Pervasive and Mobile Computing 15 (2014) 87–99

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

SHERLOCK: Semantic management of Location-Based Services in wireless environments

Roberto Yus [a,*], Eduardo Mena [a], Sergio Ilarri [a], Arantza Illarramendi [b]

[a] University of Zaragoza, María de Luna 1, Zaragoza, Spain

[b] Basque Country University, Manuel de Lardizábal s/n, San Sebastián, Spain

ARTICLE INFO

Article history:
Available online 13 August 2013

Keywords:
Location-Based Services
Knowledge representation and reasoning
Real-time query processing
Pervasive data management

ABSTRACT

Location-Based Services (LBSs) are attracting great interest nowadays, mainly due to the economic value they can provide. So, different applications are being developed for tracking, navigation, advertising, etc., but most of those applications are designed for specific scenarios and goals with implicit knowledge about the application context. However, currently it is a challenge to provide a common framework that allows us to manage knowledge obtained from data sent by heterogeneous moving objects (textual data, multimedia data, sensor data, etc.). Moreover, the challenge is even greater considering situations where the system must adapt itself to contexts where the knowledge changes dynamically and in which moving objects can use different underlying wireless technologies and positioning systems.

In this paper we present the system SHERLOCK, that offers a common framework with new functionalities for LBSs. Our system processes user requests continuously to provide up-to-date answers in heterogeneous and dynamic contexts. Ontologies and semantic techniques are used to share knowledge among devices, which enables the system to guide the user selecting the service that best fits his/her needs in the given context. Moreover, the system uses mobile agent technology to carry out the processing tasks wherever necessary in the dynamic underlying networks at any time.

1. Introduction

In the last years the interest in mobile computing has grown due to the ever-increasing use of mobile devices and their pervasiveness. The low cost of these devices, along with the high number of sensors and communication mechanisms they are equipped with, make it possible to develop useful information systems. Using special kinds of sensors, location mechanisms enable the development of *Location-Based Services (LBSs)* [1]. These services provide value added by considering the locations of the mobile users to offer customized information. For example, LBSs for taxi searching [2], helping firefighting [3], detecting nearby friends [4], or multimedia retrieval in sport events [5] have been presented, among many others.

However, current LBSs are usually designed for specific scenarios and goals and are based on predefined schemas for the modeling of the elements involved in their scenarios. Moreover, the context knowledge they manage is implicit; that is the reason why they only work for one specific goal. For example, a user that arrives in a city must know (and understand) which LBSs could provide transportation information in that city. Some ad hoc solutions have been proposed to provide users with LBSs (e.g., [6,7]) but there is a lack of a general and flexible framework that can be applied in many different scenarios.

* Corresponding author. Tel.: +34 976259876.
E-mail addresses: ryus@unizar.es (R. Yus), emena@unizar.es (E. Mena), silarri@unizar.es (S. Ilarri), a.illarramendi@ehu.es (A. Illarramendi).

1574-1192/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.pmcj.2013.07.018

To build such a general system by simply merging preexisting LBSs is not straightforward: it is a challenge to provide a common framework that allows (1) managing knowledge obtained from data sent by heterogeneous moving objects (textual data, multimedia data, sensor data, etc.); and (2) considering situations where the system must adapt itself to contexts where the knowledge changes dynamically and in which moving objects can use different underlying wireless technologies (fixed, wireless, ad hoc, etc.) and positioning systems (GPS, GLONASS, cell-based positioning).

In this paper, we present *SHERLOCK*,[1] a general and flexible system to provide LBSs based on the use of semantic techniques and mobile agents. As its namesake, the well-known Arthur Conan Doyle's character, SHERLOCK uses abductive and deductive reasoning to infer information to answer user requests. In our opinion, the use of semantic techniques can enable the development of intelligent LBSs [8]. Thus, the system uses ontology reasoning and alignment methods to represent and manage, in a distributed way, the knowledge that describes objects and interesting areas in a scenario. In this way, the system guides the user in the process of selecting the LBS that best fits his/her needs; the participating objects/devices can cooperate and exchange data and knowledge among them to relieve the user from knowing and managing such knowledge directly. Furthermore, thanks to the use of mobile agents [9], it is possible to distribute the load of the system (both the CPU power and the communication costs) wherever it is needed in the wireless environment. In this way, the required processing tasks can be carried out on the most appropriate device in the scenario. In summary, the main benefits offered by our system, from the user's point of view, are as follows.

1. It offers to the user all the available interesting LBSs at each moment and, after choosing one of them, it helps the user to express his/her information needs by querying the local knowledge at the user device. In this way, it relieves the user from managing specific knowledge about LBSs.
2. It reconciles the different views of the world and the vocabulary used to describe objects and requests. This is achieved by supporting a decentralized and dynamic discovery of new kinds of objects (that can provide different services/contents). In this way, the system manages up-to-date knowledge about the LBSs provided to the user.
3. It manages heterogeneous (fixed or mobile) devices that can be part of the system, each of them having different capabilities. Moreover, it adapts itself in run-time to different underlying networks, such as fixed infrastructures (e.g., 3G, wired networks, etc.) and Mobile Ad-hoc Networks (MANETs) [10].
4. It continuously carries out the processing to the most appropriate nodes in order to balance the processing load and communication tasks, by using mobile agents. This is important to alleviate the limited CPU power, storage, and communication capabilities of mobile devices.

The rest of this paper is as follows. First, in Section 2 we present two motivating use cases that illustrate some of the many different scenarios that SHERLOCK can manage. In Section 3, an overview of the system architecture is provided. In Section 4, we explain how SHERLOCK manages the knowledge that defines moving objects and scenarios. In Section 5, we explain how SHERLOCK offers the available LBSs to the user and helps him/her to express his/her information needs in order to obtain a user request to process. In Section 6, we explain how SHERLOCK processes user requests in distributed scenarios. In Section 7, the motivating use cases are revisited to show how to solve them following our approach, illustrated by the prototype of SHERLOCK that we have developed. In Section 8, we present some related works. Finally, conclusions and future work appear in Section 9.

2. Motivating scenarios

In this section, we present two motivating use cases (as examples of many others) that show the heterogeneity and complexity, as well as the interest, of having a flexible and global system as a common framework to provide mobile users with different non-predefined LBSs.

2.1. Looking for transportation

Imagine a person that has just arrived at the airport of a city in a foreign country and wants to get to a certain hotel but does not know the best way to go there. Indeed, in that city there probably exist several transportation services (taxi, bus, shuttle, metro, etc.) that could satisfy his demands, but in addition to their typical characteristics (e.g., cost), they may also have other specific features (e.g., shareable, door to door, etc.). So, the user needs to ask first tourist offices or websites, or search for a mobile app about transportation in that city; he could be easily overwhelmed with information and many options, and it could be difficult for him to determine which ones are relevant according to his preferences.

So, it would be very interesting for this person to just indicate the name of the destination hotel and his preferences (for example, he could prefer to pay more to reach the hotel as soon as possible) and obtain on his smartphone the real-time location of the best possible transportation means around him. To enable this, the system would have to deal with challenges such as obtaining information about the transportation means (considering geographic information about the city) and keeping it updated, showing the results to the user, etc. The interest of a system like this is beyond doubt: currently,

[1] System for Heterogeneous mobilE Requests by Leveraging Ontological and Contextual Knowledge.

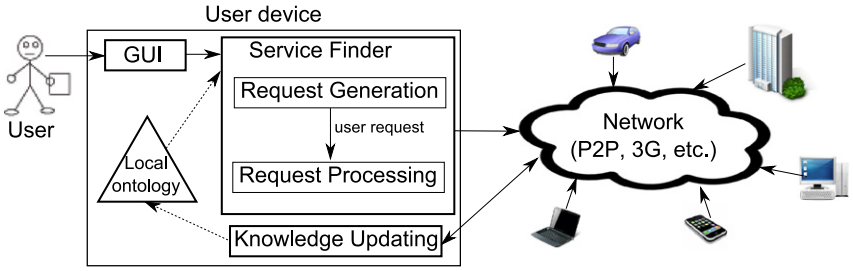

Fig. 1. High-level architecture of SHERLOCK.

although many transportation services and hotels publish their information on the Web and there exist useful services such as Google Maps, a user traveling to a certain city will probably have to deal with all the previous applications *at the same time* to try to arrange his trip.

2.2. Helping firefighting

A wildfire has broken out in a wide area of forest; the designated coordinator person is in charge of managing all the firefighters and emergency vehicles in order to suppress the wildfire. The main task of this team coordinator is to solve the problem as quickly as possible but, at the same time, keeping the team members safe. Due to the lack of a network infrastructure (the fire could have damaged it or there could be no network coverage in that area), firefighter team members usually use *walkie-talkies* to describe their location and the wildfire evolution. However, it is difficult to provide an accurate oral description of the situation while fighting a wildfire (due to smoke, geographic features, and the stressing situation). So, monitoring firefighting units in a dangerous area (and instructing them to reach a safe one), during the suppression of the wildfire, turns out to be a very challenging task for a team coordinator.

Therefore, it could be interesting for a firefighting coordinator to see, on a map displayed on his tablet, the location of all the firefighting units and the evolution of the wildfire in real-time; he would also have to keep a continuous communication with all the team members to get their last smoke and heat sensor readings. Thus, he could be able to notice changes of the wildfire that could put the life of firefighters in danger. The main challenge for a system that deals with this scenario is to monitor moving team members deployed in an environment where it is not possible to rely on a fixed network infrastructure, while detecting automatically firefighting units that could be in danger.

2.3. Common challenges

In the previous use cases, some common needs appear related to: (1) the knowledge that the system must consider, and (2) mobile computing challenges. So, on the one hand, the system must be an expert in the current user context (his/her location, device capabilities, the different kinds of elements in the scenario and their features and capabilities, the geographic information about such a context, etc.); so, it is the system, and not the user, who is in charge of knowing all the details about all the LBSs available at each location. On the other hand, the system must deal with the distributed nature of the environment (which is particularly challenging when it is not possible to rely on fixed infrastructures and ad hoc networks have to be considered) and deal with continuous request processing, scalability and fault tolerance, deployment of computations to specific geographic areas, etc.

Therefore, in the rest of this paper, we propose a system that is able to address these common challenges of the two use cases by applying semantic and distributed processing techniques. The system would be able to manage any other use case where a user is interested in obtaining information about moving objects and performing actions in highly-dynamic distributed scenarios.

3. Architecture of SHERLOCK

The main steps followed by the system to attend the information needs of a user (see Fig. 1) are: (1) *Request Generation* with the information provided by the user using a GUI, and (2) *Request Processing* over the underlying cloud of (fixed and moving) objects in the scenario. Both tasks use the knowledge of a local ontology [11] on the user device that the *Knowledge Updating* module manages and where different objects and scenarios are modeled (see Section 4).

SHERLOCK uses a mobile agent platform [12], which provides an abstraction level for the development of distributed agent-based cooperative systems, to manage the different tasks. The agents involved in the knowledge updating, request generation, and request processing, and the tasks they carry out, are explained in Sections 4–6, respectively. We include here a summary of these agents.

- *Ontology Manager (OM)*, that shares and integrates new knowledge, obtained from other objects, into the local ontology on the user device.

- *ADUS*, that generates user interfaces for applications, in a context with heterogeneous devices, considering their features.
- *Alfred*, that stores as much information as possible about the device and the user, such as user preferences, technical capabilities of the device, previous user requests, etc.
- *User Request Manager (URM)*, that helps the user to generate a request that defines his/her information needs using ontology-guided mechanisms.
- *User Request Processor (URP)*, that continuously processes the user request, with the help of Tracker agents, and returns the results to the URM.
- *Tracker*, that continuously retrieves data from target objects inside its assigned relevant area, with the help of Updater agents.
- *Updater*, that accesses the data from the target objects inside the relevant area. The information obtained is communicated to its Tracker.

In our prototype, we use the mobile agent platform SPRINGS [13] which offers RPC-based synchronous communications to support the cooperation among agents on the same device or on different devices.

4. Knowledge updating: modeling moving objects and scenarios

In our system, we consider that a moving object is both the mobile device and the physical object that it acts on behalf of in the system. For example, a person is an object in our system as long as he/she can be detected and is attached to the device that impersonates him/her in the system.[2] This allows us to consider all the participating objects as equal peers and enables flexible configurations in which every object can be a potential processing node.[3] To be part of the system, an object has to share some information about itself, that is modeled in an ontology: the context that defines the object, its geographic location, a list of sensors attached to the object, device characteristics (battery level, processor load, etc.), and the context that defines the functional capabilities of the object. Moreover, any user could create and share any other knowledge about objects, scenarios, services, etc., modeled in ontologies.

In SHERLOCK the user device knowledge is limited to its *experience*. The device starts with a basic OWL ontology containing the user's common knowledge (device technical capabilities, user name, etc.) and the basic terms to define a new LBS: concepts such as "Service", "Provider", "Parameter", "Area", and properties such as "hasParameter", "hasProvider", etc. (see Fig. 5 in Section 7.1 where basic terms are in bold). Every time it meets a new device, both objects *learn* from each other: they share their ontologies and integrate the new knowledge they find into their own local ontologies.[4] This automatic knowledge discovery mechanism is particularly interesting when the user (and his/her device) travels to areas where new or different LBSs are available; so, when traveling to other countries moving objects learn about them. For example, the device of a user residing in Zaragoza (Spain) knows that the city has taxi, bus, and tram transportation services; when this user (device) travels to New York (USA), it does not know that a metro service is available there but, as soon as it meets a *new yorker* device, that information will be shared and integrated into its local ontology. In this way, SHERLOCK alleviates the user from knowing about the vast amount of LBSs around the world.

The ontological definition of the objects and the use of a reasoner [14] based on Description Logics (DL) [15] enables the system to infer information about the objects that a user device discovers.[5] For example, SHERLOCK can reason that an object that has wheels, carries passengers, and moves along a road can be classified as a vehicle (and so provides a transportation service). Besides, the location of the objects ranges from the most precise possible (such as GPS coordinates) to more abstract locations such as neighborhoods, cities, or fare zones. So, the system also stores geographic information modeled in the local ontology. In this way, the system has knowledge and is able to reason about the neighborhoods belonging to a city, the fare of each zone, routes (roads, streets), etc. For an example of these ontologies see the fragment of the ontology used in our prototype of SHERLOCK in Fig. 5 in Section 7.1.

The knowledge of the system is managed by the following agent.

OM (Ontology Manager) *is a static agent that performs the following tasks.*

- Sharing knowledge with OMs situated on other objects.
- Integrating new knowledge into the local ontology on the user device [16]. For this task it uses a DL reasoner.

The OM is the agent in charge of managing the local ontology on the user's device. One of the main tasks of this agent is to keep the knowledge on the local ontology updated as a result of the interaction of its device with other devices. There exist two situations in which the protocol to update the local ontology is triggered.

---

[2] The right association between the device and the person could be guaranteed by external security mechanisms (e.g., see the *eGo project* at http://www.ego-project.eu/). Notice also that the same user could be associated with several devices (e.g., his/her laptop and smartphone).

[3] We are aware that privacy issues affect many elements in our architecture. Cryptography for protecting sensible information and schemas based on digital signatures/certificates for authentication can be used, although this problem is out of the scope of this paper.

[4] Only knowledge endorsed with trusted certificates will be considered.

[5] The use of a DL reasoner enables the system to detect contradictions between the existing and new knowledge and in that case to take a conservative approach.

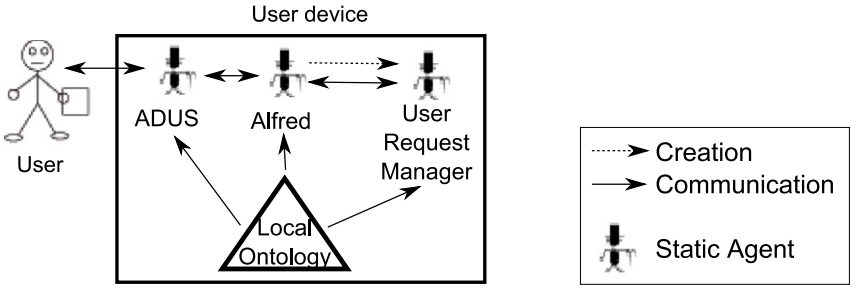

Fig. 2. Agent network involved in generating a request.

1. The OM continuously broadcasts a message asking for knowledge about services concerning the current location of the user, to keep its knowledge up-to-date. The OM residing on each device that receives this message consults its local ontology to return the relevant knowledge as an answer. To extract this relevant knowledge, OMs apply ontology modularization techniques [17]. The size of the knowledge extracted depends on the current capabilities of the devices and the current communication status.
2. When the user shows his/her interest in a specific location by clicking on a map, the OM broadcasts a message to other devices asking for knowledge related to that location, to update its current knowledge about such a location. This process is performed in parallel with the processing of user requests.

An OM integrates the new knowledge received with its local ontology by using well-known ontology integration techniques [16]; specifically, our current prototype uses the ideas presented in [18], but any other ontology matching technique could be used. In this way, an OM will "merge" terms similar enough (synonyms) and establish "is-a" relationships between subsumed terms (hyponyms). Thus, for example, the user device that receives the ontological context that defines a "shuttle" uses ontology matching techniques to discover if this knowledge is already known; otherwise, it is integrated into the local ontology. So, the system could infer that "shuttle" is a concept related to the already-known concept "transportation", and so it will be considered in transportation requests.

The OM uses a DL reasoner to manage the local ontology of the device. Having local reasoners on the devices of the users enables SHERLOCK to manage knowledge even when network disconnections make it impossible to rely on third-party devices/computers to carry out the reasoning. We studied the use of DL reasoners on Android in [19] and concluded that, although using well-known DL reasoners on current Android devices is obviously slower than in PCs, it is efficient enough (in Section 7 we show our Android prototype using a DL reasoner on the mobile devices). Moreover, thanks to the use of mobile agents it is possible to even send an agent to another device to perform the reasoning, if the user device is currently overloaded.

5. Request generation: selecting an LBS

In the following, we describe the static agents that take part in the request generation process and which reside on the user device (see Fig. 2).

ADUS (ADaptive USer) *is an interface generation agent whose goal is*

- generating user interfaces for applications, in a context with heterogeneous devices, considering their features.

ADUS is the only agent that generates GUIs for the user (adapting the ideas explained in [20]). Whenever an agent wants to interact with the user, it will communicate to ADUS a GUI specification using a GUI layout language (such as *Android XML Layouts*[6] or *XUL*[7]). ADUS is in charge of generating the specified GUI for the user device (considering its CPU speed, screen size, capability to display images or play sounds or movies, etc.) transparently to the agent that provided the GUI specification.

Alfred *is a static agent that performs the following tasks.*

- Storing as much information as possible about the device and the user, such as user preferences, technical capabilities of the device, previous user requests, etc.
- Creating a *User Request Manager* agent for each request of the user.

Thanks to the knowledge about the user that Alfred stores in the local ontology, it will be possible to infer interesting information about previous user requests when generating a new request. For example, for a user that wants to obtain transportation means, Alfred could infer that "buses" should be prioritized over "taxis" as they are "public transportations"

6 http://developer.android.com/guide/topics/ui/declaring-layout.html.
7 http://developer.mozilla.org/en/XUL.

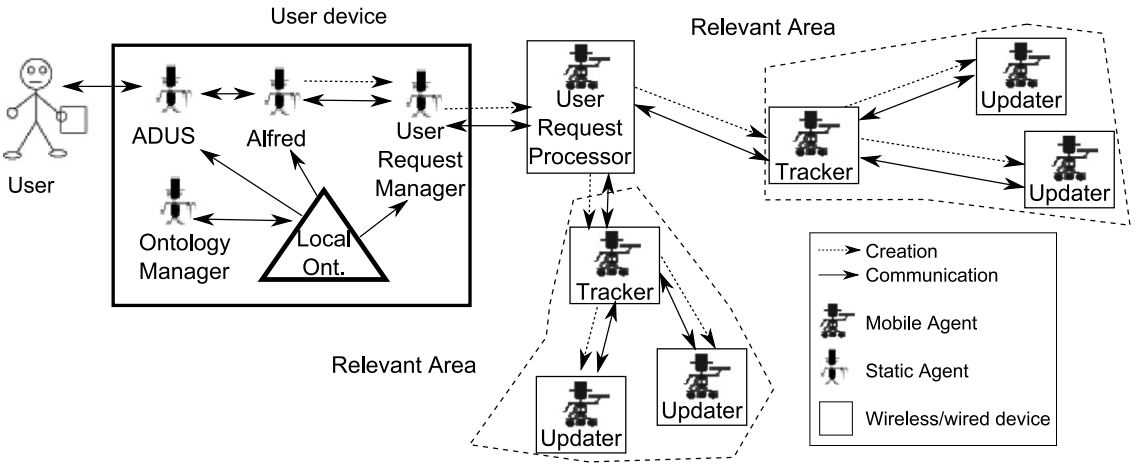

Fig. 3. Agent network deployed to process a request.

that are cheaper than "private transportations" and in previous similar requests the user usually chose to spend little money. Alfred then creates a User Request Manager and provides it with this knowledge, the context of the user, and the information that the user has input to the system (e.g., his/her interest in transportation means). The agent creation is done transparently by the mobile agent platform, which hides the implementation details from the developer of mobile agent-based applications.

URM (User Request Manager) *is a static agent that performs the following tasks.*

- Helping the user to generate a request that defines his/her information needs using ontology-guided mechanisms.
- Creating a *User Request Processor* agent (see Section 6) that will be in charge of the user request processing.

When the user expresses his/her interest in a certain geographic area (by clicking on it on a map displayed on the user device), then the URM asks the local ontology to obtain the services that are related to that specific geographic point (i.e., LBSs). Then, when the user finally selects one of these services, the URM presents (through the ADUS agent) a GUI with an input form to express the user preferences for that service. With this information the URM infers, with the help of the DL reasoner, the most appropriate service providers according to the user preferences and generates the user request. The services usually selected by the user will be available as predefined forms (e.g., SOS button, *k*-nearest gas stations, etc.).

To process the user request the URM creates a User Request Processor agent (see Section 6) that will be in charge of the user request processing. If the communication with this agent fails, the URM could try to estimate the results to provide an answer for the user (e.g., it could estimate the location of a previously-retrieved object using its last known location, direction, and speed).

6. Request processing: obtaining an answer

In this section we will explain the agents involved in the processing of a user request. To reach the target information for the user request, SHERLOCK will create a network of mobile agents [9] (see Fig. 3), which are programs that execute in contexts called *places* and can autonomously travel among devices in the scenario, resuming their execution on the destination.

In SHERLOCK mobile agents communicate to each other directly. When there is no direct communication (due to the lack of a fixed network infrastructure, which makes an ad hoc network the only possible option), SHERLOCK uses an underlying multi-hop ad hoc routing protocol [21] to allow its agents to communicate with each other; this low-level communication protocol is beyond the scope of this paper. However, we would like to stress the important role of the mobile agent technology in SHERLOCK, which is to balance the computing load and minimize the network latency. For this task, each mobile agent considers any object in the scenario as a potential processing node, so it continuously evaluates the appropriateness of the current device where it executes as well as the devices in its surroundings. As a result of this evaluation, four different decisions can be taken by the mobile agent.

1. To remain on the same device, when both the computing and communications tasks do not suffer from important delays.
2. To move to another device, where the performance and communication are expected to be better than on its current device.
3. To create a new helper mobile agent, when it detects a situation that it cannot solve alone (for example, when the agent needs to monitor too many objects or a too large area); this new helper mobile agent will be created on the most appropriate device.
4. To command a helper agent to finish its execution, when other agents are executing the same tasks more efficiently.

This adaptive behavior of the hierarchical mobile agent network, where each mobile agent executes on the device that minimizes the computing and communication delays, is specially important in highly-dynamic environments where new devices can appear/disappear or change their capabilities (e.g., a laptop can use Wi-Fi and then change to a wired connection). As the capabilities of a device (e.g., processor load, remaining battery time, communication range, etc.) will be considered when choosing a destination, fixed devices with wired communication will be preferred if available.
In the following we describe the mobile agents involved in the processing of the request.

URP (User Request Processor) *is a mobile agent with these goals.*

- Continuously processing the request and returning the results to the URM.
- Creating one *Tracker* agent for each relevant area involved in the user request. It is also in charge of the correlation of the results provided by its Trackers.

A location-based request, by definition, has always at least an area attached (called *relevant area*) that restricts the location of *target objects* in which the user is interested; these target objects (i.e., the objects in the scenario that are of interest for the user) are defined in the user request. For example, if the user is interested in cars inside a certain geographic area then, cars are the target objects and that area is the relevant area in the user request.

Relevant areas can be static (their boundaries do not change along time, for example the area of Hyde Park in London) or dynamic (they are relative to certain *reference* moving objects and/or their shapes change in time, for example the area delimited by a vehicle fleet). Thus the URP creates one Tracker agent for each relevant area in the user request, providing each of them with the area boundaries (if it is static) or with its ontological definition of the area to allow the continuous recomputation of its boundaries (in case of a dynamic area).

The URP creates more than one Tracker when it must monitor different relevant areas, or when one relevant area dynamically becomes too large for just one Tracker or gets divided into two different relevant areas.[8]

Tracker *is a mobile agent that performs the following tasks.*

- Continuously retrieving data from target objects inside its assigned relevant area.
- Maintaining a network of *Updater* agents to cover its assigned relevant area (i.e., to monitor target objects inside it). It also correlates the results provided by its Updaters.

The Tracker is in charge of monitoring a relevant area (that can be static or dynamic), as mentioned before. For example, a Tracker agent could monitor certain target objects inside a park, which is a static relevant area whose geographic limits (or *location granule* [22]) were assigned to such a Tracker. Besides, another Tracker could monitor certain vehicles in a traffic jam (dynamic relevant area) by assigning to this Tracker a list of police motorbikes (the reference objects) whose location indicates the limits of the traffic jam (the relevant area to monitor). In the case of dynamic relevant areas, the Tracker agents reevaluate continuously the location of the reference objects that delimit those relevant areas.

To achieve its goal, the Tracker manages an Updater agent network to monitor the target objects inside its relevant area and, in the case of dynamic relevant areas, to monitor the locations of the reference objects as well; the results obtained from those Updaters will be correlated by the Tracker and returned to its URP, continuously. The Tracker creates more than one Updater when it must monitor many objects or when a single Updater is not able to monitor some objects with the adequate frequency (due to communication delays). The Tracker keeps a table with the information of the objects that each Updater monitors and the lasts communication delays (see an example of this table in Section 7.2). When the communication with a certain object is too slow for all the Updaters that monitor it, the Tracker creates another Updater agent near that object.[9] Moreover, the Tracker can command an Updater to stop monitoring a certain object, and add it to its "black list", when other Updaters provide better communication with such an object; an Updater finishes its execution when it has no objects to monitor. In this way, each Tracker maintains its network of Updaters in a dynamic way (see Section 7.2 for an example of this behavior in the firefighting use case).

Updater *is a mobile agent that performs the following tasks.*

- Accessing the data from the target objects inside the relevant area. The information obtained is communicated to its Tracker.
- Discovering new knowledge interesting for the current request.

Updaters ask surrounding objects to check out whether they are members of the target classes and are located inside the relevant area. Those objects fulfilling the required features, and that do not belong to the "black list" of the Updater, will be returned to the corresponding Tracker (as well as the communication delays with them) in order to finally provide an answer to the user. When an Updater does not find any interesting object to report about in a certain period of time, it finishes its execution after informing its corresponding Tracker.

When an Updater communicates with an object, this object returns the (local) ontology context that describes it. It may happen that an object belongs to a class different from the target classes monitored by the Updater but that could

8 This could happen, for instance, for dynamic relevant areas delimited by moving objects.

9 We would like to remind that Updater agents, as well as the rest of the mobile agents in SHERLOCK, decide continuously which the best node in the scenario to execute their tasks is, as explained at the beginning of Section 6.

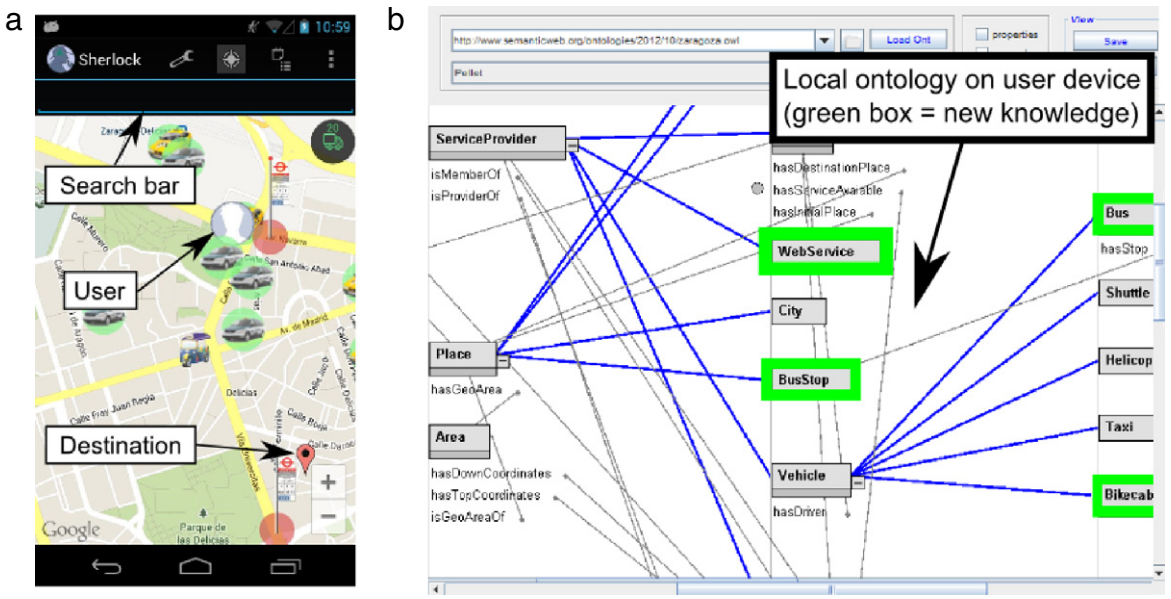

Fig. 4. GUI of the Android prototype of SHERLOCK (a) and local ontology on the device (b) for the first use case. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

be considered part of the answer anyway. For example, an Updater looking for taxis for a user that has just arrived in India could discover that a *tuk-tuk* is a vehicle defined as a private transport that carries people to a certain destination; the Updater (with the help of a DL reasoner) can discover that the *tuk-tuk* concept fulfills the kind of vehicles it is looking for, and then tuk-tuks will be considered as part of the answer, even when this vehicle class was not known by the user device that posed this user request. Indeed, new knowledge discovered by Updaters is propagated through the network of agents to enrich the local ontology on the corresponding user device.

7. Dealing with the motivating scenarios

In this section, we explain how the system deals with the two motivating use cases presented in Section 2. We have developed a SHERLOCK prototype to test the underlying ideas of the system. First, we developed a prototype running on a PC,[10] which is a Java Applet that simulates how SHERLOCK would look like when running on a smartphone (see Fig. 6(a)); this prototype uses the OWL API [23], the Pellet [24] reasoner, and the SPRINGS mobile agent platform [13]. We have recently developed an Android prototype of SHERLOCK[11] [25] (see Fig. 4(a)) that makes use of the OWL API and JFact[12] as reasoner. At the moment, we are working on porting the SPRINGS mobile agent platform to Android (we have made some initial tests with a few dozens of agents moving among three smartphones). Therefore, in the following we will use the PC prototype for the second scenario (Section 2.2).

To simulate the scenarios for tests, the prototype uses a simulator (notice that this is not part of SHERLOCK), which generates simulated moving objects, from types interesting for the services considered (i.e., taxis, buses, shuttles, etc.). These objects randomly move around the user and continuously update their location every second in a database.

7.1. First scenario: SHERLOCK for looking for transportation

In the first scenario, a user in the railway station of Zaragoza wants to find transportation that could carry him/her to "Hotel Palafox" (see Section 2.1).

1. The user types in "Hotel Palafox" in SHERLOCK search bar and clicks on the red marker (see Fig. 4(a)) to obtain the available LBSs related to hotels.
2. The User Request Manager (URM) agent deduces, after querying the local ontology on the user device and by using a DL reasoner, that an LBS called *Transportation Service* exists that has a parameter that can reference hotels[13] (see Fig. 5).

10 Available at http://sid.cps.unizar.es/SHERLOCK/PC.

11 Available at http://sid.cps.unizar.es/SHERLOCK/Android.

12 http://jfact.sourceforge.net.

13 SHERLOCK looks for services that are somehow related to the concept *Hotel* whatever the name of the property that references such a concept (we do not assume any predefined schema in the definition of services).

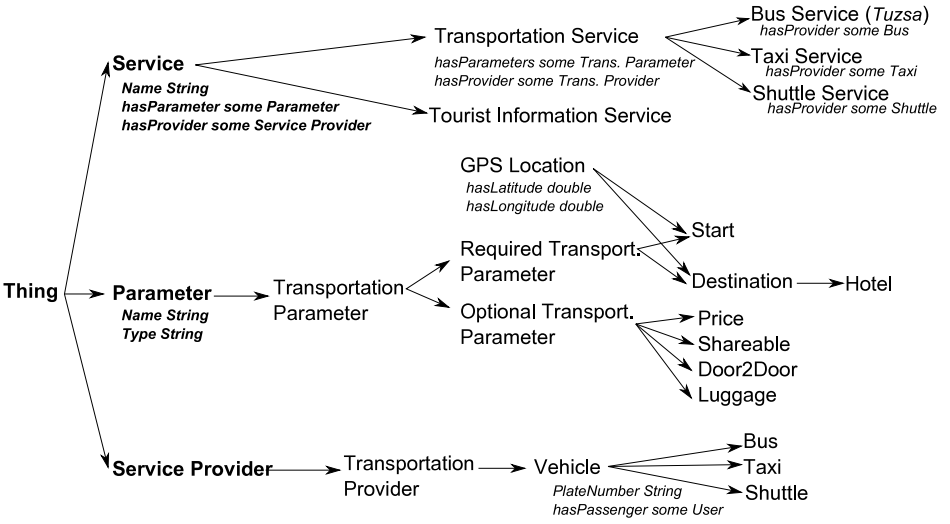

Fig. 5. Subset of the ontology for the transportation use case.

3. The user selects the *Transportation Service* and the URM obtains from the local ontology the parameters of such a service (*Price*, *Shareable*, *Door2Door*, and *Luggage*), to allow the user to specify his/her preferences.
4. The user shows his/her interest in a transport *Door2Door* (indicating that this is mandatory) that admits *Luggage*, if possible.[14] Then, the system infers that moving objects belonging to the *Taxi*, *Bus*, and *Shuttle* classes fulfill the user preferences and provide transport services. In addition, moving objects surrounding the user device share that *Tuzsa* is an instance of *Bus Transport Service* available for that specific geographic area (Zaragoza) and time, whose bus stops and schedules can be obtained from a web service.[15]
5. So, the URM agent monitors the web service for the buses and creates a network of agents to obtain taxis, shuttles, and buses located nearby with a relevant area of 1 km around the user. In the meanwhile, the Ontology Manager (OM) agent discovers that there exist moving objects classified as *Bikecab* (a subclass of *Taxi* unknown for the ontology of the user). This new knowledge enriches the user device knowledge (see green bold boxes in Fig. 4(b)) and enables the URM to infer that bikecabs also fulfill the user preferences.
6. SHERLOCK presents on the GUI the interesting objects in different colors: in green, those fulfilling all the mandatory and optional user preferences; in red, those fulfilling some optional preferences but not all the mandatory ones; the rest of the moving objects displayed fulfill all the mandatory preferences but not all the optional ones.
7. The user could click on a bus stop icon to trigger a request to obtain the remaining time for the next bus arrival. As the user does not want to wait too much, he/she finally decides to click on a taxi icon and selects its *Call Taxi* service to get to "Hotel Palafox".

Notice that the information provided by the user (a click on a map, selecting the *Transportation Service*, and filling a user-friendly form) is enough for SHERLOCK to retrieve interesting transportation for that geographic area and time, due to the use of an ontology and a DL reasoner. SHERLOCK obtains all this information from a local ontology which gets updated continuously thanks to the communication with other devices. Thus: (1) the system is decoupled from the contextual knowledge of the scenario; and (2) it adapts itself automatically to any location and service availability. Moreover, the system integrates data obtained directly from querying the moving objects in the scenario with third-party data sources (e.g., web services) specified in ontology descriptions of the service providers. In this way, if no SHERLOCK-enabled devices are located near the user (and so it is not possible to process the request with a P2P approach), his/her SHERLOCK application could use a web service to provide an answer as long as a 3G connection is available.

7.2. Second scenario: SHERLOCK for helping firefighting

In this scenario, the coordinator of a wildfire suppression in Yellowstone National Park is interested in obtaining information about eight fire outbreaks and the firefighter team (which consists of five firefighters, two firefighting trucks, and a helicopter). In this case we will illustrate this scenario by using the PC prototype of SHERLOCK.

1. The user clicks the predefined query button *Monitor fire* of the GUI (see Fig. 6(a)) that triggers the *Fire Monitoring* service. The URM obtains from the local ontology that this service requires only one parameter: a *Firefighting Team*; in this case,

[14] The use of an ontology and a DL reasoner enables the system to detect potential situations where the user preferences cannot be fulfilled by any service provider, just by checking their ontological definitions.

[15] The ontological definition of the web service (i.e., its location, how to invoke it, and what information it returns) has to be described in the ontology to enable SHERLOCK to use it.

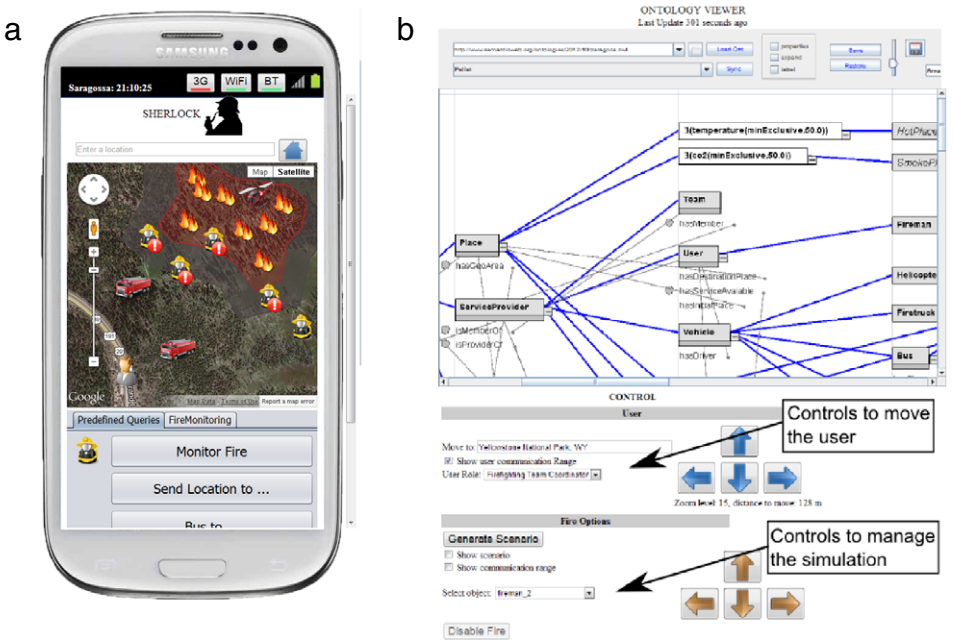

Fig. 6. GUI of the PC prototype for the fire monitoring scenario.

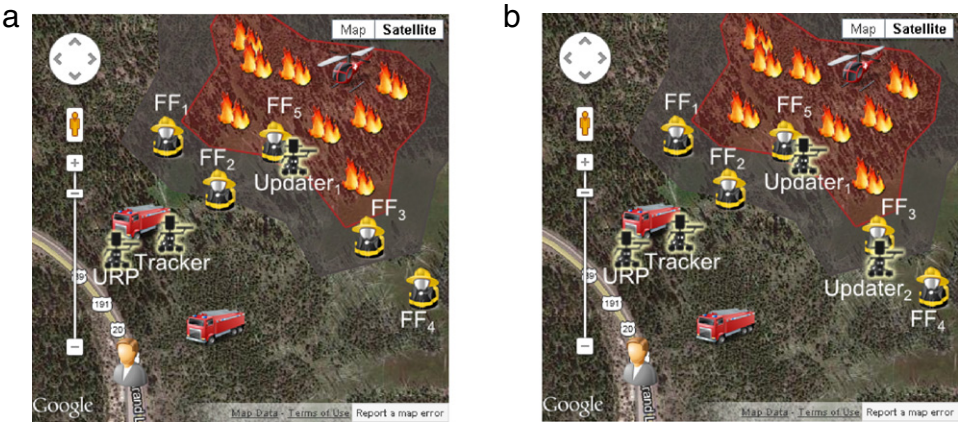

Fig. 7. Location of the mobile agents deployed to monitor a fire for the sample scenario. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the specific team members are known and stored as part of the information about the user in the local ontology. Moreover, the ontology states that this service has to monitor *Dangerous Areas*, defined as *High Temperature Area* (*hasTemperature* >50) and *High Level of CO2 Area* (*hasCO2* >400).[16]

2. The URM creates a User Request Processor (URP) agent to monitor the dynamic relevant area delimited by the location of the firefighting units. The URP moves to a truck that provides firefighters with water (see Fig. 7(a)) because its device has a powerful CPU and a high capacity battery, and then creates a Tracker agent to monitor the relevant area.
3. The Tracker, that chooses also to stay at the truck device, creates one Updater agent (*updater* 1) to obtain information from the firefighting units. Then, *updater* 1 sends the information obtained for each firefighter (the location and measures of his/her temperature and CO_2 sensors) continuously upwards through the network; the location of the firefighter units is presented to the user along with the dangerous areas computed with the firefighter sensor measures (see Fig. 7 in orange). This Updater also sends to the Tracker the communication delay with the objects (see Table 1). As *updater* 1 communicates too slow with $Firefighter_4$, the Tracker creates another Updater (*updater* 2) that will execute near $Firefighter_4$ (see Fig. 7(b)).
4. The two Updaters continue monitoring all the firefighting units and sending their information to the Tracker. The Tracker commands $updater_2$ to continue monitoring $firefighter_3$ and $firefighter_4$ because it has a better communication with them,

16 Temperatures are measured in Celsius degrees and CO_2 in ppm (parts per million).

Table 1
Table used by a Tracker agent to dynamically maintain its network of Updaters.

Reference object	Updater	Comm. delay (s)	Time stamp
$firefighter_1$	$updater_1$	0.64, 0.62, 0.68, 0.67	19:17:12
	$updater_2$	1.05, 1.12	19:17:10
$firefighter_2$	$updater_1$	0.56, 0.0, 0.0, 0.0	19:17:12
	$updater_2$	0.85, 0.81	19:17:10
$firefighter_3$	$updater_1$	0.71, 0.94	19:17:11
	$updater_2$	0.0, 0.0, 0.0	19:17:15
$firefighter_4$	$updater_1$	1.32, 1.11	19:17:11
	$updater_2$	0.43, 0.38, 0.39	19:17:15

whereas it commands $updater_1$ to stop monitoring them. In this way $updater_1$ reevaluates the appropriateness of the current device where it executes to try to stay close to $firefighter_1$ and $firefighter_2$.

5. The Tracker, by analyzing the communication delays, detects when a firefighter moves as far as to getting unreachable from the rest of the objects. Then, an alarm is generated to enable the user to react by commanding another firefighter to move closer to the last known location of the missing firefighter, in order to try to reestablish the connection with him/her.

Thus, SHERLOCK is able to manage its network of agents to dynamically adapt itself to changes even in scenarios where it is not possible to rely on a fixed infrastructure, such as the one described in this section.

8. Related work

Up to the authors' knowledge, no other work has proposed a general and flexible system based on semantics to build generic LBSs. So, we will provide an overview of contributions to some specific research areas related to our proposal.

Location-dependent queries (i.e., queries whose answer depends on the locations of certain moving objects), such as range queries (e.g., see [26,27]) and nearest-neighbor queries (e.g., see [28,29]), can be considered a basic building block of LBSs. Therefore, considerable research efforts have been invested on studying efficient ways to process them as continuous queries (see [30] for an extensive survey on location-dependent query processing). Some of the solutions proposed assume a centralized query processing environment (e.g., [31]), whereas others perform a distributed query processing using a fixed support infrastructure [7,32] or exploiting the processing capabilities of the mobile devices attached to the moving objects [6]. However, existing proposals do not solve all the challenges identified in this paper. For example, managing the knowledge about the different kinds of moving objects and their features is usually ignored and a predefined database schema (that the user must know) is assumed instead. Even though there are interesting proposals that have considered some semantic aspects (e.g., [33] proposes the management of semantic trajectories and [34] presents the concept of semantic caching of location-dependent data), they do not aim at developing a general semantics-based query processing architecture.

There are also several proposals that deal with aspects related to the case studies presented in this paper. For example, in relation to the first case study, several systems for multimodal transportation planning have been developed[17] and some works have focused on the problem of taxi searching (e.g., [2]). As an example of a context-aware system to help firefighters (second case study), [3] considers a multi-hop peer-to-peer communication model and uses a context rule engine to generate alerts, and mobile nodes carried by users combined with a fixed wireless network of sensors previously deployed, to help firefighters to determine the current fire status. However, all these proposals have been developed for quite specific scenarios where only certain types of objects and requests have to be managed. So, a general system that is able to manage all of them uniformly is missing. Moreover, even though some proposals advocate the use of semantic techniques for some tasks (e.g., in the WORKPAD project[18] ontologies are used for information integration in a mobile peer-to-peer network), they would benefit from the exploitation of semantic techniques at all levels (interpretation of users' request, knowledge modeling and reconciliation, query processing, and data integration), as we propose in SHERLOCK.

9. Conclusions and future work

In this paper, we have presented SHERLOCK, a general system that provides support for Location-Based Services that depend on highly-dynamic information and infrastructures. Moving objects collaborate by exchanging their knowledge and could become potential processing nodes at any time. Besides, we have introduced two different sample motivating scenarios that can be solved by our system; any other use case where a user is interested in obtaining information about moving objects or in asking them to perform actions in highly-dynamic distributed scenarios can also be processed by SHERLOCK. As a summary, the original contributions of SHERLOCK are the following.

17 http://ec.europa.eu/transport/its/multimodal-planners/index_en.htm.
18 http://www.dis.uniroma1.it/~workpad.

- It offers to the user all the available interesting LBSs at each moment and helps in expressing his/her information needs. In this way, it relieves the user from managing specific knowledge about LBSs.
- It supports the enrichment of the knowledge managed by discovering new information as a result of the interaction among objects.
- It uses ontologies to avoid imposing the users with a global schema. Instead, the system achieves semantic reconciliation of data sources and services.
- It deals with requests that require the continuous acquisition of data from different moving and static objects.
- It is flexible regarding the underlying network infrastructure, enabling the use of static and mobile networks.
- It continuously carries out the processing to the most appropriate nodes in order to balance the processing load and communication tasks, by using mobile agents.

We have developed and presented a prototype of SHERLOCK where we tested two use cases. We would like to highlight that the system used in both scenarios is the same and the only difference is the ontology/knowledge made available to SHERLOCK. So, just by providing it with other ontology-defined services, other scenarios will be considered. We are currently adapting the mobile agent platform SPRINGS to Android to be able to fully develop our SHERLOCK Android prototype with mobile agents.

Acknowledgments

This research work has been supported by the CICYT project TIN2010-21387-C02 and DGA FSE. Interesting discussions about moving objects took place in several meetings thanks to the support of the European COST Programme (Action IC0903 MOVE). We also thank Guillermo Esteban and Juan Mengual for their help with the implementation of our prototype, and Jorge Bobed for his ontology viewer.

References

[1] J. Schiller, A. Voisard (Eds.), Location-Based Services, Morgan Kaufmann, San Francisco, CA, USA, 2004.
[2] J.-P. Sheu, G.-Y. Chang, C.-H. Chen, A distributed taxi hailing protocol in vehicular ad-hoc networks, in: 71st IEEE Vehicular Technology Conf., VTC 2010, IEEE Computer Society, 2010, pp. 1–5.
[3] X. Jiang, N.Y. Chen, J.I. Hong, K. Wang, L. Takayama, J.A. Landay, Siren: context-aware computing for firefighting, in: 2nd Int. Conf. on Pervasive Computing, PERVASIVE'04, Springer, 2004, pp. 87–105.
[4] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, K. Wampler, Buddy tracking — efficient proximity detection among mobile friends, Pervasive and Mobile Computing 3 (5) (2007) 489–511.
[5] S. Ilarri, E. Mena, A. Illarramendi, R. Yus, M. Laka, G. Marcos, A friendly location-aware system to facilitate the work of technical directors when broadcasting sport events, Mobile Information Systems 8 (1) (2012) 17–43.
[6] B. Gedik, L. Liu, MobiEyes: a distributed location monitoring service using moving location queries, IEEE Transactions on Mobile Computing 5 (10) (2006) 1384–1402.
[7] S. Ilarri, E. Mena, A. Illarramendi, Location-dependent queries in mobile contexts: distributed processing using mobile agents, IEEE Transactions on Mobile Computing 5 (8) (2006) 1029–1043.
[8] S. Ilarri, A. Illarramendi, E. Mena, A. Sheth, Semantics in location-based services — guest editors' introduction for special issue, IEEE Internet Computing 15 (6) (2011) 10–14.
[9] D.B. Lange, M. Oshima, Seven good reasons for mobile agents, Communications of the ACM 42 (1999) 88–89.
[10] I. Chlamtac, M. Conti, J.J.N. Liu, Mobile ad hoc networking: imperatives and challenges, Ad Hoc Networks 1 (1) (2003) 13–64.
[11] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing, International Journal of Human-Computer Studies 43 (5–6) (1995) 907–928.
[12] R. Trillo, S. Ilarri, E. Mena, Comparison and performance evaluation of mobile agent platforms, in: Proc. of the 3rd Int. Conf. on Autonomic and Autonomous Systems, ICAS'07, Athens, Greece, IEEE Computer Society, ISBN: 978-0-7695-2859-5, 2007, pp. 41–46.
[13] S. Ilarri, R. Trillo, E. Mena, SPRINGS: a scalable platform for highly mobile agents in distributed computing environments, in: 2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM, 2006, pp. 633–637.
[14] R.B. Mishra, S. Kumar, Semantic web reasoners and languages, Artificial Intelligence Review 35 (4) (2011) 339–368.
[15] F. Baader, I. Horrocks, U. Sattler, Description logics as ontology languages for the semantic web, in: Mechanizing Mathematical Reasoning, in: Lecture Notes in Computer Science, vol. 2605, Springer, 2005, pp. 228–248.
[16] N. Choi, I.-Y. Song, H. Han, A survey on ontology mapping, SIGMOD Record 35 (3) (2006) 34–41.
[17] P. López-García, M. Boeker, A. Illarramendi, S. Schulz, Usability-driven pruning of large ontologies: the case of SNOMED CT, Journal of the American Medical Informatics Association 19 (2012) e102–e109.
[18] J. Gracia, E. Mena, Ontology matching with CIDER: evaluation report for the oaei 2008, in: Proc. of 3rd Ontology Matching Workshop, OM'08, at ISWC'08, 2008.
[19] R. Yus, C. Bobed, G. Esteban, F. Bobillo, E. Mena, Android goes semantic: DL reasoners on smartphones, in: 2nd International Workshop on OWL Reasoner Evaluation, ORE 2013, 2013.
[20] N. Mitrovic, J. Royo, E. Mena, ADUS: indirect generation of user interfaces on wireless devices, in: 7th Int. Workshop Mobility on Databases and Distributed Systems, MDDS'2004, IEEE Computer Society, 2004, pp. 662–666.
[21] A. Boukerche, B. Turgut, N. Aydin, M.Z. Ahmad, L. Bölöni, D. Turgut, Routing protocols in ad hoc networks: a survey, Computer Networks 55 (13) (2011) 3032–3080.
[22] S. Ilarri, C. Bobed, E. Mena, An approach to process continuous location-dependent queries on moving objects with support for location granules, Journal of Systems and Software 84 (8) (2011) 1327–1350.
[23] M. Horridge, S. Bechhofer, The OWL API: a Java API for OWL ontologies, Semantics Web 2 (1) (2011) 11–21.
[24] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz, Pellet: a practical OWL-DL reasoner, Web Semantics 5 (2) (2007) 51–53.
[25] R. Yus, E. Mena, S. Ilarri, A. Illarramendi, SHERLOCK: a system for location-based services in wireless environments using semantics, in: 22nd International World Wide Web Conference, WWW 2013, ACM Press, ISBN: 978-1-4503-2038-2, 2013, pp. 301–304.
[26] K. Rothermel, S. Schnitzer, R. Lange, F. Dürr, T. Farrell, Context-aware and quality-aware algorithms for efficient mobile object management, Pervasive and Mobile Computing 8 (1) (2012) 131–146.
[27] M.A. Cheema, L. Brankovic, X. Lin, W. Zhang, W. Wang, Continuous monitoring of distance-based range queries, IEEE Transactions on Knowledge and Data Engineering 23 (8) (2011) 1182–1199.

[28] W.-S. Ku, R. Zimmermann, Nearest neighbor queries with peer-to-peer data sharing in mobile environments, Pervasive and Mobile Computing 4 (5) (2008) 775–788.
[29] R. Benetis, S. Jensen, G. Karĉiauskas, S. Ŝaltenis, Nearest and reverse nearest neighbor queries for moving objects, The VLDB Journal 15 (3) (2006) 229–249.
[30] S. Ilarri, E. Mena, A. Illarramendi, Location-dependent query processing: where we are and where we are heading, ACM Computing Surveys 42 (3) (2010) 1–73.
[31] H. Hu, J. Xu, D.L. Lee, A generic framework for monitoring continuous spatial queries over moving objects, in: ACM SIGMOD Int. Conf. on Management of Data, SIGMOD'05, ACM, 2005, pp. 479–490.
[32] J. Jayaputera, D. Taniar, Data retrieval for location-dependent queries in a multi-cell wireless environment, Mobile Information Systems 1 (2) (2005) 91–108.
[33] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, K. Aberer, SeMiTri: a framework for semantic annotation of heterogeneous trajectories, in: 14th Int. Conf. on Extending Database Technology, EDBT 2011, ACM, 2011, pp. 259–270.
[34] B. Zheng, D.L. Lee, Semantic caching in location-dependent query processing, in: 7th Int. Symposium on Advances in Spatial and Temporal Databases, SSTD'01, Springer, 2001, pp. 97–116.