criteria that has been considered relevant. The description of the criteria used is shown below:

Scalability

- Horizontal scalability: This feature deals with how easy is to add new servers and their impact on the performance of the system.
- Bottleneck: This feature describes the existence of bottlenecks and ways to bypass them.
- Data Update: This characteristic shows the capability of the model to carry out multiple change operations (add, update or delete).

- Data Search: this property depicts the capability of the model to carry out multiple search operations.

Data integration

- Organization of data: This characteristic describes how the data is organized into the model. It can be useful to understand the potential bottlenecks of the system and the options to distribute it.
- Record with fields: This characteristic describes the capacity of the model to create records of fields to represent each Discovery Service Record.

Other

- Guarantee of result correctness: This property of the models depicts the potential false positives that can be obtained from the model.
- Access control: This characteristic shows the access control features implemented by each model.

4.2. Comparison table

Scalability

	LDAP	DNS	DHT	Search Engine
Horizontal Scalability	Tree architecture can be extended by delegating sub trees to new servers. Multiple lookup processes can be done in parallel by different servers.	Tree architecture can be extended by delegating sub trees to new servers. Multiple lookup processes can be done in parallel by different servers. Clients use cache to improve performance.	System is prepared to work with nodes continuously leaving or joining to the space. Multiple lookup processes can be routed in parallel and can begin from different nodes	System is prepared to handle many queries in parallel. Inverted index can be divided across multiple servers.
Bottleneck	There is a bottleneck in the root of the architecture (but can be replicated).	There is a bottleneck in the root that can be mitigated by replicating the root and using caches.	There is some coordination to support the structure but the failure of one node does not affect the whole system.	There is a bottleneck in the point of access to the search engine (but can be replicated).

	LDAP	DNS	DHT	Search Engine
Data Update	It is not optimized for massive update operations.	It is not optimized for massive update operations. When cache is used the visibility of changes can be delayed.	It is optimized for massive search and update operations.	It must update its index database when new data joins the system.
Data Search	Search is done in a hierarchical manner among the tree directory. It is optimized to make massive search operations.	Search is done in a hierarchical manner among the tree directory. It is optimized to make massive search operations using caches.	It is optimized for massive search and update operations.	It is optimized for massive search operations.

Data integration

	LDAP	DNS	DHT	Search Engine
Organization of data	Data will be distributed hierarchically in the nodes of the tree.	Data will be distributed hierarchically in the nodes of the tree.	Uniform distributed nodes among a key space. With no central coordination.	Data is stored internally in the inverted index structure.
Record with fields	Each entry consists of a set of attributes.	Each entry consists of a set of attributes.	Each entry can consist of any information.	Each entry can consist of any information.

Other

	LDAP	DNS	DHT	Search Engine
Guarantee of result correctness	There is no possibility of incorrect lookups.	There is no possibility of incorrect lookups.	It can return incorrect results if two EPC serial numbers produce the same hash code.	It will depend on the keyword that users give to make a search.

	LDAP	DNS	DHT	Search Engine
Access control	Control access is already implemented by Access Control Lists (ACLs) at node level.	There is some implementation of Access Control Lists (ACLs) at any level for adding, updating and deleting entries, but not for searching.	Easy to provide at least general access to the system. Fine grained controls have to be implemented	Easy to provide at least general access to the system. Fine grained control have to be implemented

5. Conclusion

This document describes some models to implement the data storage component of the Discovery Service. The models have been classified in three architectures, Open Hierarchy, Peer to Peer and Hosted Service to group models with similar characteristics. It describes briefly each model and a strategy to integrate it with the Discovery Service data.

The comparison of the models has been made based on some features that have been previously detailed. Finally the comparison table is presented with the conformity of the models about each criterion.

The comparison table reflects the level of compliance of each model with the criteria identified. The final selection of the best implementation will depend on the importance given to each criterion, but if we consider only the scalability and data integration criteria, the DHT and LDAP are the more suitable options to implement the data storage component.