Volume 51 Issue 16 14 November 2007 ISSN 1389-1286

Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 4529–4542
Computer Networks
www.elsevier.com/locate/comnet

mRDP: An HTTP-based lightweight semantic discovery protocol

Juan Ignacio Vazquez *, Diego López-de-Ipiña

Faculty of Engineering, University of Deusto, Avda. Universidades, 24, Bilbao 48007, Spain

Available online 24 June 2007

Abstract

Discovery is one of the most important activities in ubiquitous and distributed computing, with a plethora of available protocols. Most of these protocols are designed for one concrete purpose: network nodes discovery, service discovery, search of specific information stored through the network, and so forth.

Designing a single discovery system able to deal with the particularities of many different information structures and purposes seems not feasible. Moreover, these data structures possess some underlaying meanings and relationships that are usually hidden from traditional discovery protocols that use simple text-based matchmaking.

A semantic discovery protocol could solve this problem by taking advantage of semantically annotated data and performing reasoning over the information to obtain additional knowledge that can be crucial in processing the queries.

In this paper, we describe the basics of a novel semantic discovery mechanism called mRDP (Multicast Resource Discovery Protocol) built upon HTTP and Semantic Web technology to provide more powerful discovery capabilities.

Keywords: Web; HTTP; Discovery; Semantic; Protocol

1. Introduction

Discovery has always been a hot topic in networking, generally constituting an initial step to obtain information about available entities in order to perform further interactions. This issue becomes specially important in ubiquitous computing systems or mobile ad-hoc networks, where existing entities must be continuously aware of each other, and adapt the network topology as required.

Although several different discovery protocols have been proposed and used over the last years in concrete architectures, mainly for device and service discovery, there is no common agreement about a unified discovery protocol.

Edwards [9] defines discovery in ubiquitous computing systems as "*a mechanism for dynamically referencing a resource on the network*". Devices and resources come and go on a highly dynamic basis, thus McGrath highlights "spontaneity" and "automatic adaptation" as some of the most important features of the discovery process [18].

* Corresponding author.
E-mail addresses: ivazquez@eside.deusto.es (J.I. Vazquez), dipina@eside.deusto.es (D. López-de-Ipiña).
URLs: http://paginaspersonales.deusto.es/ivazquez (J.I. Vazquez), http://paginaspersonales.deusto.es/dipina (D. López-de-Ipiña).

1389-1286/$ - see front matter
doi:10.1016/j.comnet.2007.06.017

Ubiquitous computing researchers are particularly sensitive to discovery protocols, since an important part of the expected intelligence in the environment is meant to be provided during this initial phase: if the user's PDA is not able to find appropriate devices to communicate with in the environment, it will not be able to provide the required services to the user.

Although this goal may not seem very difficult to achieve – in fact several simple and lightweight discovery protocols have been used in the past – providing a higher level of intelligence during discovery is much more complicated.

For instance, an entity may issue a search request for "devices *inside* a cupboard", "computers *near* the TV" or "digital photo frames with pictures authored *by a friend of* mine".

The above examples are not especially difficult to implement in any ubiquitous computing architecture conveniently prepared to cope with them, especially the meaning of the highlighted terms: if the information is structured in a uniform way (e.g., XML) and stored somewhere, or maybe distributed across the different entities in the network, a query can be created and disseminated to obtain the results.

However, a more difficult approach consists in the possibility of creating a discovery system able to manage any kind of *unexpected* query, dealing with concepts, vocabularies and relationships among the terms that are unknown at the time of system design.

For instance, let's consider the following scenario:

- Objects are tagged in such a way that a containing object (e.g., a wardrobe, or a backpack) is able to know the identity of the objects directly placed inside. An example of such system is formed by objects tagged with barcodes and barcode readers on the containers.
- A PDA is stored in a backpack.
- The backpack is placed in a room.

If a subject is provided with this information and asked to identify available PDAs in the room he will surely point at the PDA inside the backpack as one of the items: location is a transitive property, which means that if the PDA is in the backpack, which in turn is placed in the room, the PDA is located in the room.

However, computing systems are not able to interpret information at as fine a level as humans do; they do not know about the properties of the location concept. Unless explicitly programmed to do so, "location" is a word like any other such as "point to": the PDA is pointing to the backpack, which in turn is pointing to the room, does not imply that the PDA is pointing to the room.

An intelligent discovery mechanism must provide a higher level of context interpretation and knowledge than traditional text pattern matching schemes: it dives through the information relationships, understanding their implications. Fortunately, there is one technology able to provide the required framework for annotating data and relationships and creating knowledge that can be used in the discovery process: the Semantic Web.

The Semantic Web is particularly interesting for discovery because of its *future-proof* characteristics: it can provide a solution framework for "*problems and situations yet to be defined*" [15]. Therefore, it can provide the mechanism to support the creation of undefined search queries on concepts unknown at the moment of design, exploring the connections of information structures to provide a much deeper knowledge, and more refined search results.

This paper presents mRDP, a discovery protocol based on semantic queries. In Section 2 a number of discovery protocols and architectures are mentioned along with their limitations. Section 3 provides an introduction to semantic queries. Section 4 introduces the basics of mRDP – Multicast Resource Discovery Protocol, a lightweight protocol for semantic discovery in local area networks. Sections 5 and 6 describe the format of queries and protocol messages, respectively. In Section 7 measures of mRDP performance are included. Finally, the conclusions and a brief discussion about future lines of semantic discovery are provided.

2. Related work

Different discovery protocols such as SLP [13], Jini [26], UPnP SSDP [12] and other alternatives [9] have been widely used in the past, and their limitations have been clearly identified [6]:

- Lack of rich representation: the existing architectures lack of expressive languages, representation and tools for the broad range of service descriptions.
- Lack of constraint specification and inexact matching: most protocols require exact matching, with a simplistic notion of constraints. Lack of semantic matching.

- Lack of ontology support for representing service descriptions and capabilities.

For example, Jini matching is performed by comparing interface names, not functionality, there is no reasoning mechanism and it uses absolute matching (e.g.: printer name and model, instead of "the nearest printer").

mDNS [8], DNS-SD [7] and Bonjour (a variant of DNS-SD) [1] are other candidate technologies, but neither of them embraces the potential of semantic mark-up. There are also other novel approaches, such as Cooltown [14] which used "URL sensing" for discovery, based on IR/RF, barcodes, electronic tags or optical recognition. López de Ipiña et al. carried out a similar approach in EMI^2 [17] by encoding tiny URLs in TRIP tags [16] for device identification.

Other systems have already addressed the importance of discovery and the suitability of Semantic Web technologies to help here [21] [20] [22]. In all cases, the activity that embodies the essence of the discovery process is matchmaking: how the query is resolved to identify suitable candidates.

Virtually all existing discovery protocols base the matchmaking process in the device or service type, the ID or some concrete attribute values. This approach performs relatively well for simple systems, but as McGrath points out, "*keyword or string matching is relatively easy and efficient to implement [...]. String matching performs well in limited cases: essentially when the vocabulary is controlled*" [19].

Simple discovery protocols provide basic attribute-value matching mechanisms, where service type or ID are considered as attributes. The expressive capability of such systems is thus restricted at a syntax level.

The need for more intelligent discovery capabilities is again stressed in [34]: "*service discovery protocols must work in unfamiliar computing environments [...] Service discovery design for such environments is more challenging [...] This means that service discovery protocols and the underlying computing infrastructure must have more intelligence*".

As previously mentioned, Semantic Web technologies overcome the above limitations by describing information at a higher level, and it is specially suitable for unknown domains [15].

Semantic discovery seems very promising but there are not many experiences with systems implementing this model ("*although RDF has been proposed as the service description format for interoperability between service discovery systems [23], so far there is no service description standard yet*" [33]).

During the last two years we have been designing an advanced ubiquitous computing architecture based on the Web communication model and Semantic Web technologies: SoaM – Smart Objects Awareness and Adaptation Model [27]. Since existing discovery protocols did not provide the required level of intelligence, as already explained, we designed a lightweight semantic discovery protocol for our architecture: mRDP – Multicast Resource Discovery Protocol.

3. An introduction to semantic queries

During the last years the whole Web model has been undergoing an evolution towards a new paradigm called "the Semantic Web" [3]. The basics of the Semantic Web were outlined by Berners-Lee et al. in [5].

Basically the Semantic Web is a Web of knowledge, where concepts and information are represented in a machine readable and understandable form and linked via URIs. Every concept (people, places, objects, time events, verbs, and so forth) can be identified via a unique URI, in such a way that a universe of concepts can be related to each other. RDF (Resource Description Framework) [29] is the Semantic Web mechanism for creating information graphs with arcs representing the relationships among concepts. RDF graphs can also be transformed into an XML-based serialised form, more suitable for transmission.

More interestingly, the Semantic Web does not only provide a mechanism for information representation but also for reasoning. OWL (Web Ontology Language) [28] is an example of a Semantic Web technology that can be used to represent "description logic" [2] formalisms in such a way that new information can be automatically generated by applying intrinsic reasoning mechanisms.

In fact, OWL provides a means for creating transitive properties such as "located in", symmetric properties such as "near", inverse relationships such as that established between "located in" and "contains", and so forth. Using OWL it is possible to define concepts, features and relationships for any knowledge domain.

As already mentioned, a major advantage of using RDF/OWL instead of basic attribute-value matching is that semantic processing can be

performed to resolve the query. For instance, let's consider the following scenario:

A tool manufacturing firm designs an intelligent drill that is able to automatically switch itself off if the user is drilling metal in the presence of inflammable substances. The drill periodically searches the environment for inflammable materials, issuing a warning to the user via the built-in display.

The environment and the objects within, including the drill, provide semantic information which is stored in a collective knowledge repository. For instance (using Notation 3 [4]):

```
<urn:uuid:drilll> loc:locatedIn <urn:uuid:rooml>.
<urn:uuid:drilll> task:drilling <urn:uuid:surfacel>.
<urn:uuid:surfacel> rdf:type fur:SteelShelf.
<urn:uuid:boxl> loc:contains <urn:uuid:fuell>.
<urn:uuid:boxl> loc:locatedIn <urn:uuid:rooml>.
<urn:uuid:fuell> rdf:type fuel:ChemicalFuel.
```

Existing Ontologies in the repository provide location relationships, based on <loc:locatedIn> (transitive) and <loc:contains> (transitive and inverse of <loc:locatedIn>), as well as the following additional information:

```
fur:SteelShelf rdf:subtype mat:MetallicThing.
fuel:ChemicalFuel rdf:subtype mat:InflammableThing.
```

The query "*identify inflammable material in the location where I am drilling metal*" issued periodically by the drill can be represented as:

```
IDENTIFY ?material WHERE
   ?material loc:locatedIn ?room.
   ?material rdf:type mat:InflammableThing.
   <urn:uuid:drilll> loc:locatedIn ?room.
   <urn:uuid:drilll> task:drilling ?sur.
   ?sur rdf:type mat:MetallicThing.
```

A strict matching of the previous query against existing context information in order to resolve the variables would not yield a result: there is no evidence of the nature of materials and location relationships without applying ontological reasoning. In this case, the drill would not detect the dangerous situation; the DRILL would not be considered intelligent.

However, the application of transitive and inverse relationships contained in the ontologies would generate the following new facts:

```
<urn:uuid:fuell> rdf:type mat:InflammableThing.
<urn:uuid:surfacel> rdf:type mat:MetallicThing.
<urn:uuid:fuell> loc:locatedIn <urn:uuid:boxl>.
<urn:uuid:fuell> loc:locatedIn <urn:uuid:rooml>.
```

The first and second facts are obtained through the subtype relationships provided by the existing ontology about materials. The third fact is obtained via the "inverse" relationship among <loc:contains> and <loc:locatedIn> in the location ontology (if boxl contains fuell, then fuell is located in boxl). Finally the fourth fact is generated by the transitive nature of <loc:locatedIn> (if fuell is located in boxl, which in turn is located in rooml, then fuell is located in rooml).

Semantic reasoning prior to query execution augments the available knowledge base to successfully resolve the query and obtain the selected object (<urn:uuid:fuell>). The drill is now able to display a warning message to the user. In fact, the message could contain a more descriptive label for the inflammable material that could also be found in the collective knowledge repository by issuing this query:

```
IDENTIFY ?label WHERE
   <urn:uuid:fuell> rdfs:label ?label.
```

Distributed and ubiquitous computing environments are composed of a variety of digital objects managing different parts of the collective knowledge. A semantic discovery protocol must be able to disseminate queries through the network in such a way that they reach every object, where semantic processing takes place and results are sent back. The whole process results in a distributed discovery mechanism with intrinsic intelligence.

4. Semantic discovery with mRDP

Both as an important part of our architecture and as a broader goal itself we established a number of requirements for a semantic-powered discovery protocol:

- In order to achieve spontaneity, no central registration server must be used. Thus, multicast communication will be applied.
- Since some devices can feature limited processing capabilities, two modes of operation must be provided: full semantic-powered operation, and basic operation.
- It must feature a highly expressive language for the queries.
- It must be able not only to provide searching capabilities about resources, but also to obtain

URLs where more information about them can be downloaded.
- It must be based on the TCP/IP stack with the highest possible integration with HTTP (thus, taking advantage of the Web architecture and HTTP security mechanisms).

The resulting design is mRDP – Multicast Resource Discovery Protocol, a UDP/HTTP-based protocol for semantic-powered queries, providing two different functions:

- Resource identification: search the network for resources meeting particular conditions. For example, "find all the devices located in `urn:uuid:room21`".
- Resource description location: search the network for sources of information about a particular resource. For example, "where can I obtain information about `urn:uuid:fuell`?".

While resource identification has already been visited in the previous example, description location is a new feature that enables an mRDP client to locate information providers about a particular resource, given its URI.

For example, an mRDP client may want to know all the information related to `http://people.com/bobby` in the network, or maybe all the data related to `urn:uuid:fuell` stored by any entity. The mRDP client would issue a location request about the URI, and every mRDP server containing some piece of information about that resource would reply with a URL where that information can be downloaded.

Attaching just the URL is more efficient than directly sending the whole volume of data in the reply for several reasons:

- It generates less traffic, since packets are significantly smaller than conveying all the information.
- The decision about eventually downloading the data is left to the client, which carries out the download if needed (maybe the required information has already been downloaded from another source, so there is no need to retrieve it again).
- The URL can be reused several times if the client needs to poll the data about the resource periodically, without searching the network again.

These two functions of mRDP, despite being independent, can be coordinated in a complementary process: the mRDP client obtains the identification of resources meeting some conditions, and afterwards locates the information sources where more data about those resources can be downloaded.

4.1. Operation

The mRDP architecture declares two types of agents: mRDP clients and mRDP servers. Generally, both agents coexist within the same entity, querying and providing information, promoting a P2P architecture in the network.

During resource identification, (1) an mRDP client disseminates the query through the network about a particular resource matching some concrete conditions. Every mRDP server receives the query, processes it against its information model, that is, the RDF graph, and (2) returns the results with the URIs of the resources matching the query back to the client.

During resource description location, (3) an mRDP client disseminates a request about the resource throughout the network. Every mRDP server checks whether there is information available about that resource in its RDF graph and, if so, (4) it returns one or more URLs to the client, where data about the resource can be downloaded.

The interactions illustrating both functions are depicted in Fig. 1. The client disseminates a query `q` about a resource to the network, where four mRDP servers are available, each of them with an RDF graph representing the managed information. Two of them, `B` and `C`, can resolve the query successfully and provide the identification of two resources satisfying the query: `B` provides `r1` and `C` provides `r2`. Next, the client disseminates a request to retrieve all the available information about `r1` in the network. Not only `B` replies providing at least one URL where all its stored information about `r1` can be downloaded, but `D` also replies, since it stores some pieces of information concerning `r1` (although this data was not enough to match the original query `q`).

In order to save traffic and time, mRDP servers can also provide resource description location URLs in the response to the identification message (2), so the client does not need to perform a second interaction except in the case where more information from other sources is needed.

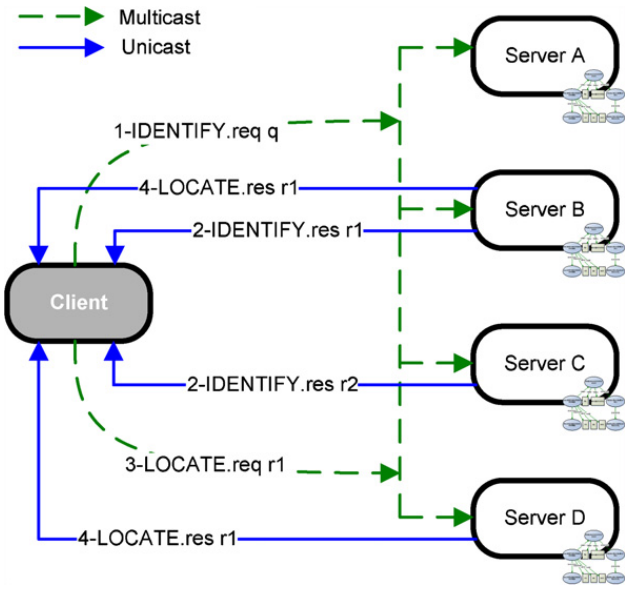

Fig. 1. Example of mRDP operation.

While client queries are always multicast, so they can use UDP as the transport protocol, the server responses are always unicast. These responses are not limited in size, and in fact, they can be very large (e.g., in a scenario where an mRDP server stores information about dozens of users, and the mRDP client disseminates a request to identify resources of type "user").

Queries can be conveyed in UDP messages since they are small, but responses could be larger and require fragmentation or "chunking" if embodied in UDP datagrams. Moreover, in order to cope with network reliability, the client can issue several copies of the query, but if servers have to deal with UDP unreliability in every response, the complexity of the protocol increases unnecessarily.

Moreover, while queries must be readable by any server in order to check whether they can provide the desired information, responses could require a higher level of privacy and security. Attaching the appropriate security mechanisms to the resulting protocol would even create a more cumbersome communication scheme.

Fortunately there is another solution to the problem, backed up by existing and well-proven techniques and better integrated into our model: the client could attach a callback HTTP URI to the request, so that every mRDP server can send the response to this endpoint, using traditional HTTP communication, augmented with authentication mechanisms (basic or digest), or even HTTPS.

Some advantages of sending the response back to the client using HTTP are:

- Since HTTP is used in other parts of the our architecture, reuse of libraries and minimisation of platform size is achieved by this strategy.
- HTTP Basic or Digest Authentication as well as HTTPS can be easily reused and implemented, thus taking advantage of existing standards.
- Since HTTP uses TCP as transport layer, reliability is intrinsically provided and messages are not limited in size.

Therefore, the mRDP communication architecture will use UDP multicast messages for sending the requests and HTTP callbacks for receiving the responses as illustrated in Fig. 2.

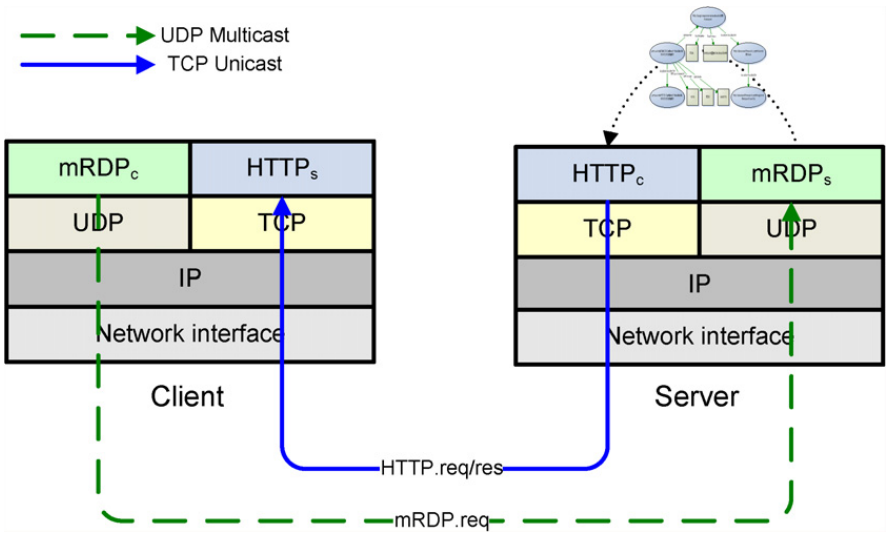

Fig. 2. mRDP over the TCP/IP protocol stack.

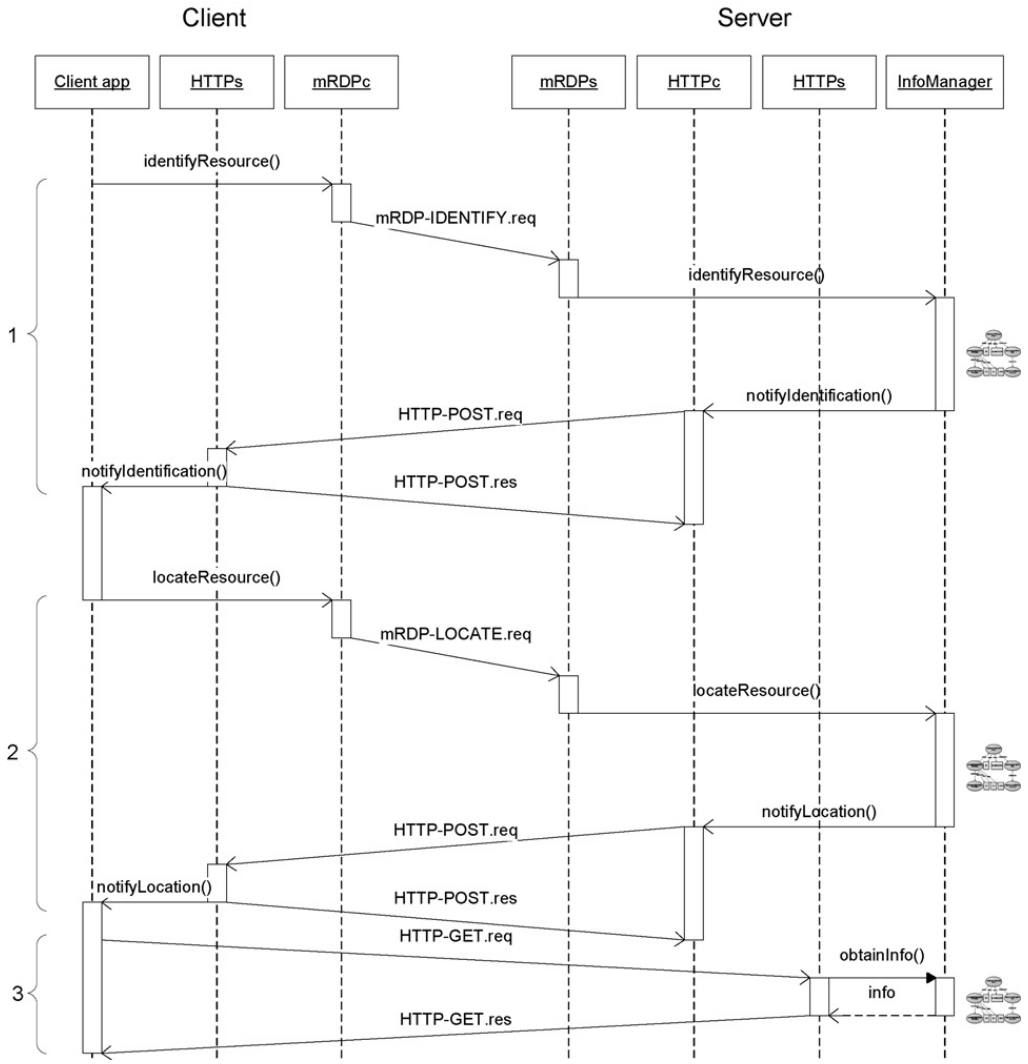

Fig. 3. mRDP UML sequence diagram with all the interactions.

A UML sequence diagram of the communication process is depicted in Fig. 3. The steps involved are:

1. A client application requests the mRDP client module to issue a request to the network conveying a certain query to find matching resources. The mRDP server module on the server side receives the query and processes it against the internal information model. The response is generated and sent back to the client in an HTTP POST request, where the HTTP server module extracts the data and passes it to the client application.
2. If the client needs to find more information for a concrete resource, it requests the mRDP client module to issue a description location request to the network. Receiving servers managing information about the queried resource generate an HTTP POST callback, containing their references for accessing that resource data, to the client HTTP server, which in turn passes them to the client application.
3. Finally, the client application may use the references to retrieve the desired resource information from the HTTP server at the server, via a simple HTTP GET operation.

Step 2 can be avoided if the response to the identification request in step 1 already contains references provided by the server to download resource information, thus generating the sequence depicted in Fig. 4.

Both mechanisms are complementary, since the server can include its references for retrieving resource information, while the client can issue an mRDP description location request to the network for obtaining more references from other servers that also manage information about the queried resource.

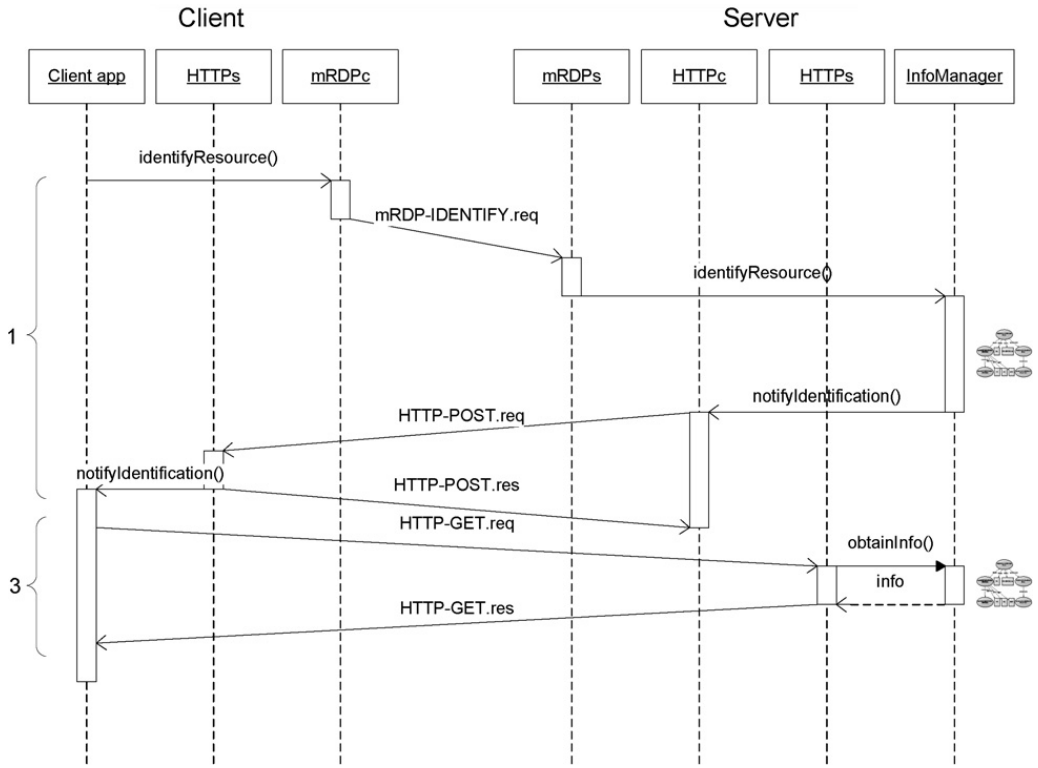

Fig. 4. Optimised mRDP UML sequence diagram.

4.2. Resource identification

The purpose of resource identification request messages is to convey some kind of query, so that resources matching the query are identified. Different query languages can be used for this purpose, but SPARQL [32] is probably the best positioned candidate: SPARQL syntax provides rich levels of expressiveness for constructing conditions about resources in a RDF graph.

Obviously the main problem with SPARQL is that it requires a SPARQL engine in every mRDP server, which is not generally possible in limited devices. Moreover, SPARQL provides a lot of additional constructions and keywords that are neither required nor useful for resource identif-i cation.

A simpler alternative to SPARQL must be provided, such that limited devices can interpret and process the queries, and generate replies when acting as mRDP servers.

5. Plant: pattern language for N-Triples

The most basic standardised RDF notation available is N-Triples [30]. N-Triples features a line-based, plain-text format and very simple grammar, a subset of Notation 3 [4], for encoding an RDF graph.

An example document representing RDF information in N-Triples format is:[1]

```
<urn:uuid:tvl>  <http://www.awareit.com/onto/2005/12/
  tvchannel> <http://www.bbc.co.uk/bbctwo>.
<urn:uuid:lightl>  <http://www.awareit.com/onto/2005/
  12/lightluminance> "4".
```

No prefixes but absolute URIs must be used in N-Triples for identifying resources. Basically the N-Triples notation is the direct serialisation of every triple in the RDF graph. N-Triples allows typed literals and blank nodes. N-Triples's simplicity makes it the best candidate for limited devices when dealing with RDF information.

However, there is an important drawback in N-Triples notation for our purpose: it can only express concrete RDF triples, neither patterns nor conditions. In order to support these kind of constructions, the N-Triples syntax must be extended to be able to produce expressions such as:

[1] Apparent line breaks in the code are due to limitations in page width.

Listing 1. Example of triple patterns based on N-Triples.

```
1. ?r <http://www.w3.org/1999/02/22-rdf-syntax-nstype>
   <http://www.awareit.com/onto/2005/12/tvTV>.
2. ?r <http://www.awareit.com/onto/2005/12/tvchannel>
   <http://www.bbc.co.uk/bbctwo>.
```

The above expression represents a query to identify a resource `?r` meeting two statement patterns (conditions): `?r`'s type is TV and its current configured channel is BBC2.

In order to be able to build this kind of expression with variables, representing RDF triple patterns, the N-Triples grammar must be extended. Therefore, we modified the `subject` and `object` productions that were originally defined in the N-Triples specification [30] in order to support variable constructs via a new `variable` production.

```
subject ::= uriref | nodeID | variable
object ::= uriref | nodeID | literal | variable
variable ::= '?' name
```

Now, triple patterns such as those illustrated in Listing 1 can be supported by this simple extension we have called Plant (Pattern Language for N-Triples).

Just as N-Triples represents a subset of Notation 3 expressiveness, Plant represents a subset of SPARQL expressiveness. If fact, the example of Listing 1 can be translated into SPARQL as:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX tv: <http://www.awareit.com/onto/2005/12/tv>
SELECT ?r
WHERE {?r rdf:type tv:TV.
       ?r tv:channel <http://www.bbc.co.uk/bbctwo>.
       }
```

However, Plant's simplicity makes it especially suitable for being processed by resource-limited devices without demanding high computing requirements. If the receiving server is able to process SPARQL queries, it can transform the Plant query into a SPARQL representation as illustrated above. If the server does not feature a SPARQL engine, it can use the Plant Query Resolution Algorithm.

The Plant Query Resolution Algorithm is a very simple and straightforward variables unification algorithm optimised for dealing with RDF information and resolve Plant queries, although another query syntax could be used.

The algorithm is structured in the following steps:

1. Optionally, map the attribute-value pairs into RDF triples (if the information is not already in RDF form).
2. Identify all the variables in the whole set of Plant patterns.
3. For every Plant pattern, select the triples in the information model that meet such a pattern and annotate the combination of valid values for the variables provided by the triple (* for any value).
4. Substitute * by the available values of the involved variable in combinations present in other Plant patterns.
5. Identify the combinations of values that match all the patterns: these combinations represent the solutions for the variables. If no combination is identified, the query cannot be resolved.

This algorithm is easy to implement in limited devices and provides the required compatibility with our semantic-powered discovery mechanism.

5.1. mRDP SPARQL queries

In addition to Plant, SPARQL queries are also supported in mRDP as illustrated further in Listing 3. Only `SELECT` constructions are allowed in mRDP SPARQL queries since they are the only ones that can produce bounded variables with values (`CONSTRUCT` and `DESCRIBE` return a graph, and `ASK` returns a boolean).

Since a SPARQL `SELECT` query can include a number of variables, the one involved in the mRDP resolution is that referred to in the request line of the protocol (see Section 6).

SPARQL queries are much more powerful than Plant queries. They can express conditions and filters with logical connectives and a broad range of operators. The main drawback of using SPARQL is that resource-limited devices will not be able to process SPARQL queries; but this is not a concern in traditional computer-based networks.

6. mRDP message format

mRDP request messages typically convey a Plant query over UDP and are multicasted to the address 224.0.24.1 and UDP port 2773. The MIME type

[11] for Plant queries is `application/com.awareit.plant`.

mRDP request messages follow an multi-line syntax, being extensible through new headers to provide additional semantics. As most Internet protocols, ASCII characters 13 and 10 (CR LF) are used to separate lines which constitute the major divisions in the message format.

An mRDP request message is composed of a request line, zero or more headers and an optional body section. The request line's main element is the command, either an `IDENTIFY` command followed by the variable to resolve in the query or a `LOCATE` command with a resource URI. The request line ends with a protocol version number.

Headers follow the same format as traditional headers in protocols such as HTTP [10], also used in RTSP [25] or SIP [24]. There are four headers defined in mRDP 1.0:

- `NSeq`: request sequence number for detecting duplicate request messages from clients and pairing requests and responses.
- `Content-Type`: MIME type of the message body section. Currently, mRDP supports two MIME types: `application/com.awareit.plant` for Plant queries, and `application/sparql-query` for SPARQL queries. This header must not appear if the message does not contain a body section (such as `LOCATE` messages).
- `Content-Length`: length in bytes of the message body section. This header must not appear if the message does not contain a body section (such as `LOCATE` messages).
- `Callback-URI`: the endpoint where the server must send the response back to the client. The `type` parameter can give information about the interface type used by this endpoint in the form of a namespace URI. Several callback URIs can be provided by the client in this header, with the same or different interface types, in order to allow the server to build the response message in the most suitable format. If no interface type is provided, the default is http://www.awareit.com/soam/2006/04/redelwshttpPost, which is the only interface type for mRDP callbacks defined at the moment.

`IDENTIFY` messages must contain a body section where the query, either in Plant or SPARQL is conveyed, while `LOCATE` messages must not contain a body.

An example of message conveying a Plant query to "find inflammable material" as depicted in Section 3 is:

Listing 2. An example mRDP resource identification message.

```
1. IDENTIFY ?material mRDP/1.0
2. NSeq: 23
3. Content-Type: application/com.awareit.plant
4. Content-Length: 483
5. Callback-URI: http://169.254.0.3/mrdpendpoint
6.
7. ?material <http://www.awareit.com/onto/2005/12/locationlocatedIn> ?room.
8. ?material <http://www.w3.org/1999/02/22-rdf-syntax-nstype>       <http://www.awareit.com/onto/2005/12/materialInflammableThing>.
9. <urn:uuid:drilll> <http://www.awareit.com/onto/2005/12/locationlocatedIn> ?room.
10. <urn:uuid:drilll> <http://www.awareit.com/onto/2005/12/tasksdrilling>. ?sur
11.   <http://www.w3.org/1999/02/22-rdf-syntax-nstype> <http://www.awareit.com/onto/2005/12/materialMetallicThing>.
```

The same message using SPARQL for the query:

Listing 3. An example mRDP resource identification message with SPARQL.

```
1. IDENTIFY ?material mRDP/1.0
2. NSeq: 23
3. Content-Type: application/sparql-query
4. Content-Length: 475
5. Callback-URI: http://169.254.0.3/mrdpendpoint
6.
7. PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
8. PREFIX loc: <http://www.awareit.com/onto/2005/12/location>
9. PREFIX mat: <http://www.awareit.com/onto/2005/12/material>
10. PREFIX task: <http://www.awareit.com/onto/2005/12/tasks>
11. SELECT ?material
12. WHERE {?material loc:locatedIn ?room.
13.   ?material rdf:type mat:InflammableThing.
14.   <urn:uuid:drilll> loc:locatedIn ?room.
15.   <urn:uuid:drilll> task:drilling ?sur.
16.   ?sur rdf:type mat:MetallicThing.
17.    }
```

Listing 4 is an example of a `LOCATE` message multicasted to the network in order to obtain information sources where more data about `urn:uuid:fuell` can be downloaded. Two possible callback endpoints are provided, one of them with

the default interface explicitly declared (though this is not required):

Listing 4. An example mRDP resource descriptions location message.

```
1. LOCATE urn:uuid:fuell mRDP/1.0
2. NSeq: 57
3. Callback-URI:http://169.254.0.3 /mrdpendpoint,
   http://169.254.0.3:8081/altmrdpep;type=http://www.
   awareit.com/soam/2006/04/redelwshttpPost
```

Due to the unreliable nature of UDP, mRDP request messages must be sent three times in order to increase the probability of reaching the possible destinations in the case some UDP packet is lost. Three times for every message is a good balance between increasing reliability without generating much network traffic (other discovery protocols such as SSDP also apply this strategy).

6.1. ReDEL: resource description endpoints language

After resolving the variables in the resource identification message or locating the information sources in the resource description location message, the mRDP server constructs the reply, which is an HTTP request to the callback URI provided by the client and conveying the required data in a suitable format.

The client receives the replies via the provided `Callback-URI`, processing them afterwards. The replies themselves are very simple, they just need to express the resources matching the query and how to access available descriptions. The format should be straightforward, so that limited clients can process it without much overload.

We have designed a simple mark-up language called ReDEL – Resource Description Endpoints Language, for annotating the information contained in the replies to `IDENTIFY` and `LOCATE` mRDP messages. A possible HTTP callback conveying a ReDEL response to the query shown in Listing 2 is the following:

Listing 5. An example of HTTP callback conveying ReDEL payload.

```
1. POST /mrdpendpoint HTTP/1.0
2. Host: 169.254.0.3
3. NSeq: 23
4. Content-Type: application/com.awareit.redel+xml
5. Content-Length: 583
6.
7. <?xml version=‘‘1.0’’ encoding=‘‘UTF-8"?>
8. <redel xmlns=‘‘http://www.awareit.com/soam/2006/04/
   redel"
9.     xmlns:xsi=“http://www.w3.org    /2001/XMLSchema-
   instance”
10.   xsi:schemaLocation=“http://www.awareit.com/soam/
   2006/04/redel
11.  http://www.awareit.com/soam/2006/04/redel.xsd”>
12.
13. <resource uri=‘‘urn:uuid:fuell">
14.       <location    url=‘‘http://169.254.0.12/fuell_
   description.rdf’’  type  =‘‘http://www.awareit.com/
   soam/2006/04/srdfws#httpGet”/>
15.      <location   url=‘‘http://169.254.0.12/sparql"
   type=‘‘http://www.w3.org/2005/08/sparql-protocol-
   query/queryHttpPost”/>
16. </resource>
17.
18. </redel>
```

Lines 13–16 embody the information related to the resource that matched the query. An entity received the request message and identified `urn:uuid:fuell` as a possible solution to the query, thus generating the above response for the client and including two URLs in lines 14 and 15 where the client can retrieve extended information about `urn:uuid:fuell`.

The first URL provides information about `urn:uuid:fuell` when requested through a simple HTTP GET message, while the second URL implements the SPARQL protocol HTTP POST binding as described in [31]. The default in case no interface is specified is HTTP GET. The mRDP client features an HTTP server for receiving this kind of callback.

The minimum requirements any implementation must support at the client side are:

- Creating `application/com.awareit.plant` queries.
- Disseminating the queries to the multicast IP address 224.0.24.1 and multicast UDP port 2773.
- Reserving an HTTP callback URI for receiving HTTP POST messages with ReDEL payload.
- Processing ReDEL documents with the solution.

And at the server side:

- Joining the multicast IP address 224.0.24.1 and listening to the multicast UDP port 2773.
- Processing `application/com.awareit.plant` queries.
- Generating ReDEL documents with the solution.
- Issuing HTTP POST callbacks for conveying the ReDEL payload.

7. Performance evaluation of lexical and semantic discovery

The benefits of semantic discovery compared to traditional text-based matching are clear: more refined results are obtained by interpreting the information relationships. Devices and computers processing semantic queries over their managed information base seem more intelligent than traditional agents, and this is especially noticeable during discovery as we depicted in Section 3. However, what is the impact on performance for semantic queries?

In order to test the performance of the matchmaking mechanism in mRDP with and without reasoning, we executed the above search against 10 mRDP servers (Pentium 1.86 GHz) powered with a Java implementation of the Plant Query Resolution Algorithm. During the first round, the servers were configured to apply both ontology and domain rules reasoning. During the second round, no form of semantic reasoning was performed.

Every test was executed 8 consecutive times, thus 10 servers $\times$ 8 times $=$ 80 tests. The first execution provokes the initialisation of internal structures and, thus, takes more time, so we isolated it. Second and subsequent executions produce more stabilised results.

Absolute values are not representative since they depend on system configuration, so we analysed relative measures. In this scenario with few rules and basic ontological relationships, lexical matchmaking is about 2.82 times faster than semantic matchmaking in the first execution, when the initialisation takes place.

Subsequent executions are much more efficient, since internal structures have been already generated. In this case, lexical matchmaking performs even better, around 5.55 times faster than semantic matchmaking after stabilisation (see Fig. 5).

While lexical matchmaking increases linearly as the knowledge base gets larger, semantic matchmaking increases exponentially as more data and ontologies are provided. As expected, semantic matchmaking is more time-consuming than the simple lexical version.

However, this issue has minimal impact on discovery performance: semantic reasoning is only carried out at concrete moments of time when information or ontologies change. Once the knowledge base has been augmented via semantic reasoning, the queries can be resolved against the data repository with lexical-like performance as long as the information remains the same.

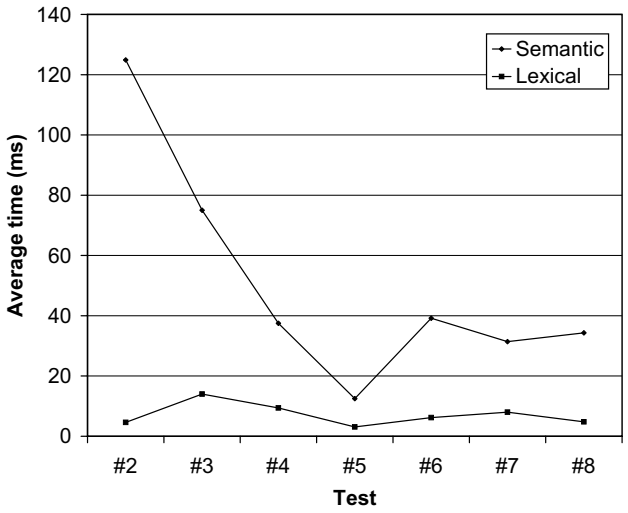

Fig. 5. Stabilised matchmaking performance in mRDP discovery.

In highly dynamic systems with continuous information updates, semantic discovery demands continuous reasoning. However, as the frequency of information change decreases, the performance of semantic discovery mimics that of lexical matchmaking for the same scenario (considering the overhead incurred by the augmented knowledge base).

When it comes to intelligence, semantic discovery is much more powerful than the traditional approach in order to find resources that meet some concrete criteria. It is able to browse the roots of the information to unleash the real meaning, create relationships and produce the right answers. We consider semantic discovery mechanisms as facilitators to provide the degree of intelligence required in modern distributed architectures.

8. Conclusion

mRDP provides a simple and powerful way for performing semantic queries in distributed environments, taking advantage of and reusing HTTP infrastructure. Several discovery mechanisms have been proposed in the past; none of them is widely accepted.

Edwards published a comparative analysis [9] about SSDP, Jini, Bluetooth SDP, SLP, Bonjour, Salutation, and some other systems. The comparison was based on the criteria of topology, transport, scope, search and security. Table 1 reproduces part

Table 1
Comparative table of discovery protocols and mRDP

System	Topology	Transport	Scope	Search	Security
UPnP SSDP	Peer-to-peer (P2P)	Unicast HTTP, multicast HTTP	Subnet	Type or ID	Authentication, access control
Jini	Hybrid	Unicast TCP, multicast UDP	Subnet, bridgeable	Type, ID or attribute	Jini/Java security mechanisms
Bluetooth Service Discovery Protocol (SDP)	Peer-to peer (P2P)	LMP and L2CAP	Proximity (−10 meters)	Type or attribute (universally unique identifier [UUID] only)	Link-level or service level encryption, authentication
Service Location Protocol	Peer to- peer (P2P) or directory	Unicast TCP, multicast UDP	Subnet, bridgeable	Type or attribute (LDAP v.3 search predicates)	Optional service authentication
Bonjour	Peer-to-peer (P2P)	Multicast DNS (mDNS), DNS service discovery (DNS-SD)	Subnet	Type	Optional IP security (IPsec), DNS security extensions (DNSsec)
Salutation	Peer-to-peer (P2P) or directory	ONC RPC over arbitrary transports	Depends on transport	Type or attribute	RPV authentication
Multicast Resource Discovery Protocol (mRDP)	*Peer-to-peer (P2P)*	*Unicast HTTP, Multicast UDP*	*Subnet, bridgeable*	*Semantic queries, attribute-based queries*	*HTTP-based security (HTTPS and HTTP Authentication)*

of the analysis involving the above protocols, with an additional row for mRDP.

mRDP exhibits some distinct characteristics compared with other alternatives. Remarkably, its powerful semantic search capabilities enable the intelligent processing of queries producing better results, both in quantity and quality, compared to traditional lexical matchmaking. Moreover, since mRDP takes advantage of the widely accepted and reliable HTTP infrastructure for callbacks, aspects such as authorisation and confidentiality are directly supported in the system.

Despite being originally conceived for ubiquitous computing scenarios, mRDP can also be applied in more general traditional computing networks for the semantic discovery of resources. Moreover, since these networks are populated by computers with full processing capabilities, it is feasible to place an SPARQL engine in every network node to benefit from full reasoning and querying capabilities.

Regarding future research on mRDP, we would like to highlight the possibility of improving the performance and expressiveness of Plant to resolve more complex queries. Although SPARQL can also be used as a query language, there are some devices that would support a slightly more powerful version of the Plant Query Resolution Algorithm, but would not support an implementation of a SPARQL engine. Finally, improving the performance of semantic reasoning in highly dynamic systems embodied on resource-limited devices is also a challenging goal in our research agenda.

References

[1] Apple Computer, Inc. Bonjour: Connect computers and electronic devices automatically, without any configuration, April 2005. Technology Brief.
[2] Franz Baader, Ian Horrocks, Ulrike Sattler, Description logics as ontology languages for the semantic web, Lecture Notes in Artificial Intelligence 2605 (2005) 228–248.
[3] Tim Berners-Lee, Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor, Harper San Francisco (1999).
[4] Tim Berners-Lee, Notation 3: An readable language for data on the Web, World Wide Web Consortium, March 2006. Online at http://www.w3.org/DesignIssues/Notation3.html.
[5] Tim Berners-Lee, James Hendler, Ora Lassila, The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities, Scientific American 284 (5) (2001) 28–37.
[6] Harry Chen, Tim Finin, Anupam Joshi, Dynamic service discovery for mobile computing: Intelligent agents meet Jini in the Aether, Baltzer Science Journal on Cluster Computing (2001) 343–354.
[7] Stuart Cheshire, Marc Krochmal, DNS-Based Service Discovery, August 2006. IETF Draft draft-cheshire-dnsext-dns-sd-04.txt.
[8] Stuart Cheshire and Marc Krochmal. Multicast DNS, August 2006. IETF Draft draft-cheshire-dnsext-multicastdns-06.txt.
[9] W. Keith Edwards, Discovery systems in ubiquitous computing, IEEE Pervasive Computing 5 (2) (2006) 70–77.

[10] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul J. Leach, Tim Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, 1999. IETF RFC 2616.
[11] Ned Freed, John C, Klensin. Media Type Specifications and Registration Procedures, December 2005, IETF RFC 4288.
[12] Yaron Y. Goland, Ting Cai, et al., Simple Service Discovery Protocol/1.0. Operating without an Arbiter, 1999. Internet Draft.
[13] Erik Guttman, Charles Perkins, John Veizades, Michael Day, Service Location Protocol, Version 2, June 1999. IETF RFC 2608.
[14] Tim Kindberg, John Barton, A web-based nomadic computing system, Computer Networks: The International Journal of Computer and Telecommunications Networking 35 (4) (2001) 443–456.
[15] Ora Lassila. Semantic Web, Quo Vadis?, October 2006. Keynote of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006).
[16] Diego López de Ipiña, Paulo Mendonça, Andy Hopper, TRIP: a low-cost vision-based location system for ubiquitous computing, Personal and Ubiquitous Computing Journal 6 (3) (2002) 206–219.
[17] Diego López de Ipiña, Juan Ignacio Vázquez, Daniel García, Javier Fernández, Iván García, David Sainz, Aitor Almeida, EMI^{2}lets: a reflective framework for enabling AmI, Journal of Universal Computer Science 12 (3) (2006) 297–314.
[18] Robert E. McGrath, Discovery and its discontents: discovery protocols for ubiquitous computing, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2000. Technical Report UIUCDCS-R-2000-2154.
[19] Robert E. McGrath, Semantic infrastructure for a ubiquitous computing environment, Ph.D. thesis, School of Computer Science, University of Illinois at Urbana-Champaign, 2005.
[20] Robert E. McGrath, Anand Ranganathan, Roy H. Campbell, M. Dennis Mickunas, Incorporating "semantic discovery" into ubiquitous computing infrastructure, in: Proceedings of System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003), 2003.
[21] Ippokratis Pandis, John Soldatos, Alexander Paar, Jurgen Reuter, Michael Carras, Lazaros Polymenakos, An ontology-based framework for dynamic resource management in ubiquitous computing environments, in: ICESS '05: Proceedings of the Second International Conference on Embedded Software and Systems (ICESS'05), IEEE Computer Society, Washington, DC, USA, 2005, pp. 195–203.
[22] Anand Ranganathan, Shiva Chetan, Jalal Al-Muhtadi, Roy H. Campbell, M. Dennis Mickunas, Olympus: A high-level programming model for pervasive computing environments, in: Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), IEEE Computer Society, 2005, pp. 7–16.
[23] Franklin Reynolds, An RDF framework for resource discovery, in: Proceedings of SemWeb 2001: The Second International Workshop on the Semantic Web, May 2001.
[24] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, Eve Schooler, SIP: Session Initiation Protocol, June 2002. IETF RFC 3261.
[25] Henning Schulzrinne, Anup Rao, Robert Lanphier, Real Time Streaming Protocol (RTSP), April 1998. IETF RFC 2326.
[26] Sun Microsystems, Jini architectural overview. Technical report, Sun Microsystems, 1999.
[27] Juan Ignacio Vazquez, Diego López de Ipiña, Iñigo Sedano, SoaM: a Web-powered architecture for designing and deploying pervasive semantic devices, IJWIS – International Journal of Web Information Systems, in press.
[28] World Wide Web Consortium, OWL Web Ontology Language Overview, World Wide Web Consortium, February 2004, W3C Recommendation.
[29] World Wide Web Consortium, RDF Primer, World Wide Web Consortium, February 2004, W3C Recommendation.
[30] World Wide Web Consortium, RDF Test Cases, World Wide Web Consortium, February 2004, W3C Recommendation.
[31] World Wide Web Consortium, SPARQL Protocol for RDF, World Wide Web Consortium, April 2006, W3C Candidate Recommendation.
[32] World Wide Web Consortium, SPARQL Query Language for RDF, World Wide Web Consortium, April 2006, W3C Candidate Recommendation.
[33] Weibin Zhao, Henning Schulzrinne, Enhancing service location protocol for efficiency, scalability and advanced discovery, Journal of Systems and Software 75 (1–2) (2005) 193–204.
[34] Fen Zhu, Matt W. Mutka, Lionel M. Ni, Service discovery in pervasive computing environments, IEEE Pervasive Computing 04 (4) (2005) 81–90.

Juan Ignacio Vazquez obtained a Ph.D. on Computer Science and Artificial Intelligence at the University of Deusto, Spain. Since 1997 he works as a lecturer on telematics, Ubiquitous Computing and mobile computing at the University of Deusto, and he also works as a researcher in MoreLab – Mobility Research Lab. His main research interests involve novel communication architectures for devices, the application of semantic reasoning in Ubiquitous Computing, and embedded intelligence for wireless sensor networks.

Diego Lopez-de-Ipiña completed a Ph.D. at the University of Cambridge, UK on Sentient Computing in 2002. After that, he took part on the Trigenix start-up, specialized on providing personalized interfaces to mobile devices. Since September 2003 he is a lecturer at the University of Deusto, Spain, and researcher of MoreLab – Mobility Research Lab. His main research interests are middleware architectures for Ambient Intelligence, context-aware mobile computing and the synergy between Web 2.0 and Pervasive Computing.