An ontology-based semantic service for cooperative urban equipments

D. Gregor, S.L. Toral*, T. Ariza, F. Barrero

E.S. Ingenieros, University of Seville Electronic Engineering, Camino de los Descubrimientos, s/n 41092 Seville, Spain

1. Introduction

The urban traffic control is extremely complex and dynamic. Metropolitan and rural areas are changing rapidly with the evolution of information and communications technologies (ICT), which calls for better organization and knowledge management. Intelligent infrastructure development in these areas are currently explored and extended by analyzing the adaptability of developed tools to the continuous changes that occur in the setting of ITS (intelligent transportation systems). The real-time traffic parameter estimation and control operations are a challenge for control of urban traffic systems (Choy et al., 2003). The current UTCS (urban traffic control systems) are usually based on centralized networks (Chen et al., 2008). However, current control systems are demanded to manage undesirable changes in traffic flow such as accidents, and be able to optimize the flow by adjusting traffic signals and coordinate operations for each signal (Toral et al., 2011).

In this context, the ITS can share information and operate in large collaborative environments. For example, an incident management system in urban environments can directly influence on emergency response, providing timely and accurate information on the incident and the seriousness of the case. However, in most cases, the heterogeneous information does not provide the necessary data to fulfill certain tasks. The information shared and common understanding can sometimes be confusing, leading to accidents, delays and chaos in the urban traffic, depending on the area to be implemented. In fact, one of the main challenges in the design and operation of ITS is to provide users a complete and transparent formal environment regardless of the characteristics and capabilities of the components connected to the system. The intelligent location of services and information is probably one of the most known problems in the ITS. The dynamic discovery of information, composition and invocation of services through intelligent agents represent a potential solution to these problems. For this purpose, devices and applications from different suppliers and vendors need a flexible way to work collaboratively with each other. They should be able to discover services autonomously, coordinate their work, resolve conflicts based on critical situations, negotiate with users; in essence, they should be

* Corresponding author. Tel.: +34 954 48 12 93; fax: +34 954 48 73 71. E-mail addresses: derlis.gregor@gmail.com (D. Gregor), toral@esi.us.es (S.L. Toral), matere@trajano.us.es (T. Ariza), fbarrero@esi.us.es (F. Barrero).
able to adapt to changes in the environment itself. Rockl and Robertson (2010) argue that the success of cooperative ITS applications is mainly in exchanging information between distributed nodes. According to them, the transmission of large flows of information contrasts with the limited bandwidth of the channels that tend to be shared by all nodes participating in the ITS. They propose that the solution lies mainly in the cooperative commitment to select relevant pieces of information for dissemination according to its value.

Technologies such as distributed artificial intelligence and multi-agent systems (MAS) have gained considerable importance due to their ability to solve complex real world problems (Balaji and Srinivasan, 2010), using a collection of collaborative agents which interact with each other to achieve common and individual goals (Li et al., 2006; Chen and Cheng, 2010). However, in heterogeneous situations like urban environments, distributed systems are well known for their difficulty to interoperate between agents, which make it grow interest on unified software platforms and standard implementations (Wang et al., 2006).

SOA is presented as an attractive alternative to enable interoperability of systems and reuse of resources. It can serve as underlying support for MAS applications in distributed and heterogeneous systems (Wang et al., 2010; Suri et al., 2007). To implement SOA systems, two main technologies stand out, WebServices and CORBA. The core of WebServices is SOAP (simple object access protocol) communication protocol based on XML (extensible markup language) to integrate services. But processing large SOAP messages would reduce significantly the system performance, causing bottlenecks (Tekli et al., 2012). The common object request broker architecture (CORBA) provides object reference and interface semantics so that method invocations between objects written in different programming languages or executing on different end systems are performed transparently through a set of object request brokers (ORBs) (Gill, 2006).

SOAP communication produces high network traffic causing high latency compared to CORBA. This represents a central problem, specifically in wireless communication networks (Phan et al., 2008). While Web services are presented at times as more flexible compared to CORBA, they are still at an early stage compared to CORBA. In practice, SOA-based applications are not always successful because most of them are made on an ad-hoc based primarily on personal experiences (Guo et al., 2010). Although companies are increasing their dependence on SOAs, these systems are still in a nascent stage, immature and with large security problems (Kabbani et al., 2010). Unquestionably CORBA does not offer the same as SOA promises, but is presented as the best subset of tools to support what SOA “tries to be". CORBA is a mature technology that includes the most important services, as the naming service and the event service and more. The location of the information or a specific service may be, for example, through a CORBA trading service, where representation would be given by pairs of name/value and their properties cover the reference to the service itself, descriptively. A trader acts as a database that stores references to objects that are described by their properties (Henning and Vinoski, 1999).

Recent publications has begun to propose new architectures for ITS. Chen et al. (2008) presented the design and implementation of a framework for public transport. They include a mechanism for collection of data through Web services, the incorporation and composition of schemas for route planning. MAS-based approaches have been adopted in various transportation management systems (Chen and Cheng, 2010), particularly in traffic and bus management. Fernandez and Ossowski (2011) support the premise that the use of MAS allows a decoupled design and implementation of different modules (agents), thus promoting reuse. They also show the study on a multi-agent architecture, service-oriented for the construction of advanced DSSs (decision support systems) for transportation management. They show how different dialogues between DMs (decision makers) and the system itself can be modelled in terms of organizational concepts through taxonomies of roles and types of interactions. Terziyan et al. (2010) presented a focus on requirements and architecture necessary for traffic management systems, showing how such a system can be beneficial from the semantic point of view through technologic agents but at the same time wondering how to combine them with the data processing and automated tools. However, none of the above approaches takes advantage of the service oriented paradigm or the dynamic selection of services. The main problem remains in working with the information between these services: how to find services that meet real needs, how to trust that the data received/sent are reliable or which techniques and mechanisms implemented on the ITS meet a real environment.

The application of traditional search mechanisms, based on basic properties, can lead to inefficient results. The search for a particular service should be done intelligently. There should be a mechanism of differentiation and weighting of information. Customers who make requests on a particular service need to be able to gain some level of truth and confidence about a particular service. They must be able to take decisions of convenience between services in order to obtain the best results. Customers require some intelligence in the search. Trading services are unable to provide this level of intelligence. Some approaches in the design and implementation of a CORBA trading service do not address the main problems of finding services or large transfers of information in distributed systems, triggering large performance and scalability problems (Craske and Tari, 1999; Zhu et al., 2005).

In recent years, the incorporation of ontologies has increased, above all in semantic Web sites, enabling interoperability of heterogeneous data (Zhao et al., 2009; Xue et al., 2012). The incorporation of ontologies implemented in a standardized set of concepts such as RDF (resource description framework) (W3C Members, 2004a) extended by class and properties specification mechanisms as RDFS (RDF-Schema) (W3C Members, 2004b) and a service that manages ontologies in a cooperative way, would play an important role in ITS-oriented environment. This paper proposes the development of an SS (semantic service) implemented in TAO CORBA, and ready to run on different processors such as SPARC, AMD, Intel, ARM, PA RISC (32/64 bit), etc., and operating systems like Solaris, Windows, Linux, HP-UX, IBM, Mac OS, etc. The SS will use ontologies mainly based on theories of knowledge and natural language processing. The different traffic services, mostly implemented on devices with embedded architectures, interoperate with the SS who will be responsible for providing and managing all the information of the C/S (client/server) architecture, and intermediate between them in a dynamic and intelligent way. The SS will use cooperative transaction data from the time of collection of information by automating decision-making in situations that require intelligent intervention of the urban environment.

The rest of the paper is structured as follows: Section 2 presents the developed ontologies on intelligent transportation systems field. Section 3 explains the proposed semantic service architecture and implementation. The details about the implementation of the semantic service in TAO CORBA are included in Section 4, followed by the obtained results in Section 5. Finally, conclusions are discussed in Section 6.

2. Ontologies on intelligent transportation systems

The use of ontologies can substantially reduce the failure rate when searching services in scenarios like urban environments,
Section 5.3 details a real scenario where the use of the ITS ontology greatly helps to the reduction of failures in information recovery. The ontologies provide the semantic level and the appropriate means to specify the best mechanisms for interoperation based on the logical representation of information (Lamparter and Schnizler, 2006). This semantic annotation will provide large amount of information in a logical structure, allowing devices (clients) to search and link information in a more automatic and straightforward way. It is clear that in areas where the implementation of ITS is increasing, customers and/or devices need to find automatically appropriate services and information, minimizing delays and increasing the system performance. The dynamic location of services with certain characteristics and properties through semantic search certainly promises greater interaction and engagement of customers and accuracy in the provided information as well.

From the point of view of interoperability between devices, the most striking advantage of using ITS ontologies would be the intelligent information retrieval. The devices could make use of data and metadata of different types at runtime, and the services would be able to store their object references in descriptive ontologies.

In this paper, the terms and schemas used to build an ITS ontology have been selected from the taxonomy proposed by RITA (2010) (Research and Innovative Technology Administration) and the information collected by USDOT (2011) (United States Department of Transportation), OPJ (Office Program Joint)

![Diagram of ITS management schema](image)

**Fig. 1.** Arterial management schema (piece of ITS.rdfs).

**Listing 1.** Example of homonymy in the serv_name property and in the Car.Counter object.
based on the impact that ITS projects have on transport networks. The scope of the implemented ontology in this publication is the ITSs in general, considering the most important classes involved in ITS, and storing them in a schema called ITS.rdfs. For illustration purposes, we show in Fig. 1 the part of the ontology related to arterial management. This piece of the ITS ontology can be used directly as it is shown or can be integrated into a larger knowledge base, thanks to the adaptability of the proposed semantic service.

This piece of the ontology ITS.rdfs mainly consider all the information related to arterial management systems, deployed at urban intersections. The concepts derive from the root “THING”, which represents the general class of all ITS ontology. Each class has a unique identifier or name that sets it apart from the whole structure. These classes have in turn subclasses, which represent more specific concepts to the class. Relations between classes represent a type of interaction between the modeled domain concepts.

The relationship “is-a” between classes is defined in terms of the relation “instance-of”. For example, Traffic_Control class is a sub-class of the Arterial_Management class, if and only if all the Traffic_Control instances are also instances of Arterial_Management.

Fig. 2. Semantic service (SS) block diagram.

Listing 2. Semantic manager IDL.

```
module Semantic {

  interface Manager {

    exception KbNotFound {;
    }
    exception KbNotMatch {;

    void AddToSchema (in string NewInfo)
      raises (KbNotFound);

    void ClientQuery (inout string QueryDetail)
      raises (KbNotFound, KbNotMatch);

    void destroy {;
    }
  };
}
/* end module Semantic */
```
Instances represent concrete objects belonging to a domain class. These classes itself represent devices, services or containers of services implemented in the ITS environment. When a service wishes to incorporate information within the knowledge base run by the SS, the service analyzes the model looking for the <rdf:type> property. This property is used to ensure that the resource, which has been sent by the exporter (server) is actually an instance of at least one of the classes in the schema managed by the SS. The piece of the ontology ITS.rdf presented in Fig. 1 shows the classes and subclasses in charge of traffic along urban roads and intersections, using traffic detectors, signals, and various channels to communicate information to travelers. They use the information collected by traffic surveillance devices. They are also responsible for disseminating important information about the journey and conditions at all times such as DMS (dynamic message signs) or HAR (highway advisory radio). An important issue to consider in ontological semantics is how to overcome the problems of homonymy. A construct can have an unambiguous meaning, for example, be a class, but the term used to describe this construct belongs to the natural language, and therefore is subjected to ambiguity. One of the features of semantic service discovery is that it is based on a set of keywords or indexing terms that can be syntactically equivalent but semantically different (homonymy). The declarations of namespaces help to eliminate ambiguities and solve the problems of homonymy. Namespaces are collections of names, identified by a reference URI (uniform resource identifier) which can be used as element types and attribute names. The URI combines URNs (uniform resource names) and URLs (uniform resource locators) (names/addresses) to identify resources. Thanks to the differentiation with namespaces, subjects, predicates and objects can be equal homonomically within other statements in the same schema. For example, different services on different devices may have the same property serv_name, the same object Car.Counter, but with a different namespace because they provide different type of information. This is shown in Listing 1.

3. Semantic service description

The main goal of this work is to provide the correct choice of services and to guarantee the accuracy of the information received/sent, as this is a very important and key desire in urban
environments. The incorporation of semantic location, invocation, discovery, composition, execution and monitoring of services is proposed as the most feasible solution in distributed systems. The semantic service (SS) has been implemented to provide a formal mechanism for semantically describing all relevant aspects of urban services to facilitate the automation in discovering, combining and invoking urban services on distributed platforms.

The semantic service (SS) is responsible to perform the following tasks:

- Parse and store the main schema to manage.
- Create a copy and store the schema as triple-stores.
- Include in the model information submitted by importers.
- Search for compatible nodes in the main schema and add new resources in case the received model match with the schema.
- If the received model does not match with some information managed by the SS, an exception is set back.
- Address requests for customers queries about the model.
- Parse the triple-store and return to the importer the query result or an exception.

A general outline of the system architecture is shown in Fig. 2. Listing 2 shows the IDL SemanticManager, core of the SS to establish C/S communications. The interface manager declares the API used by objects to communicate with the SS. Both, the importer and the exporter who want to communicate with the SS can make calls to specified remote methods in this interface. We also set the exceptions the importers and exporters should receive in case the knowledge base is not found or the query does not match the required information.

Next, the sequence diagram of the main processes involved in the SS is presented. The first process corresponds to the communication with the exporter of the semantic information. Fig. 3 shows how the SS deals with requests of exporters for inclusion of new information that is required to be published.

The SS reads and parse the information received looking the `<rdf:type>` property, instance of rdf:Property that is used to indicate that the resource is an instance of a class. This property is used to ensure that at least one resource that has the string received from the exporter is actually an instance of at least one of the classes in the main schema. If parsing the information received does not find this label, the listing process could not be done. In this sense, if there were redundant or duplicate nodes, they will replace to avoid data replication. Then a disjunctive function between subject and object nodes of the received model and the main schema is applied. If one of the claims is true, the expression \( PVQ \) will be true. If it becomes true in one of the nodes, the received model can be stored within the knowledge base of the SS. If it is false, this would mean that the exporter trying to export their data does not belong to the domain over which the SS works. In this case, an exception Semantic::Manager::KbNotFound() is sent back to the exporter.

The second process is related to the importer, which makes a request for a particular service to the SS. Fig. 4 describes how the SS processes queries received from importer requests.

The importer is able to perform RDQL or SPARQL queries to the SS. The SS reads the received string, checks the type of language and builds the query. Once the language is defined, the SS checks and parses all nodes in the main schema for results. The SS will be capable of delivering links to the customer, boolean statements, RDF graphs, triples or string in syntax form. The SS use Rasqal to parse and query the stored schema. The string sent to the importer contains all the necessary information to make requests directly to the exporter. Moreover, this string includes the IOR of the exporter. If the received query does not have concordance with the main schema, the SS send back to the importer an exception Semantic::Manager::KbNotFound(). In case there are not any matches, the importer will receive an exception: Semantic::Manager::KbNotFound().

Fig. 5 illustrates the complete sequence of the proposed architecture. More specifically, this figure shows the sequence...
that takes place when a request of the exporter and importer arrives to the SS.

4. Semantic service implementation

For the development of the SS, a set of libraries (free software/open source) Redland RDF will be used to deal with the semantic information (Beckett, 2011a). One of the main advantages of using the Redland API is that it is implemented in C and can easily be ported to C++ or compiled for embedded systems, as in our case. There are other APIs developed for other languages such as OWL-Protegé or Jena, implemented in Java. However, Java is not always
fully supported in all embedded devices, and considering the high heterogeneity of ITS equipments it is reasonable to choose a more flexible option like Redland. Redland provides high-level interfaces, which allows the use of instances and models stored in different structures, and the ability to perform queries and manipulate data using different languages such as C, Perl, Python, PHP, etc. Another visible advantage of using this library is that it uses an object-oriented API, which provides numerous implementations of classes and modules that can be added, removed or replaced, to allow different functions and applications with specific optimizations. The framework provides a nucleus for developing new RDF applications experimenting with various implementation techniques (Beckett, 2001). Redland is designed to cover the four lower layers of semantic building block, Fig. 6 (Berners-Lee, 2000).

In the Redland class diagram, Fig. 7, classes are used and associated with each other. The support classes are used throughout the rest of the classes, as needed. The Stream classes are used whenever a statement sequence is accepted or generated by the Model, Storage or Parser class. The Model class uses the Stream class to perform serialization/de-serialization of the model and returns a list of statements by reference. The Parser class is only used to provide a sequence of statements as a result of the analysis. At a simpler level, each model object has a mapping one-to-one to the storage object that it represents (Beckett, 2001).

The statements, in the class diagram, need to be stored efficiently. The storage needs to use existing systems like relational database or other mechanisms that support the scalability of the information. It is expected that information is available and can be manipulated at runtime in distributed environments. For performance testing, we planned to use persistent storage on a database engine (free software/open source) embedded general-purpose db-1.5.19 BerkeleyDB and a file storage system for comparison purposes as well. From the point of view of database performance, Berkeley is designed to handle high-performance applications on critical mission. It supports multiple languages such as C, C++, Java, Perl, Python, PHP, Tcl, and so on. This database stores each data value with an associated key. It also supports multiple values for a key mapping. It has been implemented on many platforms including UNIX, Linux, Windows and some real-time OS (operating systems) (Oracle, 2010). We will also use a library (free software/open source licenses issued under LGPL (GPL)) Raptor RDF Syntax Library (Beckett, 2011b) which provides a set of parsers and serializers to generate RDF triplets analyzing the syntax, or serializing the triplets into syntax. It supports syntax analyzers for RDF/XML, N-Quads, N-Triples, TriG, Turtle, RDF labels including all versions of RSS, Atom 1.0 and 0.3, GRDDL and microformats for HTML, XHTML/XML and RDFa. The serializer supports RDF/XML (regular and short), Atom 1.0, GraphViz, JSON, N-Quads, N-Triples, RSS 1.0 and XMP. This tool has been designed to work with Redland, but as a standalone tool. It is a portable library that works across multiple POSIX systems (Unix, GNU/Linux, BSD, OSX, cygwin, win32). Another library that will be used (free software/open source) is Rasqal (RDF Query Library) (Beckett, 2011c), responsible for building and executing queries. This tool will be implemented on the importer side returning a boolean link, syntax or RDF triplets graph. It supports query languages like SPARQL 1.0, RDQL, SPARQL Query Draft 1.1, Update 1.1 Syntax and experimental SPARQL extensions (LAQRS). Rasqal can write query results XML link SPARQL, SPARQL JSON, CSV, TSV, HTML, ASCII tables, RDF/XML and SPARQL Turtle/N3 and read XML, RDF/XML and Turtle/N3.
For communicating the SS, the exporter and importer, TAO CORBA has been chosen as the basic architecture to supper the distributed system (Velastin et al., 2004). The use of CORBA as a framework of standards and open systems concepts defined by the OMG (2003) provides the possibility of developing cooperative services. Remote objects can be invoked transparently in distributed and heterogeneous systems, through the ORB (object request broker). The ORB will be responsible for managing communication between the importer and exporter (Toral et al., 2011).

The main advantage of using CORBA for developing semantic services focused on ITS is that the communication between importers and exporters can be done using different programming languages, i.e., a client written in Java can invoke operations on a CORBA object written in C++, Ada, Perl, etc. and can be implemented on different hardware platforms. A client application can invoke a method on a CORBA object sending a request through the client stub generated by the IDL compiler, which is a local representation of a CORBA server (Huang, 2011). From the point of view of the customer, no matter what the object is local or remote, since the presence of the stub makes the invocation transparent in fact. The customer will need to obtain a reference to the server to perform the invocation. The server-side process will be similar, but using the skeleton generated by the IDL compiler.

TAO is presented as a best platform that fits the design requirements and it is stable even in very large bursts of requests. TAO CORBA has solved many of the problems of communication, including communication from heterogeneous applications, offering a high level of transparency. A desirable feature in any distributed architecture is the quickly, efficiently and intelligently interoperability of devices. The construction, implementation and coordination of cooperative methods can be improved reusing existing ontologies and sharing information between them. In this line, we propose the design and implementation of the SS, able to analyze and store ontological schemas in different storage providers such as Files, BerkeleyDB, PostgreSQL, governing and coordinating cooperative work and information flow between importer/exporter devices. The SS is responsible for managing the interoperability of metadata. Exporters and importers that have relevant metadata may consult other devices and cooperatively interact at runtime. The exporter who wants to export their data to the main schema managed by the SS must belong to the ontology domain; otherwise, it will be rejected. The SS will be in charge of looking for a service according to the importer query requirements.

![Fig. 10. Performance parsing and storing the main schema in BerkeleyDB.](image1)

![Fig. 11. Semantic exporter RDF.](image2)
5. Results

The semantic service has been implemented in a hardware platform to test its functionality. To emulate the behaviour of urban equipments, we have considered a hardware platform of embedded devices typically used in ITS equipments. More specifically, we have used embedded devices based on ARM microprocessors for this purpose, using cross-compilation techniques. Basically, the considered equipments are executing a main process consisting of typical urban operations, like controlling a smart camera or a traffic light, but at the same time they are executing several CORBA services including the proposed semantic service.

Fig. 8 shows the proposed platform using CORBA to communicate several distributed resources allocated on several

Fig. 12. Performance of searching compatible nodes and adding new resources in the main schema.

```
(1) PREFIX kb: <http://edsplab.us.es/kb#>
(2) CONSTRUCT {?Car.Counter kb:ior ?ior .
(3)   ?x kb:serv_creator ?serv_creator .}
(4) WHERE {
(6)   ?x kb:serv_creator ?serv_creator .
(7) LIMIT 1
```

Listing 3. SPARQL query Q from importer.

Fig. 13. Performance of parsing the main schema and returning the result to the importer based on the SPARQL query.
devices through a TCP/IP network. The semantic service is implemented on the equipment labelled as SERVER-EPIARM, which is using a GNU/Linux Debian port for ARM processors. The other two equipments, labelled as ARM-1 and ARM-2, implement typical traffic control operations and Standard CORBA services as well. They are based on an ARM 9 processor, and their applications have been compiled using arm-Linux gcc-3.3.2 and glib-2.3.2. They also include a Linux operating system kernel 2.4.20-celf3. We have implemented the SS, importers and exporters in TAO CORBA version 1.6a. The three boards based on ARM processors of the proposed platform can be visualized in Fig. 9.

5.1. Performance and scalability of the proposed system

We conducted several experiments to evaluate the performance and scalability of the proposed system. An important metric of performance is the throughput, which can be measured as:

$$\text{TputkB} = \frac{\text{size (kB)}}{\text{RTT (s)}}$$

where RTT is round trip time in seconds. However, the semantic service is working with data consisting of triplets. Thus, throughput is redefined as follows:

$$\text{TputkT} = \frac{\left(\frac{\text{no. of triples}}{1000}\right)}{\text{RTT (s)}}$$

The overall system performance has been tested using a Berkeley database as main storage system.

First, when running the SS, we have to load a schema or model to work with it. This schema is stored in the database while exporters/importers make requests, everything managed by the same SS. For our tests, the ontology ITS.rdfs consisting of 524 triplets and a weight of 108.61 kb has been used. The process of analysis and storage of the main schema is done in 290 ms, with a transfer rate $\text{TputkB} = 374.53 \text{ kb/s}$. The behaviour of the SS is measured in Fig. 10 as a function of the number of triplets in the main schema stored in Berkeley DB and a file system.

Notice that the performance during the experiment is quite stable in both storage systems even with data set of 12 ktriplets.

![Fig. 14. TputkT parsing and storing the main schema in the databases.](image-url)

![Fig. 15. Real scenario recovering ITS information.](image-url)
The next process was to calculate the system performance by adding new Statements of the exporter in the main schema. In this case, the exporter creates a RDF file that contains 14 declarations, Fig. 11. This RDF is marshalled in a string containing all the exporter information, which it is sent to the SS. The more detailed the service information is, the more accurate and intelligent searches can be performed by clients.

These statements contain all information necessary for the importer to locate the exporter on parameterized queries. The SS receives the exporter string, search for supported nodes within the framework and adds the new resources in case of a positive match. The 14 triplets have a weight of 2.84 kb. The delay of the SS when reading and parsing the received temporal model is 16 ms with a transfer rate of Tputkb = 176.40 kb/s. Once the main model is corroborated, the next step consists of adding new declarations. The delay of seeking supported nodes in the main schema and adding new statements is 339 ms with a transfer rate of Tputkb = 320 kb/s. With the new added statements, the main schema reaches 538 triplets. As the above results, this process is illustrated in Fig. 12 varying the size of the main schema.

The same than the process of parsing and storing the main schema, the performance of adding new exporter resources behaves quite stable, even with data set of 12 kTriplets. To perform the test with the importer, we have used SPARQL following a W3C recommendation. Listing 3 shows the SPARQL query Q made by the importer to the SS. The query should return the IOR of the Car.Counter service and its creator. The query begins in line (1) with the clause PREFIX, which associates the label kb to the IRI <http://edspab.us.es/kb#>. In line (2), the clause CONSTRUCT is responsible of returning to the client an RDF graph built by the substitution of variables in a set of triplet type templates.

Lines (5) and (6) provide patterns to meet the outcome of the query on the graph data. In line (7), the clause LIMIT 1 provides an upper limit in the number of returned results. In this case it is set to 1, as the results should be only one IOR. The importer then performs the query and the SS parse it into the database looking for a result that satisfies the request. The delay of SS doing parsing and processing queries was 26 ms. In this process, the SS model is marshalled in a string sent to the importer. The weight of marshalling the data to be returned to the importer is 1.05 kb, and contains all the information in RDF format that the importer needs to reach the exporter. Fig. 13 shows the performance of the SS ClientQuery process by making the query on the managed database.

From the point of view of implementation, the file storage system is not feasible due to the high latency obtained. This method has no indexing of data. It is only useful for small schemas models stored in memory. The SS through Redland, uses multiple hash to store various combinations of subjects, predicates and objects requiring quick accesses. The persistent storage via Berkeley DB is the most mature and suitable for large models, tested in 2–3 million of nodes (Beckett, 2011a). The obtained results show that even working with a total size of 12 kTriplets, the whole schema is analyzed and the response is built in less than half a second in the worst case using Berkeley DB. The importer received the response of the SS in 41 ms. Once the importer receives the marshalled string, it must re-build the RDF model parsing it as required by the query. In our example, the importer wants to find the Car.Counter exporter IOR in the received model. The importer delay for extracting the IOR was 56 ms, while the delay making the request and receiving the

### Table 1
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>serv_name</td>
<td>Car.Counter #0</td>
</tr>
<tr>
<td>urban_location</td>
<td>Paseo de Colon S/N Torre de Oro</td>
</tr>
<tr>
<td>serv_description</td>
<td>Service counting urban vehicles at intersections</td>
</tr>
<tr>
<td>host_name</td>
<td>EpiArm12</td>
</tr>
<tr>
<td>serv_name</td>
<td>Car.Counter #1</td>
</tr>
<tr>
<td>urban_location</td>
<td>Paseo de Colon 07</td>
</tr>
<tr>
<td>serv_description</td>
<td>Service counting urban vehicles at intersections</td>
</tr>
<tr>
<td>host_name</td>
<td>EpiArm13</td>
</tr>
</tbody>
</table>

Listing 4. Importer trading service query for scenario of Fig. 14.

```
PREFIX kb: <http://edspab.us.es/kb#>
CONSTRUCT {?Car_Counter kb:ior ?ior .
  ?x kb:serv_latitude ?serv_latitude .
  ?x kb:urban_location ?urban_location}

WHERE {
  ?x kb:serv_latitude ?serv_latitude .
  ?x kb:urban_location ?urban_location
  FILTER regex(?urban_location, "Paseo de Colon Torre de O", "i")
}
LIMIT 1
```

Listing 5. Importer semantic service query for scenario of Fig. 14.
response from the exporter was 25 ms. As a result, the entire process from importer query until the exporter response requires 122 ms, with a transfer rate \( T_{\text{putkb}} = 25.67 \text{ kb/s} \).

In general, these models are subjected to failures and random variations in time, depending on many factors like speed of the device where they are implemented, processing rate, reliability, buffer size, etc. Obviously, when we talk about distributed systems in embedded systems, the performance is affected.

5.3. Comparison to CORBA trading service

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car.Counter</td>
<td>urban_location</td>
<td>Paseo de Colon S/N, Torre de Oro</td>
</tr>
<tr>
<td>Car.Counter</td>
<td>serv_latitude</td>
<td>37.225887</td>
</tr>
<tr>
<td>Car.Counter</td>
<td>serv_longitude</td>
<td>5.564797</td>
</tr>
<tr>
<td>Car.Counter</td>
<td>ior</td>
<td>IOR:01000000010000049444c3a565672e6f726725465737424a973467567567</td>
</tr>
</tbody>
</table>

As a difference to the trading service that can return partial results without any further indication (Henning and Vinoski, 1999), the database of the SS ensures that the obtained result is successful or otherwise, an error indication is returned. More specifically, the use of namespaces ensures there is no duplication and the resolution of homonymy problems in the retrieved information. Fig. 16 illustrates the returned data structured in RDF/XML format.

The ability to specify metadata schemas in RDF allow applications to access a particular schema or an accessible resource publicly available via Web, and recover the structure and semantics of that particular set of elements. From a realistic point of view, this not guarantees a total exchange and interoperability between the various sets of metadata, but greatly simplifies this task and significantly reduce faults as a difference to the trading service, where data and information is presented in pairs.

Fig. 17 compares the delay in recovering information for the importers of trading and semantic service, considering that both of them have to look for in a list of 45 services. The results show that the semantic importer, even with increased data flow, provides better results than the trading importer.

The transfer rate in kb/s is compared in Fig. 18. Obviously, the throughput for the SS is higher as it is working with triplets.

6. Conclusions

This paper highlights the possibilities offered by the recovery of semantic information as well as the flexibility of using metadata in ITS environments. The main objective of applying ontologies in the field of ITS is the possibility of reusing the existing knowledge base to improve the accuracy of the information received/sent information as well as facilitating the communication between devices and applications achieving a common understanding between them. More specifically, the proposed SS provides a formal mechanism for semantically describing all

![Fig. 16. Structured RDF/XML received by the importer.](image)

![Fig. 17. Delay in recovery of information.](image)
relevant aspects of urban services to facilitate the automation in discovering, combining and invoking urban services on distributed platforms.

Experimental results demonstrate the feasibility and effectiveness of the proposed approach, being Berkeley database the most feasible storage system. The addition of SPARQL queries on the importer side avoids dealing with rigid applications when importing data. The importer would only need to modify the query to obtain different results as a RDF/XML structure.

Additionally, the SS was compared to the trading service using an ARM embedded platform. The provided application example shows the information represented in pairs does not satisfy the requirements of accessibility, especially in situations with problems of homonymy. In general, the higher the semantic description and logical structuring of the metadata, the greater the precision obtained in information retrieval. As a future work, the proposed SS could be extended creating federated services, allowing interoperability across different databases.

References


Lamparter, S. Schnizler, B. Trading services in ontology-driven markets, University of Karlsruhe (TH), Karlsruhe, Germany, Symposium on Applied Computing, 2006.


OMG. Common object request broker architecture (CORBA/BIP), Technical report; formal/02-11-03.

Oracle. BerkeleyDB 11g release 2 (Library version 11.2.5.1) documentation, 2010.


Rockl M, Robertson P. Data dissemination in cooperative ITS from an information-centric perspective. IEEE International Conference on Communications (ICC) 2010:1–6.


W3C Members. RDF vocabulary description language 1.0: RDF-Schema, W3c recommendation, 2004b.


