A Knowledge-Based Approach to Enhance Provision of Location-Based Services in Wireless Environments

ROBERTO YUS[1], CARLOS BOBED[2,3], AND EDUARDO MENA[3]
[1]University of California, Irvine, Irvine, CA 92697, USA
[2]everis / NTT Data, 28046 Madrid, Spain
[3]University of Zaragoza, 50018 Zaragoza, Spain

Corresponding authors: Roberto Yus (ryuspeir@uci.edu) and Carlos Bobed (cbobed@unizar.es)

ABSTRACT Location-Based Services (LBS) are attracting a great interest with the fast expansion of mobile computing nowadays. These services use the user location to customize the offered information. However, most of those services are designed for specific scenarios and goals with implicit knowledge about the application context. As a consequence, hundreds of them are available (even with the same purpose). So, it is difficult for users to choose the most suitable service as they are in charge of knowing/finding the services which will be interesting for them, and handle the information that such services need. In this paper, we present an approach to handle LBS for mobile users which relieves them from knowing and managing the knowledge related to such services. This approach consists of a proposal for the modeling of such information as ontologies, which are handled by an agent-based architecture. Also, we propose to maintain updated the knowledge each mobile device contains by leveraging the exchange of information with others. For accessing the local knowledge, we present an SPARQL-like query language which avoids the ambiguities of natural language. Finally, we propose an approach to translate the user information needs into formal requests expressed in this query language, which could be later processed against the knowledge repositories to obtain the results the user needs.

INDEX TERMS Location-based services, mobile computing, Semantic Web.

I. INTRODUCTION

In the last few years, the technological advances we have witnessed regarding mobile devices have enabled new kinds of information systems and paradigms that were not feasible before. To this extent, the plethora of sensors that mobile devices (e.g., smartphones and tablets) currently include, along with their increasing computational power and battery lifetimes, have turned each mobile device into a potential and capable data capture and processing node of a massively distributed information system. Among these sensors, location mechanisms have proved to be specially important as they enable the development of Location-Based Services (LBS) [1], which provide value added by considering locations of the mobile users to offer customized information.

The associate editor coordinating the review of this manuscript and approving it for publication was Kaigui Bian[ID].

Due to the pervasiveness of mobile computing in our daily lives, there are currently hundreds of LBS available for users (many of them with the same purpose). In fact, current LBS are usually tailored to specific scenarios, where both services and data are completely attached to predefined and non-evolving schemas. Besides, they usually work with implicit context knowledge (e.g., possibly hardcoded within the application/service), which contributes to their non-reusability. For example, LBS for taxi searching [2], helping firefighting [3], detecting and recommending nearby friends [4], or multimedia retrieval in sport events [5] have been presented, among many others. Therefore, it is difficult to handle the information about all the LBS which could be interesting for us, and to select the most appropriate service matching our information needs.

Some ad hoc solutions (e.g., [6], [7]) and even general architectures (e.g., our SHERLOCK architecture presented in [8], [9]) have been proposed to provide users with LBS.

For instance, the latter architecture has been applied in different scenarios, such as finding transportation for a user and coordinating a team of firefighters suppressing a wildfire (as presented in [8], [9]), handling an emergency situation caused by a traffic accident (presented in [10]), or helping a technical director in the live broadcasting of a sport event (presented in [11]). These systems require an appropriate management of information related to LBS, which might be interesting for users regarding their current context. However, due to the heterogeneity of LBS and user information needs, handling this information is not trivial.

In this paper, we propose an approach to handle LBS and user information needs that is grounded in semantic technologies, such as ontologies [12] and semantic reasoners, for the representation and handling of the knowledge related to LBS. In particular, we have developed and integrated our proposal within the SHERLOCK system [8]. Our approach utilizes an agent-based architecture to distribute different tasks, such as the maintenance of the knowledge and the interaction with users to obtain their information needs. In summary, our approach presents the following benefits:

- It includes an ontological model for modeling the knowledge about relevant LBS, their domain, context, and interesting services. This model is general enough to cover the heterogeneity of most LBS and can be easily extended.
- It maintains the knowledge about LBS updated by leveraging the communication between devices. Ontologies defining LBS, based on the previous model, are shared and integrated by software agents enabled with semantic capabilities.
- It provides means to access the knowledge about LBS through a query language which we defined based on the standard semantic query language SPARQL.[1]
- It guides the user in the process of selecting the LBS that best fits her needs. Then, it formalizes the user information needs into a user request which, for instance, might involve queries in the previous language avoiding the ambiguities of natural language.

Therefore, we can summarize the main contributions of this paper as follows:

- An architecture for the discovery of LBS based on automatic knowledge sharing across mobile devices in the scenario.
- A new query language combining elements to handle geospatial and semantic definitions, which allows to formally describe the user requests.

Note that the benefits of our approach do not restrict to SHERLOCK: Systems providing LBS to users can benefit from this management of the knowledge about such services and the user information needs. These systems could process the formal requests generated, for example against external data sources, to retrieve the desired data.

[1]SPARQL 1.1 Overview, https://www.w3.org/TR/sparql11-overview/, last accessed on 20th April, 2020.

The rest of the paper is as follows. In Section II, we present a general overview of our approach including a description of its features and the agents that support them. In Section III, we describe how the knowledge about the user, her device, and services that might be interested for her, is defined. In Section IV, we present the formal query language we designed to enable accessing this knowledge, and the agent in charge of processing it. In Section V, we detail the agents involved in updating the local knowledge related to LBS in a mobile device enabled with our approach by interacting with other devices. In Section VI, we present the agents in charge of interacting with the user to help her select the LBS that would fulfill her information needs. In Section VII, we present how our proposal is applied to different use cases motivating our work. Finally, related works and conclusions are presented in Section VIII and Section IX, respectively.

II. OVERVIEW OF THE SYSTEM

In our approach, in order to manage the information about services which might be relevant for a user, we advocate for a distributed architecture where each device is an independent node and is responsible for: managing their own knowledge, updating it, and integrating new knowledge by collaborating with the rest of the devices. Local interactions and knowledge sharing is the key to keep each device's local knowledge updated without having to rely on a preexisting infrastructure. For instance, let us imagine a user which arrives in a foreign city. Despite having prepared the trip previously, she does not have any current information about the different services which might be available locally (e.g., she might have downloaded a map, but she does not have information about services offering transportation). In this scenario, instead of relying on a centralized external server, the devices of local people could automatically share with her device all the required information, thus, making her device able to fulfill her current information needs. To do so, the main two tasks that every device must perform in our approach are:

- *Knowledge acquiring and updating*: A device is continuously sharing knowledge with its neighbors about both available services (along with the required background knowledge to understand them) and their contexts. The information received is integrated within the local knowledge.
- *Request generation*: Using its locally integrated knowledge, each device is able to provide users with the set of available context-relevant services, helping and guiding her to express her information needs.

In the following section, we show the agent-based architecture for mobile devices we designed to handle the previous two tasks.

A. INNER ARCHITECTURE OF DEVICES

In our approach mobile devices collaborate to serve their users' needs. Figure 1 shows the inner architecture of each device as well as their peer-to-peer collaboration. The three main features of a device incorporating our approach are:
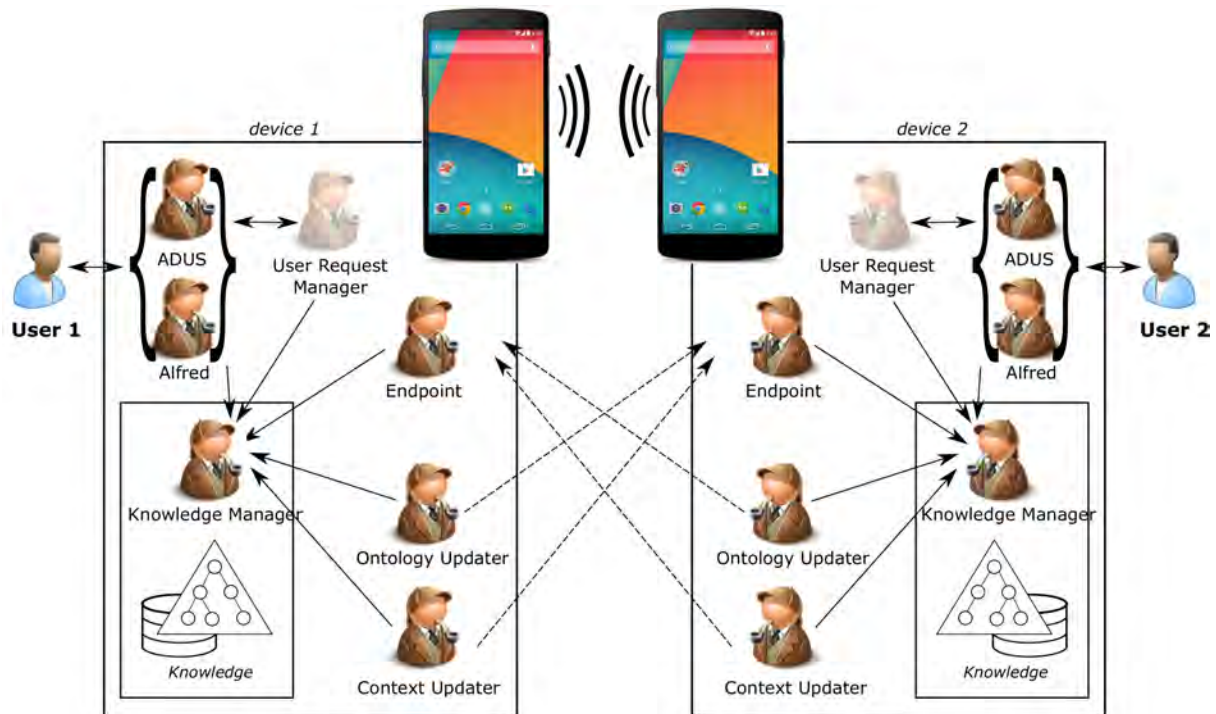
FIGURE 1. Agents to interact with a user and to manage knowledge in our approach.

- It manages local knowledge which comprises information about the user, the device, available services, and the environment in general (see Section III). This knowledge is exploited to offer the user services and mechanisms to express her needs, and is managed by the following agents:
  - *Knowledge Manager (KM)*, which is in charge of managing the knowledge stored in the device and process queries to access it.
  - *Endpoint* agent, which provides access to the knowledge and processing capabilities of the device to external agents and applications.
- It maintains the local knowledge continuously updated through the interaction with other devices. This process is handled by the following agents:
  - *Ontology Updater (OU)*, which shares and integrates new knowledge related to services, obtained from other objects, into the local ontology on the user device.
  - *Context Updater (CU)*, which specializes on updating and inferring knowledge about the user and the context of her device.
- It interacts with the user to provide her with the available services, which might be interesting depending on her context. The interaction is handled by the following agents:
  - *Alfred*, which specializes on interacting with the user and stores as much information as possible about these interactions.

  - *ADUS*, which generates graphical user interfaces (GUIs), adapted to the user profile and device capabilities, by rendering a GUI description provided by incoming agents that want to interact with the user.
- It captures the user information needs and translates them into a formal request. This task is handled by the *User Request Manager (URM)* agent, which is created on demand and uses ontology-guided mechanisms to generate, with the help of the local knowledge, a request that defines the user needs.

In the following sections, firstly, we will explain how the knowledge used by our system is modeled (see Section III). Then, we will explain how this knowledge can be accessed by presenting the query language we designed and the agents in charge of processing it, i.e., *KM* and *Endpoint* agents, (see Section IV). Afterwards, we move onto how our approach keeps the shared knowledge updated by interacting with other devices, and the agents involved in this process, i.e., *CU*, and *OU* agents, (see Section V). Finally, we will deal with the interaction with users, explaining *Alfred* and *ADUS* agents, and the translation of the user needs into a formal request by the *URM* agent (see Section VI), and how our approach is applied to the different motivating use cases (see Section VII).

III. KNOWLEDGE MODELING
Our approach uses ontologies [12] to model information about the user, her device, the different services she can use, and the scenarios around her. These ontologies are

TABLE 1. Information stored about the context of the device.

Property	Description	Example
Device model	The specific device which will be used to obtain information about its features.	Google Pixel 3
Available battery	An estimation of the remaining battery time, which might be infinite if the device is not running on batteries.	2 hours
Available processor	Percentage of processor that is available at the moment.	58%
Available memory	Amount of memory that is reserved to our system.	550 MB
Available storage	Amount of persistent storage that is reserved to our system.	2 GB
Available bandwidth	The bandwidth available for each wireless interface.	<4G, 2Mbps>
Coverage area	The coverage area that the device can see for each wireless interface. It can be static (pre-known) or dynamic (updated in every data refreshment).	<WiFi, 80m>
Sensor readings	Raw data extracted from the different sensors on the device.	<-0.06, 0.05, 9.61, 2019-02-24 19:25:35, Accelerometer>

represented using OWL,[2] de facto standard language to implement expressive ontologies in the Web enabling the definition of complex knowledge. Moreover, as OWL has formal semantics based on Description Logics [13] (DL), it is possible to perform several reasoning tasks to deduce implicit knowledge (i.e., logical consequences of the knowledge in an ontology) using semantic reasoners (i.e., DL reasoners[3]).

In addition to the OWL models, part of the factual data which are more prone to change dynamically is stored separately using a database manager as we will explain in Section IV-B1. This is done to efficiently manage semantic data that comprises both static and volatile knowledge [15]. For instance, information such as the current capabilities of the device (e.g., current battery available) or the GPS location of the user are highly dynamic whereas information about services is less prone to changes. Therefore, the former is stored in a database whereas the latter is modeled in an ontology.

Depending on the subject being modeled, the knowledge handled by our system can be broadly classified into four different categories (see Figure 2): *user context*, *device context*, *service definitions*, and *scenarios*. However, to improve readability and despite being a continuum, we adopt a higher-level separation of the knowledge which takes into account the purpose that the knowledge is used in our system for: *contextual knowledge* (including user context and device context), and *user request knowledge* (including services and scenarios).
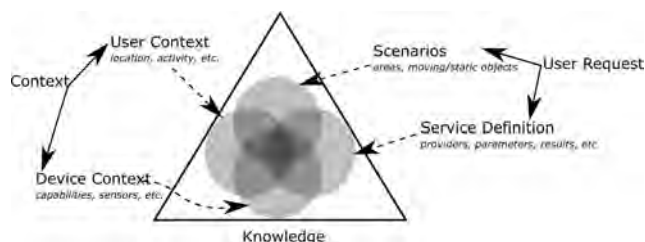
FIGURE 2. Different knowledge managed by a device.

[2]OWL Web Ontology Language, http://www.w3.org/TR/owl-primer, last accessed on 20th April, 2020.
[3]We showed that using semantic reasoners locally on current mobile devices is feasible in [14].

A. CONTEXTUAL KNOWLEDGE

We propose to manage information about the context of users and their devices to infer different aspects of their status, and use such inferences to: 1) perform a context-aware service provision, and 2) help in protecting their privacy when sharing information with other devices.

The device context (see Table 1) includes the features of the device along with a snapshot of its current capabilities. This information can be used to, for example, determine if the communication with other devices has to be limited to drain less battery.

The user context (see Table 2) includes information about the profile of the user and her current context according to the broadly adopted definition by Abowd *et al.* [16] where context is split into ''primary context pieces'' (i.e., identity, time, location, and activity) as well as ''secondary context pieces'' (i.e., pieces of context related to the primary context pieces, e.g., a user's phone number can be obtained by using the user's identity).

Notice that location is probably the most important context piece that the system has to manage as it is essential for LBS. The notion of location in our system includes both the position of an object (i.e., its coordinates) as well as the place where the object is (i.e., its geographic area). The hierarchical structure of the information stored about the context of a user makes our approach able to use different granularities of context pieces depending on the situation. For example, the location of a user in our system can be viewed from her coordinates to the building, the city level, or the region level where the user is in. Moreover, sharing the location of the user's device allow us to estimate its coverage area [17], which is important to process location-based queries in a distributed way.

B. USER REQUEST KNOWLEDGE

We propose a model based on a basic schema, which can be extended if needed, to define services and make our system aware of the available functionality. This schema comprises knowledge about both the definition of such services (e.g., which parameters they receive, how they are invoked, the type of the result -if any-, etc.), and the different entities

TABLE 2. Information stored about the context of the user.

| Property	Description	Example
Object class/es	The object is an instance of a class which is shared with other devices.	Person, firefighter.
Location	The physical position of the object which comprises the GPS coordinates as well as hierarchy of places.	<40.7588559, -73.9853107>, Times Square
Mobility	Whether the object is a moving object or not. A mobile object with a maximum velocity of 0 is not the same as an object that cannot move at all.	Yes
Maximum speed	When dealing with moving objects, the maximum velocity can be used to estimate positions and make the system more robust against communication failures for example.	5kmh
Direction	The direction of movement of a moving object which can be used, for example, to estimate future positions.	096°01′18″
Extent	The area (2D) or volume (3D) that the object occupies physically.	A 2D circle with a 1 meter diameter
Activity	The activity that the user is performing.	Walking
Secondary context	Pieces of information related to the activity and location of the user.	Walking towards Central Park
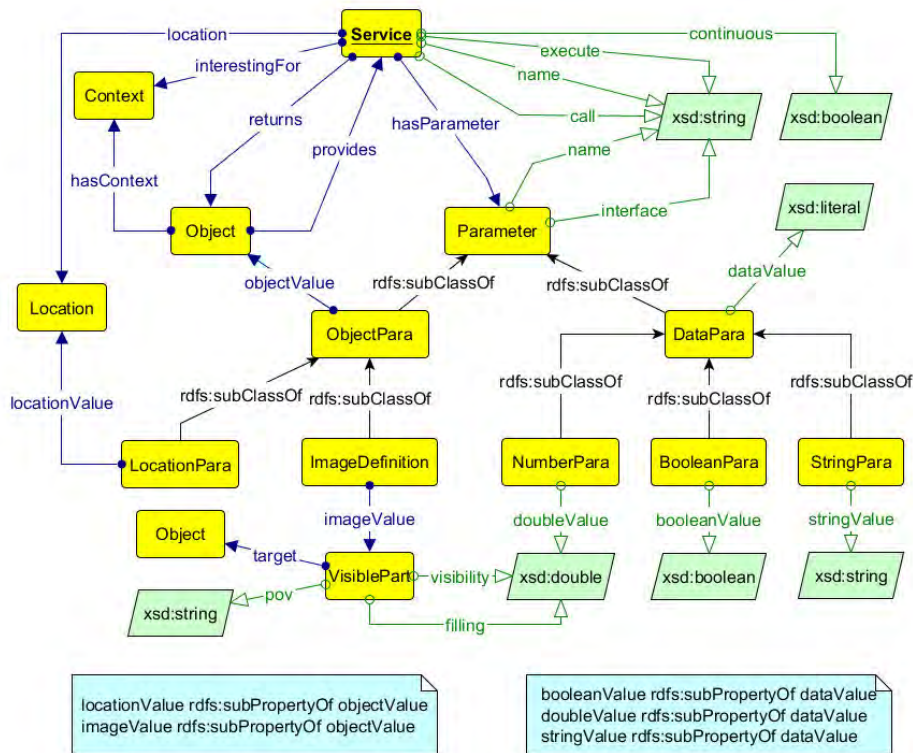
FIGURE 3. Basic ontology for service modeling.

needed to completely define their semantics (e.g., if we define a service to look for means of transport, in the ontology, the entity *meanOfTransport* should appear). Our model has been designed to enable service definers (users and/or companies which want to incorporate a new service into devices using our approach) to easily define LBS and does not aim at matching directly services as in classic Semantic Web Services approaches (see Section VIII-B for a further discussion on this issue); but it relies on integration of the shared schemas to discover new instances with similar functionality.

In our model (see Figure 3), services are instances of the *Service* class in the ontology. Specializing this *Service* class, the service definers can arrange services into *families of services* which are used to group those sharing functionality. Regarding the model, this implies that they are services which belong to the same class, share functionality, and return compatible results. For instance, imagine a service to find buses (returning the location of buses) and another one to find taxis (returning the location of taxis). These services can be created as instance of the *Service* class or could be

grouped as instances of the family of services *Find Transports* as both return transportation means. In fact, our approach requires that service definers select an existing family for their newly added services (or define a new one, extending the vocabulary).

In addition, the following properties are used to define a service (see Figure 3):

- *returns*: To model the type of objects the service will obtain (e.g., taxis in the previous example).
- *hasParameter*: To define parameters of the service that are required for its functionality or that can be used to impose constraints over the results. These parameters can take numbers, boolean values, strings, and even other objects as values. Further types can be incorporated by extending the *Parameter* class. For example, we included a new parameter *ImageDefinition* needed for a service to find images fulfilling certain constraints. A particular type of parameter is the location for LBS, which can be assigned to the service through the *location* property.
- *interestingFor*: To know which services are relevant to a particular user's situation. This property has to be populated by the user or company who is modeling the services, deciding whether that particular service will be interesting for a certain context or not. In this case, the context for which the service is interesting has to be defined using its attached activity and/or location (e.g., the service to find monuments in New York could be defined as interesting for contexts whose activity is "tourism" and whose location is "New York").
- *continuous*: To define whether a service has to be continuously evaluated. For example, a service to ask a camera to take a picture might have to be evaluated once only, whereas a service to obtain taxis near the user is expected to be continuously reevaluated to obtain updated results until the user is not interested in taxis anymore.

Apart from their ontological classification, services defined following this model can further be classified attending to the way they are processed into three types of services:

1) *Querying services*: Services which provide functionality to find objects specified using our query language (see Section IV-A). The definition of such services in our model must comprise the kind of information that the service will obtain as a result (e.g., a service to find pictures will return pictures) and its parameters (if any). As an example, Figure 4 shows the definition of services of this type to obtain transports. First, the concepts *Find Transports*, *Find Taxis*, and *Find Buses* represent families of services with properties to relate them to the information they return (*Transports*, *Taxis*, and *Buses*, respectively) and their parameters (which have been defined for the family *Find Transports* and inherited by the other two families). Then, the actual services are created as instances (*FindBusesService* and *FindTaxisService*).
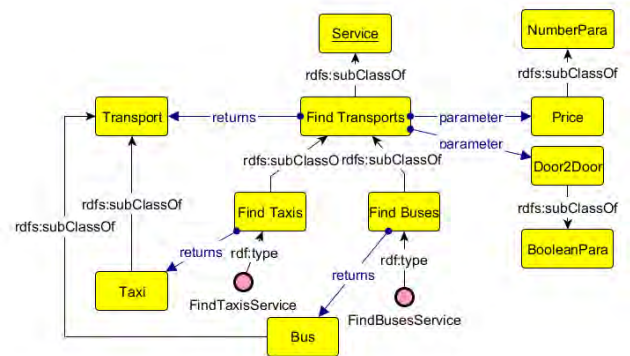

FIGURE 4. Example of the definition of a query service to obtain transports.

2) *Invoked services*: Services provided by a particular provider object. This kind of services includes both third-party external services and services offered by other devices (which might involve notifying or interacting with another user). The definition of this kind of services is extended with information about: 1) their provider (via *provider* property), 2) the access mechanism to be used (via *call* property), and 3) for those which require user's interaction, an specification of the graphical interface to be used, which will be used by the ADUS agent to interact with the user. Figure 5 shows a service to take pictures that is provided by devices equipped with cameras, and the service of the Metropolitan Transportation Authority of New York which returns the location of buses through a web service.
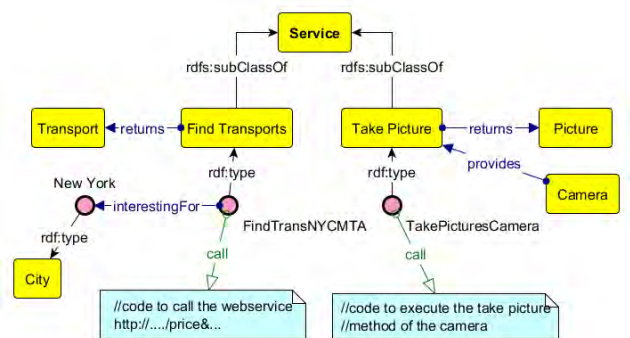

FIGURE 5. Two examples of the definition of external services: 1) to take pictures and 2) to obtain transports from a web service.

3) *Composed services*: Services defined via a composition of services of the two previous types (i.e., these services use them as atomic actions). This composition is defined by a workflow specified in BPMN [18], which is included in the service definition in XPDL format[4] using the *execute* property. Currently, we restrict our approach to support a subset of BPMN enough to

[4]http://www.xpdl.org, last accessed on 20th April, 2020.

model services executing atomic services in parallel, sequence, and combinations of the previous ones.[5]

Any service of these three kinds of services will be translated into *user requests* when a user selects them, as we will explain in Subsection VI. In particular, 1) a querying service will be translated into a query in our formal query language, 2) an external service into an *external call*, and 3) a composed service into a *workflow* composed of queries and/or calls.

IV. ACCESSING KNOWLEDGE

In this section, we explain the query language we designed to access the knowledge on the device to retrieve geospatial and semantic information. Then, we explain the agents involved in providing access to the knowledge in a device by processing queries posed in our query language.

A. GEOSPATIAL SEMANTIC QUERY LANGUAGE

The design of our approach has been guided by criteria of maximizing both expressivity and flexibility of supported LBS. Thus, in order not to be constrained to a predefined set of scenarios, with hardcoded queries and user needs, we designed a query language, named *Geospatial Semantic Query Language (GS-QL)*, used to access the knowledge explained in the previous section. We designed this query language taking as basis our previous experience in the field of location-based queries. In particular, we took as baseline the expressivity of the SQL-like query language proposed in [19] which allows to write location-based queries using different location granularities.

As our approach manages knowledge modeled using ontologies, we based GS-QL on SPARQL [20], a standard query language able to handle RDF data. To integrate both aspects (locations and DL semantics) in the same query language, we adopted (and adapted) the GeoSPARQL [21] and SPARQL-DL [22] extensions of SPARQL. The former extension is a standard for representation and querying of geospatial linked data from the Open Geospatial Consortium (OGC); while the latter one is a subset of SPARQL extended with predicates that are fully aligned with OWL 2, and which covers the typical functions associated with OWL.

The simplified grammar of GS-QL is shown in Figure 6[6] and explained in the rest of this section. We can distinguish two main parts in a GS-QL query:

- The *projections* clause, which declares the free variables that are used to match the result of the query. Note that as we do not have any attached schema (as, for example, in SQL), the meanings of these defined variables are not yet specified.
- The list of *where* clauses, which defines both the location constraints and conditions (*Conds*) that the required

---

[5]Note that our approach only supports the usage of simple atomic services within the workflows. We do not aim at supporting complex service composition (e.g., workflows within workflows).

[6]The complete grammar, as well as examples of the use of the query language, can be consulted at [23].

General query structure

Query	→	*SELECT* Projections Where
Projections	→	Var (',' Var)*
Where	→	*WHERE* '{' Conds '}'
			Where *OR WHERE* '{' Conds '}'
Conds	→	LICons? ObjectCons ProjectionsCons?

Location of Interest Constraints

LICons	→	*LI* '{' Patterns '}'

Object Constraints

ObjectCons	→	*OD* '{' ObjectDefs '}'
ObjectDefs	→	TypeDef (',' TypeDef)*
TypeDef	→	OptionalDef	ObjectDef
OptionalDef	→	*OPTIONAL* '(' ObjectDef ')'
ObjectDef	→	Patterns	*CASE* '(' Patterns ')'

Projection Constraints

ProjectionsCons	→	ProjCons (',' ProjCons)*
ProjCons	→	*PropertyValue*(VarOrIRI , VarOrIRI , VarOrIRI)

Patterns Definition

Patterns	→	Pattern (',' Pattern)*
Pattern	→	DLFunction	GeoFilter

/* DL-related functions */

DLFunction	→	SPARQLDL	DLExtension
SPARQLDL	→	…/* all the SPARQL-DL predicates */
DLExtension	→	*Domain*(VarOrIRI ',' VarOrIRI)
			Range(VarOrIRI ',' VarOrIRI)

/* GeoSPARQL functions */

GeoFilter	→	*FILTER* '(' GeoFunction ')'
GeoFunction	→	*geof:sfIntersects*(Geometry , Geometry)
			geof:sfWithin(Geometry , Geometry)
Geometry	→	VarOrIRI	*geof:buffer*(Geometry , Real , Units)

Basic grammar productions

VarOrIRI	→	Var	IRI
IRI	→	…
Var	→	…
…	…	…

FIGURE 6. Simplified grammar of the proposed query language.

objects have to met (*LICons* and *ObjectCons*, respectively), and the bindings of the previously declared variables to properties of such objects (*ProjectionsCons*).

1) CONSTRAINTS IN GS-QL

Location constraints defined within the *LICons* fragment of a *where* clause impose conditions on the locations of the returned objects, defining the *relevant area* of the query. We explicitly separate the definition of location constraints from object ones to clearly distinguish from spatial constraints that are to be interpreted continuously (e.g., retrieve objects that *are* within New York) from spatial patterns that refer to static properties of the objects (e.g., retrieve people that were *born* within New York). Moreover, note that location constraints are not mandatory, allowing for both location and non-location based queries.

Object constraints within the *ObjectCons* fragment of a *where* clause define semantically the objects that are to be returned. The patterns in this fragment can appear modified by an OPTIONAL clause which makes them not mandatory, and/or grouped with the help of a CASE operator. This latter operator allows for grouping object definitions by expressing the shared properties and separating the particular patterns into different CASES. Formally, let *DEF* be the set of patterns within an *ObjectCons* fragment which define the object, S_{DEF} the subset of patterns in *DEF* which are not within a CASE

clause, and S_{CASE} the set of sets of patterns within CASE clauses in *DEF*, then:

$$DEF(x) \Leftrightarrow S_{DEF}(x) \land \exists p \in S_{CASE} \mid p(x)$$

This is, all the mandatory patterns (i.e., those that are not modified by an OPTIONAL operator) are so except for those which are inside a CASE function, which are added to the body of the definition following a logical OR semantics. For example, with the CASE clause it is possible to define objects of interest which are *available vehicles*, and specifically *Taxis* of a particular operator, or *Buses* in general in the same query definition:

```
OD{
    Type(?thing, sherlock:Vehicle),
    PropertyValue(?thing, available, <true>),
    CASE(Type(?thing, sherlock:Taxi),
        PropertyValue (?thing, operator, <TaxiCab Co.>),
    CASE(Type(?thing, sherlock:Bus))
}
```

Finally, the projection constraints are used to select the information to return to the user. Therefore, the attributes defined in the projection constraints are those whose values are obtained from the objects that satisfy both location and object constrains in the query, and are returned as results.

2) PREDICATES IN THE PATTERNS
We can distinguish two main groups of predicates that can be used to form the patterns:

- Geographical predicates (*GeoFilter* in the grammar), which are taken from GeoSPARQL [21]. We have adopted three different functions which we needed to express inside constraints. In particular, we reuse `geof:intersects` and `geof:within` tests, which test intersection and inclusion relationships between spatial elements, and `geof:buffer` operation, which performs the dilation of a spatial element by a given distance. We focused on predicates that allowed us to define inside constraints as other types of location-dependent constraints (e.g., nearest) can be expressed by using inside constraints (for more details, see [7]).
- DL-related predicates (*DLFunctions* in the grammar), which are mainly taken from SPARQL-DL [22]. We have adopted all the SPARQL-DL predicates, keeping the same semantics as in their original definitions.[7] These predicates include functions to check, for example, if a given class is a direct subclass of another, or disjoint with it. Besides, we have included two functions to obtain the domain and range of a given property. These functions in the DL extension are not part of SPARQL-DL as these operations are not standard in DL, but are useful to our semantic agents. Their parameters are a property and a class, allowing at most one free variable, and their exact semantics depend on the position of such free variable (see Table 3). These functions enable to: 1) check whether a class is within the domain/range of the property, explicitly defined in

TABLE 3. Semantics of the introduced DL operators: domain and range.

Operator	Interpretation
Domain(C, P)	$true \Leftrightarrow C \sqsubseteq ExplicitDomain(C, P)$
Domain(?C, P)	$\{x \in concepts(O) \mid Domain(x, P)\}$
Domain(C, ?P)	$\{y \in roles(O) \mid Domain(C, y)\}$
Range(C, P)	$true \Leftrightarrow C \sqsubseteq ExplicitRange(C, P)$
Range(?C, P)	$\{x \in concepts(O) \mid Range(x, P)\}$
Range(C, ?P)	$\{y \in roles(O) \mid Range(C, y)\}$

the ontology, or a subclass of it (i.e., the class is an *implicit* domain/range[8]), 2) obtain all the classes which are explicit or implicit domain/range of a given property, and 3) obtain properties for which a given class is a explicit or implicit domain/range.

3) BENEFITS OF GS-QL
The adoption of this query language in our approach has the following benefits:

- It provides part of the expressivity of SQL and complements it taking into account semantic and geographic technologies.
- It decouples the system from a specific scenario, increasing its flexibility.
- It makes it easier to retrieve information from the local knowledge of the device for applications, agents, or people. Indeed, it is more flexible than developing APIs to access such an information.

B. AGENTS HANDLING KNOWLEDGE ACCESSING
In the following section, we explain the agents that provide access to the local knowledge on the device. First, we present the Knowledge Manager agent, which is in charge of the management of the knowledge and processing of GS-QL queries. Then, we detail the Endpoint agent, which provides an interface to the GS-QL processing capabilities of the device, and therefore the local knowledge, to external agents.

1) KNOWLEDGE MANAGER AGENT
The *Knowledge Manager* agent (KM from now on) encapsulates the managing of the knowledge stored in the device including highly-volatile data (e.g., the specific location of the user and the surrounding objects, sensor readings of the device, or even instances of current providers of each service) and more static information (e.g., the model of the device and its features, or definitions of services). To handle the volatile part of the knowledge, the KM agent adopts the strategy presented in [15], where static and volatile knowledge is stored in ontologies and databases, respectively, and is detected and marked at modeling time, allowing continuous DL query processing with enough expressiveness.

The KM agent also handles the access to the knowledge stored in the device, providing the rest of agents with services to update and retrieve both extensional and intensional

---

[7]The interested reader is referred to [22] for their detailed definitions.

[8]Notice that, we assume that subclasses of the explicitly defined domain/range inherit the quality of being also part of the domain/range of the property (following the inheritance model of object-oriented programming).

knowledge. This task includes to be in charge of processing GS-QL queries against the local ontology on the device posed by other local agents, and agents which belong to other users/devices, through the *Endpoint* agent.

2) ENDPOINT AGENT
Apart from interacting with users, a device in our approach can interact with an external agent through the *Endpoint* agent. The task of the Endpoint agent is to offer an interface to the processing capabilities of the device to other agents. This way, whenever a request is posed against the Endpoint agent, it decides how to handle it, forwarding the query to the KM agent if appropriate. All the knowledge access is limited to a single entry point which implements the required access control mechanisms to detect whether an agent can execute a query over the knowledge or not. Thus, if the Endpoint agent receives a query posed by an agent from another device (on behalf of another user), it can evaluate the privacy preferences of the user to control access and limit the sensitive information returned as a result.

Semantic Web technologies have been used in the literature to represent and enforce user privacy preferences, also called privacy policies. For instance, in [24] the authors used a semantic policy language to represent policies and reason over such language to enable access control to data in RDF stores. Although it is out of the scope of this work to deal with privacy issues, we took it into account to design the Endpoint agent which can use a similar approach to [24] to enforce access control over the local knowledge on the device.

V. UPDATING KNOWLEDGE
In this section, we focus on the knowledge update that devices in our approach continuously perform. This task is performed by two static agents that reside in each device, namely, the *Context Updater* and *Ontology Updater* agents. In the following, we detail the behavior of each of such agents.
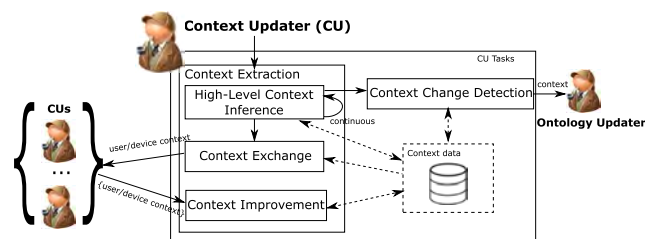

FIGURE 7. Context Updater agent (CU) tasks.

A. CONTEXT UPDATER AGENT
The *Context Updater* agent (CU from now on) is in charge of making appropriate decisions taking into account the current context of the user and her device. For that, the CU agent performs the following tasks (see Figure 7): context extraction and context change detection.

1) CONTEXT EXTRACTION
The CU agent infers new information about the context of the user from low-level sensory information obtained from both the device sensors and other CU agents, using the technique explained in [25]. Firstly, it infers the high-level context of the user continuously from low-level sensory data (e.g., the CU could infer that the user is running from the readings of her accelerometer). Then, the CU agent requests other CUs around to send their inferred high-level context information.[9] The CU agents use this exchanged information to enrich the context of their users (e.g., if a user device has no location information, the location of other users around can be used to enrich it) or even fix it (e.g., the readings of the GPS sensor could be erroneous and the information obtained from other devices could help to fix it). This process is performed continuously as some of the pieces of a user context are highly-volatile and change frequently, such as the location.

2) CONTEXT CHANGE DETECTION
Whenever the CU agent infers a new context for the user, it is in charge of detecting significant changes (e.g., when the user moves to a different city). These changes of context are used to reevaluate the information (e.g., services) that might be interesting for the user by the Ontology Updater agent.

B. ONTOLOGY UPDATER AGENT
The Ontology Updater agent (OU) is in charge of keeping the knowledge on the local ontology on the device updated. OU agents from different devices *learn* from their interactions as they exchange part of their local ontologies and data, integrating them in their local knowledge. In this scenario, appropriate knowledge management is crucial in order to keep the approach scalable (otherwise, a device would end up handling huge amounts of information, which might even not be related to the current context of the user). For that, each OU agent performs the following tasks (see Figure 8): knowledge exchange and maintenance.
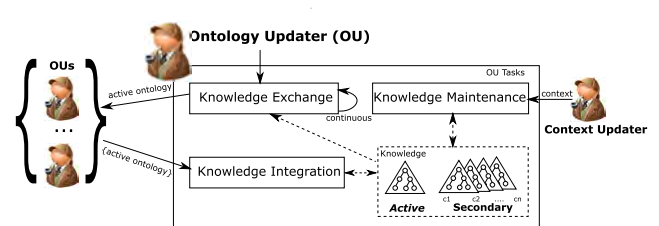

FIGURE 8. Ontology Updater agent (OU) tasks.

1) KNOWLEDGE EXCHANGE
Whenever two devices meet (i.e., they connect to the same network or they establish their own ad hoc network), their OU agents exchange knowledge so both devices learn from

---

[9]Notice that the privacy preferences of their users are checked by the Endpoint agent before exchanging information as explained before.

the interaction, increasing the information that a particular device has about its environment. To restrict the information exchanged, OU agents only exchange their *active ontologies* (i.e., the knowledge relevant to the user's current context) and associated data. This process is performed continuously, tracking the list of devices recently contacted to avoid exchanging the same information with the same devices all over.

When OU agents exchange knowledge the privacy preferences of their users are checked by the Endpoint agent before retrieving the knowledge to share in order to avoid disclosures. Finally, OU agents rely on a digital signature schema to enforce trust on the exchanged pieces of knowledge: each OU agent checks the validity of the signature/certificate of the user/company which defined each particular piece of knowledge (for instance, each service) before integrating it (note that the knowledge definer might be different from the knowledge exchanger).

2) KNOWLEDGE MAINTENANCE

Instead of integrating all the knowledge in an ever-growing ontology (which might lead to scalability problems), the OU keeps *active* just a module of the ontology which applies to the current user's context, while storing information that might be interesting in other contexts in *secondary* modules. Thus, the size of the ontology which will be used during the capturing of a user information need and its processing is minimized (local reasoning on current mobile devices has been shown feasible for small and medium ontologies [14]) whereas no knowledge is forgotten.[10]
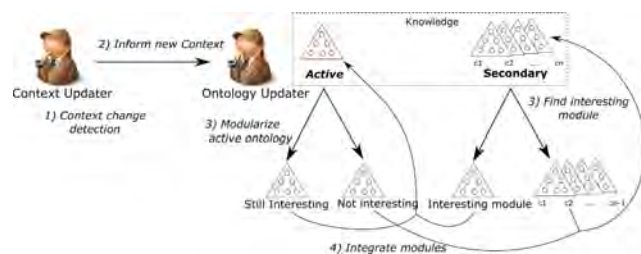
FIGURE 9. Steps involved in the management of knowledge.

As Figure 9 shows, whenever the CU agent informs the OU agent about a significant change on the user context, the OU agent starts the process of selecting the knowledge that might be of interest to the user. For that, it first checks the current active ontology to obtain which part is still of interest (e.g., definitions of services which do not depend on the specific location of the user and so might be always interesting for her) and which not. To extract such knowledge, the OU agent uses different ontology modularization techniques [27] exploiting the information about the current

context obtained from the CU agent (e.g., the new city where the user is). In parallel, the OU agent checks the secondary ontology, where different modules labeled using the context (in our case, the city) are stored, to find more interesting information. Afterwards, the interesting knowledge from the active and secondary ontologies are integrated to become the new active module, whereas the rest is also integrated and stored in the secondary ontology.

3) KNOWLEDGE INTEGRATION

As explained in Section III, devices have a pre-shared ontology which is extended by service definers in order to ontologically describe their services and the terms needed to do so. While this predefined vocabulary is useful to provide a base common knowledge model, the vocabulary extensions made by different vendors are likely not to be completely aligned, even when dealing with similar domains. For example, two different contributors might define a service to find transports and a service to find taxis without explicitly linking them, even though that the relation might be obvious. Therefore, when receiving knowledge from other devices and before integrating them, the OU agent has to align the exchanged schemas [28].

In our approach, we advocate to combine different techniques in order to extract synonym as well as subsumption relationships between terms (which is strongly helped by the pre-shared vocabulary). In particular, knowledge integration is performed by using the techniques explained in [29], [30] and in [31], which help in discovering synonymy and subsumption relationships between concepts from the local ontology of the user device and ontologies shared with it.

In the following section, we explain how this knowledge is used to express the user information needs, which involves helping the user to select the appropriate service, enabling her to input her constraints, and finally, translating this information into a formal request.

VI. MANAGING USER INTERACTION

Devices in our approach interact with their users to provide them with the available services depending on their context, helping them to express their needs. Recalling the architecture of a device (Section II-A), there are two static agents within each device which handle the interaction with the users:

- Alfred, which stores information from the user such as her preferences or the information provided when interacting with the system that could be used to help in selecting services that might be interesting for her or to fill in parameters attending to previous selections by the user.
- ADUS, which generates graphical user interfaces (GUIs) for applications, in a context with heterogeneous devices, considering their features. ADUS generates dynamically the interfaces needed when an agent wants to interact with the user as explained in [32], [33]. For this, such an agent must make a petition to the ADUS

---

[10]Serialization and incremental reasoning are two desirable characteristics of the DL reasoner used in each device; however, as noted in [26] there is no current reasoner which supports both at the same time. Thus, incremental reasoning should be more important.

agent along with the associated interface specification from the ontology (see *interface* property in Figure 3).

ADUS is able to create interfaces to obtain input from the user for usual types such as booleans, numbers, or strings, as well as more complex types such as instances of concepts defined in the ontology, locations/areas, and descriptions of images. ADUS can be extended to support other types of information and widgets, provided that they are defined in the ontology, and the code or module needed to capture such information is deployed and incorporated to the ADUS agent.

On the other hand, to help capturing the user information needs and translating them into a formal request in order to avoid ambiguities, our approach relies on an extra agent which is created on demand, the *User Request Manager* agent. The URM agent performs three main tasks[11] (see Figure 10): 1) Service Selection, 2) Parameter Provision, and 3) Service Handling.
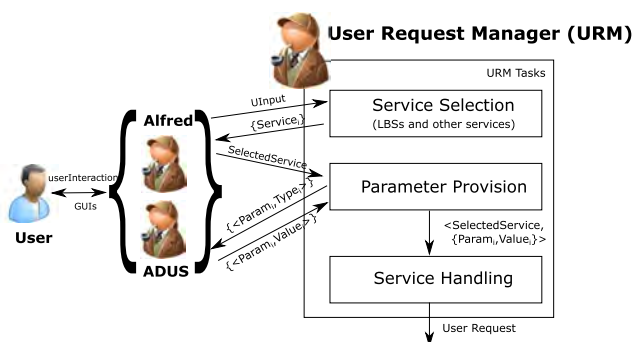

FIGURE 10. User Request Manager agent (URM) tasks.

A. SERVICE SELECTION

The URM is in charge of obtaining the services (based on a location or not) that are relevant for a user in a particular situation. To start a service selection, the user can choose to select a service from a displayed list of available services, or by selecting an entity on a displayed map (e.g., objects displayed as the result of a previous request, or a particular GPS coordinate[12]). Alfred captures such an interaction, and creates a new URM agent with the information captured from the user. By default, the URM obtains the available services by querying the *active ontology* with the user's context information (i.e., location and current activity).

When the user has selected a particular entity, the URM retrieves the services that are related to such entity instead. For that, firstly, the URM obtains a list of entities which geographically contain the selected one. For example, if the user selected the MoMa museum in a map, the URM would obtain the city (location) in which the museum is, and "Museum", which is the class to which the entity belongs to. Then, for each entity obtained before (instances and

---

[11]For the detailed algorithms followed by the URM agent in the request generation process, we refer the reader to [23].

[12]It might be the current user's location.

concepts), the URM obtains all the services which are related to it.[13]

If no service is retrieved or the user does not find an appropriate one, the URM queries the secondary modules in order to maximize the chances to find the required service. The result of this task is a list of services along with their *families* to be shown to the user for her to select one.

B. PARAMETER PROVISION

The result of the previous step is the selection of a family of services (i.e., concepts that are subclass of *Service*) or a particular service (i.e., an instance of such concepts). For example, the user could select the family of "Find Transportation" services or the specific "Find MTA Transportations" service (which is the service provided by New York's Metropolitan Transportation Authority). In fact, a user that wants to find transports regardless of the provider of this information would use the former one, whereas a user that wants the information offered by a specific provider would select the latter.

Firstly, the URM has to obtain the parameters of such services, if any. These parameters, which have been defined when the service was modeled, are the formal parameters that the service requires to be invoked or that can be used to restrict the information returned by the service. Thus, depending on the selected entity, the URM has to obtain the set of parameters to be fulfilled as follows:

- *Service family*: The user has selected a family of services that share a set of formal parameters needed and a set of returned objects. We will denote such a family of services as *SServ*. The URM consults the ontology to obtain all the constraints of the type $SServ \sqsubseteq parameter$:$?x$, which define the minimum set of parameters that such a service has to receive. The result is a set $\{fp_1, \ldots, fp_n\}$ of parameters that are applicable to that service.
- *Particular service*: The user has selected a particular instance of a service. In this case, firstly, the URM consults the service family of the selected instance. Then, the URM obtains all the parameters that correspond to the service due to the concept hierarchy (as in the previous item). Finally, as this service might have extra or constant value parameters, the URM extends (and overrides) the previous result set $\{fp_1, \ldots, fp_n\}$ with the parameters applicable to such a particular instance obtained by consulting the Service ontology via *parameter* property. This would be retrieved using the clause $PropertyValue(<serviceSelected>, parameter, ?ip)$.

In both cases, the result is a set of parameters which have to be assigned a value to in order to be able to invoke the service or to filter its results out. The parameters that have a predefined value are not required from the user and are automatically filled for the final request. For the rest of them, to obtain the actual values of the parameters, the URM

---

[13]In particular, the URM explores the *provides*, *returns*, and *interestingFor* properties to check whether a service is relevant for such entity.

relies on ADUS and Alfred. Each parameter comes along with information about their expected value to be entered (e.g., a location, a boolean, or even an instance of a concept defined in the ontology).

Notice that there are three types of "parameters" shown to the user: 1) parameters defined in the ontology through the *hasParameter* property; 2) location for LBS; and 3) provider in the case of services provided by several providers. A service might not have any parameter at all, although typically services will have parameters of the first type used to filter out the information returned. LBS will need, by definition, a location which might have to be requested to the user. Finally, external services (e.g., provided by other devices as explained in Section III-B) might need the user to select the specific provider. For example, if the user selected the external service provided by taxis to "pick her up", she will have to select the specific taxi, or any or even all, through the Graphical User Interface. In this last case, the list of specific providers (e.g., taxis) is obtained executing a GS-QL against the local knowledge.

C. SERVICE HANDLING

With the service selected by the user (recall that there are three types of services possible) and the values that the user introduced for the parameters associated with such service, the goal of this task is to create the formal request. This request could be later processed against the local knowledge by the KM agent and even against external knowledge on other devices using approaches to process queries against other devices [7]–[9].

In the case of a querying service, the goal of this step is to translate the information provided by the user into a formal query using GS-QL (the detailed algorithm can be found in [9]). For that, the URM agent exploits the information in the model to translate the definition of:

1) The target object(s) (i.e., the entities that the service returns) by including: 1) the specific target(s) of such service modeled in the local ontology (through the *returns* property), and the ontological definition of the target of such service (different devices could have different information in their local ontologies and thus, the ontological definition might be needed to understand the request); and 2) constraints to fulfill the parameters and values that the user selected.

2) The location of interest, in the case of a LBS, using the special location parameters included by the user. A `WHERE` clause (see Section IV-A) is generated for each different location selected by the user or modeled in the service definition and the previous definition of the target objects are included in them.

In the case of an invoked service, the information provided by the user is included in the invocation, as specified by the call included in the ontology. Also, for services that contain an execution plan the querying services in it are translated to formal GS-QL queries and invoked services are translated into invocations. Note how the user does not need to be aware

neither of the details of the query language, nor the schema and the definitions of available services in order to translate her information needs into a formal request.

VII. DEALING WITH USE CASES

In this section, we describe three different LBS, as representation of many others, that can be handled by our approach. First, we showcase how those LBS can be modeled using the ontology presented in Section III. Then, we explain the most important steps involved in the interaction with the user (as explained in Section VI).

Looking for Transportation: In our first scenario, a person arrives in a foreign city and needs to find transportation. Let's imagine that John is visiting Zaragoza (Spain); he has just arrived at the railway station of Zaragoza and wants to find transportation that could carry him to his hotel ("Hotel Palafox"). It is the first time that John visits Spain and he does not know anything about transportation there, but he would prefer a private transport that could carry him directly to the destination.

Helping Firefighting: In our second scenario, the coordinator of a firefighting team in charge of the suppression of a wildfire needs information about his team and the environment. Let's imagine that John is the coordinator of a wildfire suppression team in Yellowstone National Park, and is interested in obtaining information about fire outbreaks and the firefighter team under his command (which consists of five firefighters, two firefighting trucks, and a helicopter). In particular, John needs information about the location of each of the members of the team as well as their sensors readings. Also, he needs the approximate location of the fire outbreaks to have information about the affected area.

Handling Sport Events Broadcasting: In the third scenario, a Technical Director (TD) needs assistance in the live broadcasting of a sport event. In this case, let's imagine that John is the TD in charge of the live broadcasting of *La Bandera de la Concha 2019*, a famous rowing race celebrated annually in San Sebastian (Spain). Among the many tasks that John has to perform, the most important one is to select the camera to broadcast at each moment. John is an experienced TD and has some specific shots in his mind to broadcast. So, he would like to define them and then obtain the list of cameras (from the broadcasting company and even from the audience) that could provide them.

A. KNOWLEDGE MODELING

To handle the first scenario, our approach needs knowledge about transportation services in the area as well as the surroundings (e.g., about the previously mentioned hotel). Figure 11 shows an excerpt of the ontology which models a definition of the different services to find transports. The general *Find Transportation* service returns any type of transportation and could be part of the local knowledge of the device prior to arriving in Zaragoza. This service will be processed as a query as explained in Section VI-C. The *Find Bus Tuzsa* service is a particular service that operates in the
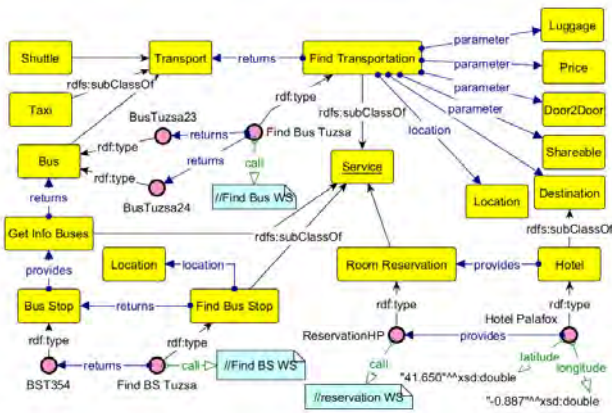
FIGURE 11. Excerpt of the ontology for the "Looking for Transportation" scenario.

city and returns the location of buses belonging to the local bus corporation of Zaragoza through a web service (which was unknown for John). Notice also that we define other services to obtain the location of bus stops and information about buses from them.

Let's imagine that the knowledge about Zaragoza and the different transportation means which operate in it has been shared by a device in the tourist information center at the railway station. This knowledge has been integrated into the local ontology on John's device. Therefore, the moment John reaches the railway station, his device learns this knowledge by autonomously communicating with the information center.
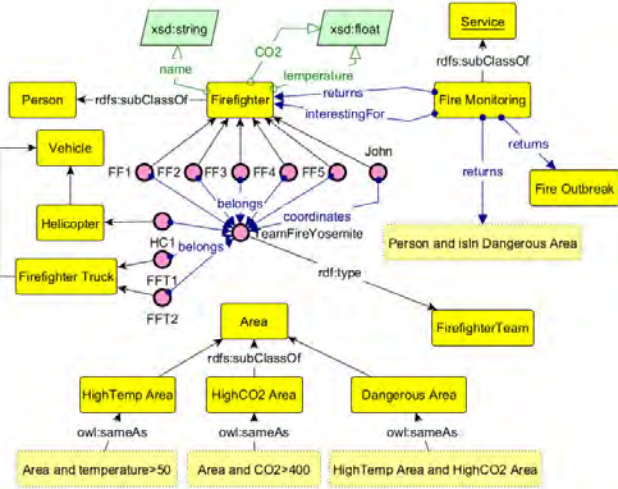

FIGURE 12. Excerpt of the ontology for the "Helping Firefighting" scenario.

To manage the second scenario, we define the knowledge in Figure 12. First, we define a service to monitor wild fires which returns the location of fire outbreaks as well as the location of any personnel or vehicle involved in the fire suppression (the *Fire Monitoring* service). Additionally, this service returns also the location of any person that might be

in danger. To model this, we first defined the type of returned information semantically as those instances of *People* that fulfill *isIn Dangerous Area*, and defined such area as *High Temperature Area* (*hasTemperature > 50*) and *High Level of CO2 Area* (*hasCO2 > 400*).[14] This shows that more complex definitions of concepts can be modeled thanks to the generality and flexibility of our approach. Finally, we defined information about John and his team, including the members and equipment.

As in the previous scenario, this knowledge could have been defined by a knowledge engineer working for the fire-fighting unit and shared with John's device before departing from the station.

The third scenario, needs a service to obtain cameras that could provide a specific view of different objects. We defined the general service *Find Camera* as an implementation of such services (see Figure 13 for an excerpt of the ontology). This service returns entities of type *Camera*, which could be even attached to a mobile devices such as an smartphone, and has parameters such as the distance from the camera to the objects inside the field-of-view and the visibility of such objects. Notice that, for the latter we have modeled that such parameter can be obtained through a specific GUI (a 3D Query-by-example interface). Also, we have defined two services provided by these cameras to ask them to take pictures and videos and share them with the requester, the *Take Picture* and *Take Video* services, respectively.

FIGURE 13. Service to find cameras that could obtain a certain shot.

We have defined also a service to manage the broadcasting of a sport event (see Figure 14 for an excerpt of the ontology)

[14]Temperatures are measured in Celsius degrees and CO_2 in ppm (parts per million).
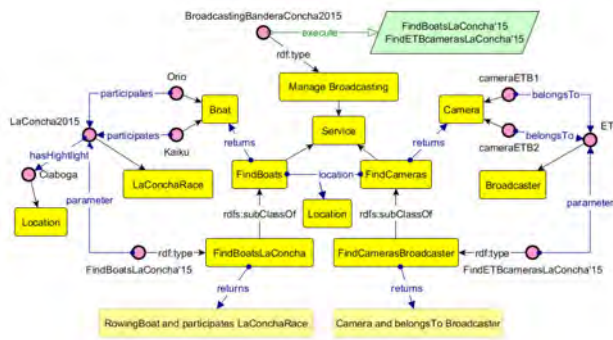
FIGURE 14. Excerpt of the ontology for the "Handling Sport Events Broadcasting" scenario.

and a specific instance of this service for the rowing race in the example (*BroadcastingBanderaConcha2019*). Notice that, this service obtains information about the rowing boats participating in the race and the cameras of the broadcaster (using the *Find Camera* service modeled in Figure 13). Also, we have defined knowledge related to the rowing race which is used in combination with the previous definition of the service. This includes the participants of the race and cameras managed by the broadcasting company, as well as possible interesting locations for the broadcast (such as the *ciaboga* area which is the turning point for the boats). All this knowledge would be defined by the broadcasting company and its information system would share it with the TD's device.

B. USER INTERACTION MANAGEMENT

We show the most important steps that explain how our approach deals with the previous scenarios focusing on the request generation part (i.e., translating the user information needs into a GS-QL query). For some of them we will use screenshots of an Android prototype we developed based on the ideas presented in this paper.

1) SERVICE SELECTION

First, the user needs to interact with the system to show her interest in a specific service. This exploratory discovery of services can be done in different ways. For example, in the first scenario John types in *Hotel Palafox* in a search bar (see Figure 15(a)). A User Request Manager (URM) agent is created which finds an instance of the hotel class whose name corresponds to that string, and therefore understands that the user is interested in a hotel. The URM deduces, after querying the local ontology on the user device, that there are two LBS related to hotels in its local ontology: *Find Transportation* and *Room Reservation*. Remember that the URM looks for services that are somehow related to the concept *Hotel* whatever the name of the property that references such a concept is (we do not assume any predefined schema in the definition of services). In this case, the properties are *parameter* (because *Hotel* is a subclass of *Destination*, which is a parameter of the *Find Transportation* service) and *provides* (because *Hotel*

(a)   (b)

FIGURE 15. Screenshots of the prototype executing the first scenario.

provides the *Room Reservation* service), respectively. The user then selects the *Find Transportation* service. In the case of the rest of the scenarios, let's imagine that the user taps on the service tab where a URM agent displays a list of service which can be interesting for him. For instance, the *Fire Monitoring* service appears in this list for the coordinator of the firefighter team as the user selected the *firefighter* profile and it matches the profile linked to the service through the *interestingFor* property. Similarly, the TD of the third scenario selects the *Manage Broadcasting* service to obtain the real-time location of the rowing boats and the cameras under his control. The TD is interested in broadcasting a shot of the local team rowing boat (*Donostiarra*). For that he first wants to obtain a list of cameras that could provide such a shot to select one of them and broadcast its feed using the *Find Cameras* service.

2) PARAMETER PROVISION

The URM created for each scenario obtains from the local ontology the parameters of the selected service. In the case of the *Find Transportation* service, the URM obtains from the local ontology the parameters of such a service, (*Price*, *Shareable*, *Door2Door*, and *Luggage*). The user shows his interest in a transport *Door2Door* (indicating that this is mandatory) that admits *Luggage*, if possible. In the case of the *Fire Monitoring* service, the parameter is the location to monitor. For the *Find Cameras* service, the URM obtains that one of the parameters associated with the service in the ontology is the definition of a sample shot.

Then ADUS generates a GUI to fill in these parameters. For example, Figure 15(b) shows the GUI generated to fill in the parameters for the *Find Transportation* service. In the case of services with parameters related to a location (e.g., the *Fire Monitoring* service) the URM offers the user a map through ADUS to select such location of interest. ADUS displays a

3DQBE (3D Query-By-Example) interface [34] (linked to the parameter through the *interface* property) to define the specific shot of the rowing boat to retrieve. On that interface, the user defines the kind of shot to obtain by rotating the camera view and even including other objects in the scene. The interface translates this sample shot into: "An image showing 50% of the top view and 70% of the right side view of Kaiku, and 40% of the front view and 15% of the right side view of any other rowing boat".

3) SERVICE HANDLING

With the information defined by the user, the URM can handle the selected service. This handling might result in the invocation of an external service or the processing of a query. As an example of the former, we can cite the *Find Bus Tuzsa* service in the first scenario, which is an instance of the *Find Transportation* service available for that specific geographic area (Zaragoza) and time: the URM can obtain information about buses in the city from a web service. In the case of services that require the processing of a query, the URM has to translate the user needs into a formal GS-QL query. For instance, the URM handling the *Find Transportation* service infers that objects belonging to the *Taxi*, *Bus*, and *Shuttle* classes fulfill the user preferences and provide transport services. With this information the URM generates the query in Figure 16. The query includes the inferred interesting transports (*Taxi*, *Shuttle*, and *Bus*) as well as the general definition of interesting transport that the user selected (a *Transport* that is *door2door* and admits *luggage*). The URM includes transports that do not fulfill completely the requirements of the user (i.e., *Bus*) to maximize the chances of obtaining results.

```
PREFIX sher: <http://sid.cps.unizar.es/ontology/sherlock/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX geof: <http://www.opengis.net/def/geosparql/function/>

SELECT ?name, ?lat, ?lon
WHERE{
 LI{
  FILTER(geof:sfWithin(?thing,
      geof:buffer(<userloc>, 1, km))
 },
 OD{
  CASE(Type(?thing, sher:Transport),
      PropertyValue(?thing, sher:parameter, ?door2door),
      PropertyValue(?door2door, sher:name, ``door2door''),
      PropertyValue(?door2door, sher:value, true)),
      PropertyValue(?thing, sher:parameter, ?luggage),
      PropertyValue(?luggage, sher:name, ``luggage''),
      PropertyValue(?luggage, sher:value, true)),
  CASE(Type(?thing, sher:Taxi))),
  CASE(Type(?thing, sher:Shuttle)),
  CASE(Type(?thing, sher:Bus)),
 },
 PropertyValue(?thing, sher:name, ?name),
 PropertyValue(?thing, sher:latitude, ?lat),
 PropertyValue(?thing, sher:longitude, ?lon),
}
```

FIGURE 16. Formal query for the first scenario (*Find Transportation*).

For the second use case, the actual query generated is similar to the previous (see Figure 17) and target types are *Firefighters*, *Fire Outbreaks*, and the type defined as *People and isIn Dangerous Area* (i.e., instances of the *People* class that

```
SELECT ?name, ?lat, ?lon
WHERE {
 LI{
  FILTER(geof:sfWithin(?thing,
    geof:buffer(<selectedLoc>, 10, km))
 },
 OD{
  CASE(Type(?thing, sher:Person),
    PropertyValue(?thing, sher:isIn, ?area),
    Type(?area, sher:Area),
    PropertyValue(?area, sher:temperature, ">50"),
    PropertyValue(?thing, sher:CO2, ">400")),
  CASE(Type(?thing, sher:Firefighters))),
  CASE(Type(?thing, sher:Fire Outbreaks)),
 },
 PropertyValue(?thing, sher:name, ?name),
 PropertyValue(?thing, sher:latitude, ?lat),
 PropertyValue(?thing, sher:longitude, ?lon),
}
```

FIGURE 17. Formal query for the second scenario (*Fire Monitoring*).

```
SELECT ?videoStream, ?timestamp, ?lat, ?lon, ?dir, ?fov
WHERE {
 LI{
  FILTER(geof:sfWithin(?thing, sher:LaConcha)),
 },
 OD{
  Type(?videoStream, sher:Video),
  Type(?camera, sher:Camera),
  PropertyValue(?camera, sher:captures, ?videoStream),
  CASE(
   PropertyValue(?videoStream, sher:views, ?vTopKaiku),
   PropertyValue(?vTopKaiku, sher:target, Kaiku),
   PropertyValue(?vTopKaiku, sher:angle, Top),
   PropertyValue(?vTopKaiku, sher:amount, 50),
    ...
  ),
  CASE(
   PropertyValue(?videoStream, sher:target, Kaiku)
  ),
 },
 PropertyValue(?camera, geo:lat, ?lat),
 PropertyValue(?camera, geo:lon, ?lon),
 PropertyValue(?camera, sher:direction, ?dir),
 PropertyValue(?camera, sher:fov, ?fov),
 PropertyValue(?videoStream, sher:timestamp, ?timestamp)
}
```

FIGURE 18. Formal query for the third scenario (*Handling Event Broadcasting*).

are located in an area classified as *Dangerous Area*). Notice that the definition of what a dangerous area is is included in the query too. The query for the third use case contains two parts (see Figure 18), the first one (first CASE clause) defines the video stream to obtain as showing the specific shot that the TD defined. As this might be too restrictive, the second part (the second CASE clause) defines also the video stream as showing the "Kaiku" rowing boat. Also, notice that the query selects, in addition to the video stream URL, some parameters about the camera such as the location, direction, and field-of-view. These parameters can be used later to identify whether the camera can obtain the requested shot or not.

4) UPDATING KNOWLEDGE

In all the scenarios, while the URM agent interacts with the user, the Ontology Updater (OU) agent is actively updating the local knowledge on the device through interactions with other devices. This way, the OU on the device of the tourist in the first scenario discovers that there exist moving objects

classified as *Bikecab* (an unknown subclass of *Taxi* for the ontology of the user). This new knowledge enriches the user device knowledge and enables the URM to infer that bikecabs also fulfill the user preferences and the query being processed is edited to reflect it in the next reevaluation of the query.

5) CONCLUSIONS

We have shown how our approach can be used to model and handle different interesting LBS. The information provided by the user (a click on a map, selecting service, filling a user-friendly form) is enough for our approach to generate a formal query to retrieve interesting information for the user. For instance, John did not know anything about specific transportation means in the city (e.g., buses in Zaragoza) and our approach managed all this knowledge for him and even keep it updated when meeting other devices (e.g., learning about *bikecabs*). We have also shown that our approach can handle more complex specification of user preferences (such as the 3DQBE interface to visually define the kind of camera shot the TD wants to obtain) and also more complex definition of relevant data to capture (such as the specification of people that are in a dangerous area which implies inferring whether a specific geographic area is dangerous or not). Finally, we have shown how our approach can integrate existing third-party data sources specified in ontology descriptions of the services providers (e.g., existing web services to find the location of buses in the city).

By leveraging our approach, a system that monitors transports, cameras, or fire outbreaks, could process the detailed formal queries generated to offer the user the exact information she needs. This way, that system would need to focus just on obtaining information about those elements and do not need to deal with the complexities related to interacting with different users that have different information needs and knowledge about a specific scenario.

VIII. RELATED WORK

The main goal of our approach is to provide users with multiple services, based on her location or not, helping her to express her information needs and keeping the knowledge about available services updated. Up to the author's knowledge, no other work has proposed a general and flexible system based on semantics to: 1) handle diverse LBS, and 2) capture the user information needs into formal requests related to such services. Therefore, we will provide an overview of contributions to some specific research areas related to our proposal. Firstly, we present works focused on providing LBS and location-dependent queries, which are the building blocks of LBS. Secondly, we present Service-Oriented Architectures, which are generally focused on providing services to build applications (i.e., B2B services). This kind of services would be considered external ones in our approach. Moreover, within SOA approaches, we highlight Semantic Web Services, which are closely related to our goal of modeling services using ontologies.

A. LOCATION-BASED SERVICES

Location-Based Services (LBS) have been defined before as ''services that integrate a mobile device's location or position with other information so as to provide added value to a user'' [1]. Although LBS appeared in the 90s, the research community is still actively working on research challenges associated with them [35]. There are plenty of applications in the literature to provide users with specific location-based services [36]. For example, taxi searching [2], helping firefighting [3], detecting and recommending nearby friends [4], or multimedia retrieval in sport events [5], among many others. Also, there are some proposals of architectures to provide LBS. For example, in [37] an architecture to support LBS applications is presented. The Base Stations (BS) serving cells in a cellular network contain a geolocation server and database that gathers information about mobile devices in the area and their requests. This way, when a mobile device connects with a BS and executes a service registered in the local registry, the server can execute the service using the information in the database and return the result to the mobile device. In [38] a LBS system is presented with a similar decentralized architecture. In this system, a local registry is placed in each cell of the cellular network system which enables providers to register their services. The system running on each Base Transceiver Station (BTS) which serves a specific cell broadcasts the information from its local registry to devices connected to the BTS. Then, mobile devices can execute a call to specific services. The main difference between these approaches and ours is that their decentralized architecture is based on a set of BS which maintain information about objects and services in their cell whereas we do not assume the existence of a fixed infrastructure. Another difference is that our approach also helps the user to express her information needs and integrates new information about services using semantic techniques for interoperability. The Snap4City platform [39] provides a centralized architecture in which geographical and statistical data from different sources is collected to enable developers to create LBS. The platform uses the Km4City multiontology [40] to represent geographical information using a common metamodel. As in the previous case, the main difference with our system is the decentralized architecture presented in this paper.

B. SERVICE-ORIENTED ARCHITECTURES

Context-aware frameworks simplify the development of context-aware applications/services (see [41] for a survey on context-aware systems). For example, the highly referenced Service-Oriented Context-Aware Middleware (SOCAM) architecture [42] supports the building of context-aware mobile services. SOCAM is based on a centralized server which gathers context data from context providers and offers it to clients. Context-aware services can be built by defining rules which specify under which circumstances an action has be performed. SOCAM uses a set of OWL ontologies for modeling the context information that context-aware services

can use. As explained in [43], most of these architectures are based on a centralized server which contains the services available and offer them to the user. However, in our approach each device independently handles the knowledge related to services and keep it updated thought the interaction with other devices. Also, our proposal takes into account the process to capture the user information needs.

The service oriented nature of our proposal along with the use of ontologies in order to describe the scenarios and the different elements of the handled services could bring to mind Semantic Web Services [28], [44]. According to the W3C definition: *"a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards"*.[15] This is, Web Services are oriented to achieve system interoperability. On top of this technology, Semantic Web Services appeared [28], which are Web Services whose functionality is described using ontological terms and semantic annotations, aiming at enabling automatic and dynamic interaction between software systems. However, the Web Service's stack is designed to make a service available to other programs using Web technologies, while the notion of service handled in our distributed architecture is oriented completely to provide services to the final user. Moreover, our approach is aimed at providing location-based services, using all the available resources (i.e., locally integrated knowledge, the distributed architecture of devices, or even third-party services - which might include Web Services). Thus, instead of adopting an existing formalism [45] to extend it and complexly define our services, we have advocated for keeping a simpler model, relying on the simplicity of our schema to both: 1) make it possible to perform a light-weight yet flexible service discovery instead of using more complex approaches [46] which might overload mobile devices; and 2) make the definition of services easier for service providers. The proposed model is not aimed at matchmaking or automatic service composition (a provider has to provide the workflow if a composed service is required), but at providing a context-aware search of relevant services within a distributed architecture which lacks of a previously global and shared schema (which is achieve via knowledge sharing among devices).

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we have detailed an approach for the management of knowledge related to Location-Based Services (LBS) as well as for helping users to express their information needs. The management of knowledge is based on the modeling and maintenance of contextual information about users and their devices, as well as information about services

and functionalities. To be useful, this knowledge is kept updated by leveraging the collaboration between mobile devices equipped with out approach. Also, we presented a query language for the system based on SPARQL, a standard semantic query language, which can be used to obtain information from a the local knowledge in a device. Besides, we explained how our approach leverages the knowledge about services and the user to understand her information needs. An agent offers users services that might be interesting for them and capture their preferences to generate a formal user request expressed in our query language. Finally, we illustrated the flexibility of our approach applying it to three heterogeneous use cases. The contributions of the approach presented in this paper are the following:

1) It enables devices to exchange knowledge related to services which might be interesting for their users. Through their interactions, devices exchange ontologies which each node integrates into their local knowledge so they can learn from the interaction. At the same time, it maintains the local knowledge on the device updated while taking into account its size to enable efficient reasoning.

2) It offers to the user the available services which might be interesting for her depending on her context and interactions with the system. This way, it relieves the user from managing specific knowledge about services. Also, it helps users to select the most appropriate service through a guided process which interacts with her to obtain her specific information needs.

3) It translates the user requirements into formal requests in GS-QL, a SPARQL-like query language we designed incorporating extensions of SPARQL to handle semantic and location constraints. This language decouples our approach from a specific scenario increasing its flexibility. Also, it can be used by external services/users/applications to obtain information from the local knowledge of a device.

As future work, we plan to focus on the processing of the user requests expressed in GS-QL generated as a result of the process explained in this paper. As we introduced in [8], [9], the use of mobile agents can be beneficial for this processing as they can be deployed to locations where the information might be available.

---

[15]https://www.w3.org/TR/ws-arch, last accessed on 20th March, 2020.

REFERENCES

[1] J. Schiller and A. Voisard, *Location-Based Services*. San Mateo, CA, USA: Morgan Kaufmann, 2004.

[2] C.-R. Dow, D.-B. Nguyen, S.-C. Wang, S.-F. Hwang, and M. F. Tsai, "A geo-aware taxi carrying management system by using location based services and zone queuing techniques on Internet of Things," *Mobile Inf. Syst.*, vol. 2016, pp. 1–10, 2016, doi: 10.1155/2016/9817374.

[3] S. G. Hong, K.-H. Son, H. Lee, M. Bae, and K. B. Lee, "Augmented IoT service architecture assisting safe firefighting operation," in *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2018, pp. 1–6.

[4] H. Bagci and P. Karagoz, "Context-aware friend recommendation for location based social networks using random walk," in *Proc. 25th Int. Conf. Companion World Wide Web (WWW) Companion*, 2016, pp. 531–536.

[5] S. Ilarri, E. Mena, A. Illarramendi, R. Yus, M. Laka, and G. Marcos, "A friendly location-aware system to facilitate the work of technical directors when broadcasting sport events," *Mobile Inf. Syst.*, vol. 8, no. 1, pp. 17–43, 2012.

[6] B. Gedik and L. Liu, "MobiEyes: A distributed location monitoring service using moving location queries," *IEEE Trans. Mobile Comput.*, vol. 5, no. 10, pp. 1384–1402, Oct. 2006.

[7] S. Ilarri, E. Mena, and A. Illarramendi, "Location-dependent queries in mobile contexts: Distributed processing using mobile agents," *IEEE Trans. Mobile Comput.*, vol. 5, no. 8, pp. 1029–1043, Aug. 2006.

[8] R. Yus, E. Mena, S. Ilarri, and A. Illarramendi, "SHERLOCK: Semantic management of location-based services in wireless environments," *Pervasive Mobile Comput.*, vol. 15, pp. 87–99, Dec. 2014.

[9] R. Yus, "Semantic management of location-based services in wireless environments," Ph.D. dissertation, Dept. Comput. Sci. Syst. Eng., Univ. Zaragoza, Zaragoza, Spain, Mar. 2016.

[10] R. Yus and E. Mena, "Emergency management using SHERLOCK," in *Proc. 13th Annu. Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, 2015, p. 495.

[11] R. Yus and E. Mena, "Continuous processing of real-time multimedia requests using semantic techniques," in *Proc. 13th Int. Conf. Adv. Mobile Comput. Multimedia (MoMM)*, 2015, pp. 216–220.

[12] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *Int. J. Hum.-Comput. Stud.*, vol. 43, nos. 5–6, pp. 907–928, Nov. 1995.

[13] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[14] C. Bobed, R. Yus, F. Bobillo, and E. Mena, "Semantic reasoning on mobile devices: Do androids dream of efficient reasoners?" *J. Web Semantics*, vol. 35, pp. 167–183, Dec. 2015.

[15] C. Bobed, F. Bobillo, S. Ilarri, and E. Mena, "Answering continuous description logic queries: Managing static and volatile knowledge in ontologies," *Int. J. Semantic Web Inf. Syst.*, vol. 10, no. 3, pp. 1–44, Jul. 2014.

[16] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proc. 1st Int. Symp. Handheld Ubiquitous Comput. (HUC)*, 1999, pp. 304–307.

[17] J. Bernad, C. Bobed, and E. Mena, "Estimating local coverage areas for location dependent queries," in *Proc. 33rd Annu. ACM Symp. Appl. Comput. (SAC)*, 2018, pp. 940–947.

[18] "Introduction to BPMN," IBM Cooperation, White Paper, 2004, vol. 2. [Online]. Available: https://www.omg.org/bpmn/Documents/Introduction_to_BPMN.pdf

[19] S. Ilarri, C. Bobed, and E. Mena, "An approach to process continuous location-dependent queries on moving objects with support for location granules," *J. Syst. Softw.*, vol. 84, no. 8, pp. 1327–1350, Aug. 2011.

[20] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," in *W3C Recommendation*, vol. 15. 2008. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/

[21] R. Battle and D. Kolas, "GeoSPARQL: Enabling a geospatial semantic Web," *Semantic Web J.*, vol. 3, no. 4, pp. 355–370, 2011.

[22] E. Sirin and B. Parsia, "SPARQL-DL: SPARQL query for OWL-DL," in *Proc. 3rd Int. Workshop OWL Experiences Directions (OWLED)*, vol. 258, 2007, pp. 1–10.

[23] *Additional Material: A Knowledge-Based Approach to Enhance Provision of Location-Based Services in Wireless Environments*. Accessed: Apr. 30, 2020. [Online]. Available: http://sid.cps.unizar.es/LBSManagement

[24] A. Padia, T. Finin, and A. Joshi, "Attribute-based fine grained access control for triple stores," in *Proc. 3rd Int. Workshop Soc., Privacy Semantic Web Policy Technol. (PrivOn)*, 2015, pp. 1–15.

[25] R. Yus, P. Pappachan, P. K. Das, T. Finin, A. Joshi, and E. Mena, "Semantics for privacy and shared context," in *Proc. 2nd Int. Workshop Soc., Privacy Semantic Web Policy Technol. (PrivOn)*, 2014.

[26] C. Bobed, F. Bobillo, E. Mena, and J. Z. Pan, "On serializable incremental semantic reasoners," in *Proc. Knowl. Capture Conf. (K-CAP)*, 2017, pp. 187–190.

[27] H. Stuckenschmidt, C. Parent, and S. Spaccapietra, *Modular Ontologies Concepts, Theories and Techniques for Knowledge Modularization* (Lecture Notes in Computer Science), vol. 5445. Springer, 2009.

[28] J. Euzenat and P. Shvaiko, *Ontology Matching*, vol. 18. Springer, 2007.

[29] R. Yus, E. Mena, and E. Solano-Bes, "Generic rules for the discovery of subsumption relationships based on ontological contexts," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Intell. Agent Technol. (WI-IAT)*, Dec. 2015, pp. 309–312.

[30] F. Bobillo, C. Bobed, and E. Mena, "On the generalization of the discovery of subsumption relationships to the fuzzy case," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2017, pp. 1–6.

[31] J. Gracia and K. Asooja, "Monolingual and cross-lingual ontology matching with CIDER-CL: Evaluation report for OAEI 2013," in *Proc. 8th Int. Workshop Ontol. Matching (OM)*, 2013, pp. 1–8.

[32] N. Mitrovic, J. A. Royo, and E. Mena, "ADUS: Indirect generation of user interfaces on wireless devices," in *Proc. 7th Int. Workshop Mobility Databases Distrib. Syst. (MDDS)*, 2004, pp. 662–666.

[33] N. Mitrovic, C. Bobed, and E. Mena, "Dynamic user interface architecture for mobile applications based on mobile agents," in *Proc. 5th Int. Workshop Methods, Eval., Tools Appl. Creation Consumption Structured Data e-Soc. (Meta4es)*, 2016, pp. 282–292.

[34] R. Yus, S. Ilarri, and E. Mena, "Real-time selection of video streams for live TV broadcasting based on query-by-example using a 3D model," *Multimedia Tools Appl.*, vol. 74, no. 8, pp. 2659–2685, Apr. 2015.

[35] H. Huang, G. Gartner, J. M. Krisp, M. Raubal, and N. Van de Weghe, "Location based services: Ongoing evolution and research agenda," *J. Location Based Services*, vol. 12, no. 2, pp. 63–93, Apr. 2018.

[36] J. Raper, G. Gartner, H. Karimi, and C. Rizos, "Applications of location-based services: A selected review," *J. Location Based Services*, vol. 1, no. 2, pp. 89–111, Jun. 2007.

[37] R. Beaubrun, B. Moulin, and N. Jabeur, "An architecture for delivering location-based services," *Int. J. Comput. Sci. Netw. Secur.*, vol. 7, no. 7, pp. 160–166, 2007.

[38] M. D'Souza and V. S. Ananthanarayana, "Decentralized registry based architecture for location-based services," in *Proc. 6th Int. Conf. Ind. Inf. Syst.*, Aug. 2011, pp. 136–139.

[39] P. Bellini, P. Nesi, M. Paolucci, M. Soderi, and P. Zamperlin, "Snap4city platform: Semantic to improve loca-tion based services," in *Proc. 15th Int. Conf. Location-Based Services*, 2019, p. 237.

[40] P. Nesi, C. Badii, P. Bellini, D. Cenni, G. Martelli, and M. Paolucci, "Km4City smart city API: An integrated support for mobility services," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, May 2016, pp. 1–8.

[41] K. Haruna, M. Akmar Ismail, S. Suhendroyono, D. Damiasih, A. Pierewan, H. Chiroma, and T. Herawan, "Context-aware recommender system: A review of recent developmental process and future research direction," *Appl. Sci.*, vol. 7, no. 12, p. 1211, 2017.

[42] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, "An ontology-based context model in intelligent environments," in *Proc. Commun. Netw. Distrib. Syst. Modeling Simulation Conf. (CNDS)*, 2004.

[43] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, pp. 263–277, 2007.

[44] H. Nacer and D. Aissani, "Semantic Web services: Standards, applications, challenges and solutions," *J. Netw. Comput. Appl.*, vol. 44, pp. 134–151, Sep. 2014.

[45] H. H. Wang, N. Gibbins, T. Payne, A. Patelli, and Y. Wang, "A survey of semantic Web services formalisms," *Concurrency Computation: Pract. Exper.*, vol. 27, no. 15, pp. 4053–4072, Oct. 2015.

[46] L. D. Ngan and R. Kanagasabai, "Semantic Web service discovery: State-of-the-art and research challenges," *Pers. Ubiquitous Comput.*, vol. 17, no. 8, pp. 1741–1752, Dec. 2013.

ROBERTO YUS received the Ph.D. degree in computer science from the University of Zaragoza, Spain, in 2016, researching on issues related to semantic data management in distributed and mobile environments. During his Ph.D., he was a Visiting Researcher for a year in the ebiquity research group at the University of Maryland, Baltimore, and at the Information Systems Group, University of California, Irvine, USA. He is currently a Postdoctoral Scholar with the Department of Computer Science, University of California, Irvine. His current research interest includes privacy issues in data management on the Internet of Things.

CARLOS BOBED received the degree in computer science and the Ph.D. degree in computer science from the University of Zaragoza, Spain, in 2005 and 2013, respectively. After taking his Ph.D., he was a Visiting Researcher in the University of Aberdeen, U.K., and the University of Rennes, France, where he was a Postdoctoral Researcher as well. He is currently a Research Scientist / Engineer with everis / NTT Data. He is the author of several research publications in international journals and conferences, and has served as a reviewer of international journals and as a Program Committee member of many international conferences. His research interests include semantic Web and its associated technologies, mobile computing, and natural language processing.

EDUARDO MENA is currently an Associate Professor with the Department of Computer Science and System Engineering, University of Zaragoza, Spain, where he is also the Head of the research group Sistemas de Informacion Distribuidos (SID) (Distributed Information Systems). He is the author of several research publications in international journals, conferences, and workshops. Also, he is author of the book *Ontology-Based Query Processing for Global Information Systems* which describes the OBSERVER system, considered one of the classic projects from mid-90's in the area of global information systems, in collaboration with Dr. A. Sheth. His work has resulted in more than 180 publications and (according to Google Scholar) he accumulates more than 4000 citations with an H-index of 29. He has also served as a reviewer of international journals, and as a Program Committee member of many international conferences and workshops. Since 1991, he has been developed his research work in the area of query processing in distributed and heterogeneous environments, with special emphasis on the use of knowledge representation systems based on Description Logics to describe ontologies, and mobile agent technology and its application to mobile computing. His research interest areas include interoperable and heterogeneous data systems, semantic Web, ontologies and knowledge representation languages, mobile computing, and the Internet computing.

● ● ●