Fast prototyping and deployment of Context-Aware Smart Outdoor Environments

Pouyan Ziafati and Fulvio Mastrogiovanni and Antonio Sgorbissa

***Abstract*—The article describes a tool for the fast prototyping and deployment of context-aware applications, in particular to welcome visitors in urban areas.**

The system has been conceived to guarantee continuous access to everybody, everywhere, at any time, and therefore it does not rely on any special device to connect visitors to the intelligent environment. In fact, we assume that visitors are equipped with low-end mobile phones embedded with bluetooth technology, which provide approximate positioning information. In spite of this, the system must be able to assess the current context in terms of user location, preferences, current activity, and to suggest city-tours and activities which meets the most the visitor's expectations.

The article shows how, basing on Google maps API and OWL-DL ontologies, the rapid prototyping and rapid deployment of outdoor context-aware applications based on bluetooth messaging and positioning information can be achieved.

I. Introduction

Context-awareness is an important aspect of smart environments, enabling them to ubiquitously gather and use context information to seamlessly provide the most relevant services to users. Context-awareness increases the richness of communication in human-computer interaction and enables adaptability according to the current state of the environment, hence resulting in more usability and effectiveness of applications. Specially in outdoor smart environments in which the user's context is changing rapidly, the automatic adaptation of the system's behaviour to the current context is highly desirable. In the most cited definition in the literature [1] context is defined as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" and context-awareness is defined as "using context to provide relevant information and / or services to the user, where relevancy depends on the user's task".

Context-awareness is an active area of research in pervasive and mobile computing [10], [11], [12], [13], and several outdoor context-aware systems, specially in the domain of tourist city guide, have been developed during the last few years. However, most of these systems mainly focus on applications rather than context-awareness technology aspects and utilize context information in an ad-hoc manner. A survey of a number of context-aware tourist guide systems [2] shows that these systems are weak with respect to standardization, reusability, extensibility and interoperability. Moreover they do not provide a good support for: a) fully utilization of time and history context data; b) combining context data to acquire higher level context information; c) dynamic adaptation of system functionalities; d) push based access to context data; e) incorporating external context information.

The general advantages of advanced knowledge representation and reasoning techniques, such as Ontology-based and rule-based approaches, in developing context-aware applications have been already recognized by the research community [3], [14]. Formal specifications of these techniques such as their knowledge representation expressiveness and reasoning capabilities are well defined in theories. However, to fully explore the applicability of these techniques for developing context-aware applications, their advantages and limitations should be evaluated against real world scenarios. Moreover, aside from some issues such as scalability, we believe that one important reason why such techniques are not widely used for developing real world context-aware applications is the big gap which exists between knowledge engineering experts who develop context-aware applications and non-technical domain experts who are deploying these systems in their specific application domains. User-friendly tools are needed to allow non-technical domain experts to configure and extend the context-aware functionalities of the system and test its behaviour in different situations before the actual deployment. Bridging this gap by providing such tools would enable and encourage a wider use of context-awareness techniques in a wider area of application domains.

Another issue in developing context-aware smart outdoor environments is a lack of suitable prototyping tools and simulation frameworks to be used in the development phase, testing and rapid prototyping of such systems. In many context-aware smart outdoor environment systems, such as context-aware outdoor tourist guides, application scenarios are considered in a vast geographical areas, comprising embedded and mobile sensors and actuators, as well as mobile users. Therefore the design and development of software for such systems without simulation and rapid prototyping tools are highly costly and time consuming.

This paper addresses the above issues and provides design methodologies and software tools for rapid prototyping and development of context-aware smart outdoor environments. The main contribution of our work is two-fold.

First we have designed and developed a Context Module which represents context information in the OWL-DL

P.Ziafati and F. Mastrogiovanni and A. Sgorbissa are with the Department of Communication, Computer and System Sciences (DIST), University of Genova, Via Opera Pia 13, 16145, Genova, Italy; e-mail: pziafati@gmail.com, {fulvio.mastrogiovanni, antonio.sgorbissa}@unige.it

ontology language[1] and utilizes semantic web rule language(SWRL)[2] and Semantic Query-Enhanced Web Rule Language(SQWRL) [4] for context reasoning and querying. OWL-DL and its associated rule languages, SWRL and SQWRL, have been used in developing other context-aware applications before [5], [6], [7], [8], [9]. However, those systems do not provide many details about using these technologies, and do not discuss the related benefits and limitations that have been faced during development. On the other hand, we will present benefits and limitations of these technologies in more details, by considering a complex scenario and discussing about different issues related to context modelling and reasoning using these techniques.

Our second contribution is providing a tool for rapid prototyping and development of context-aware smart outdoor environments, most notably the development of an interactive simulation software that can be used both for design and simulation of such systems. To this end, we will show how different existing technologies such as Google map API, and Protégé ontology editor and API can be leveraged to enable the rapid development and prototyping of powerful context-aware smart outdoor environment applications with high-level context representation and reasoning capabilities. As a case study, we consider the design of an outdoor tourist guide and present a proof of concept prototype application which is used to explain and evaluate our approach.

The paper is organized as follows: Section II presents the tourist guide case study. Section III describes the Context Module. Section IV discusses about different challenges of using OWL-DL and its corresponding rule languages for developing context-aware systems. Section V presents the simulator and the usability test of the system. Finally, Section VI is devoted to conclusions.

II. Case Study: A Context-Aware Outdoor Tourist Guide

To describe and evaluate our approach, we consider a case study which has been fully developed and tested in a simulated environment, i.e., a context-aware outdoor tourist guide. The system is aimed to provide context-aware information to tourists in an urban area. We assume that tourists are equipped with bluetooth-enabled mobile devices (e.g., mobile phones), and that a number of bluetooth beacons are deployed in the environment in order to recognize the presence of mobile devices held by tourists and communicate with them within a short range ($\approx$ 10 meters). A bluetooth beacon, called agent in our scenario, can be fixed in a location of interest or even embedded in a mobile robot, and it is referred to as a beacon or a robot accordingly.

The basic idea is to use short-range bluetooth communication between mobile phones and agents deployed in places of interest (close to churches, museums, shops, road-junctions, etc.) for two purposes.

- Localize tourists. GPS, even when available on mobile phones, cannot be used in the narrow streets of old towns because of satellite occlusions. However, short-range communication can be used for this purpose very effectively: when a mobile phone is detected by an agent, the tourist is necessarily in the neighbourhood of the agent.
- Sending messages to tourists. When a tourist is within communication range, the agent can send one or more messages to the mobile phone, whose content can include text and pictures and can vary depending on the context.

[1]http://www.w3.org/TR/owl-guide
[2]http://www.w3.org/Submission/SWRL/

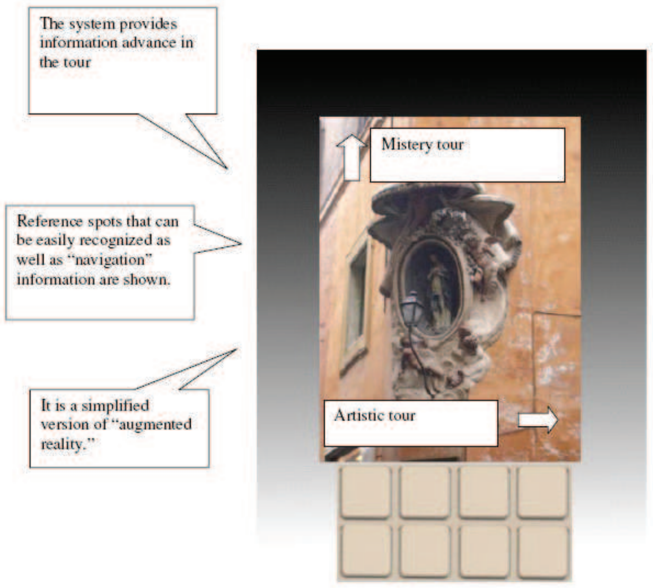

The system provides information advance in the tour
Mistery tour
Reference spots that can be easily recognized as well as "navigation" information are shown.
It is a simplified version of "augmented reality."
Artistic tour

Fig. 1. Walking directions.

Then, the system provides the following services.

- Each agent provides location-aware and customized information about that place (e.g., the history of a church or a square, nearby restaurants and night–clubs, etc., customized to a tourist language, age, etc.). Robots are mobile and can present different locations over time.
- Each tourist can choose to follow a specific tour such as an artistic tour, historic tour, mystery tour, shop tour, etc. Each tour is defined as an ordered sequence of locations (i.e., corresponding to agents). When following a tour, in every location, the system provides the tourist with walking directions towards the location to visit next.
- The system provides a context-aware advertisement service, in which advertisements are delivered to the right tourists in the right locations at the right times.

As an example, Figure 1 shows an hypothetical message sent to a tourist which is walking along the *mystery tour*. When the tourist is within a communication range, the agent sends a message which includes a picture and some text. Walking directions are given in a subjective perspective, which is known to be more human-like and, hence, user-friendly than standard top view maps. Notice also that, in this location, the *mystery tour* intersects with the *artistic tour*, and this information is made available to the tourist.

Before the visit, the user can explicitly declare her / his own preferences, for example by registering on a web site at home or in a tourist information centre (or even during the visit itself, if the mobile phone is provided with functionalities for wireless internet navigation). However, the key idea here is that user preferences can be inferred by the system itself by simply monitoring sequences of activities performed by the user. By monitoring the time spent by every individual tourist in different locations at different times of the day, as well as the time spent while moving between one location and the other (which can be simply detected through short-range communication between agents and mobile device) the system tries to guess the preferences of the user. Such preferences are then included in the context of a tourist, and the system customizes its behaviour accordingly to better suits the present context of each tourist.

In the following we show an incomplete list of types of context information provided by the Context Module of the system.

- Tourist preference: *language* (e.g., *Italian*, *English*, etc.), *age* (e.g. *Child*, *Adult*, etc.), *storyTellingStyle* (e.g., *fun*, *mystery*, *artistic*), *tourType* (e.g. *General*, *Art*, *Church*, *Shop*).
- Tourist location: *locationType*(e.g., *Bar*, *Church*, etc.), individual locations (e.g. *robot1*, *beacon3*, etc.).
- Time of the day: *Morning*, *LunchTime*, *Afternoon*, *Evening*, *DinnerTime*, *Midnight*, *Morning&Evening* (time is *Morning* or *Evening*), *DayTime* (time is *Morning* or *LunchTime* or *Evening*), *NightTime* (*DinnerTime* or *Midnight*)
- Tourist history: a sequence of location where a tourist has been in a specific time of the day.
- Summary of tourist history: the amount of total time and the number of times that a tourist has visited locations of a specific type during a specific time of the day:
  - *spendTimeArt* (the total amount of time a tourist has spent in art locations, including churches, museums, monuments), *spendTimeBarMorningEvening* (the total amount of time a tourist has spent in bar locations during morning and evening), *spendTime* (the total amount of time tourist has spent in all locations), *spendDistance* (the total distance a tourist has walked while visiting locations in the city)
  - *numberTimeArt* (the number of art locations that a tourist has visited, including churches, museums, monuments), *numberTimeBarMorningEvening* (the number of bar locations a tourist has visited during morning and evening), *numberTime* (the total number of locations that a tourist has ever visited), *numberDays* (the number of days a user has spent in the system)
- Tourist type: high level context information which the system can infer about a tourist such as: *ChurchLover*, *ArtLover* (the tourist is *ChurchLover* and / or *MonumentLover* and / or *MuseumLover*), *RestaurantLover*, *AntiqueShopLover*, *ShopLover* (the tourist is *AntiqueShopLover* and / or *sexShopLover*), *SlowWalker*, *FastWalker*, etc.

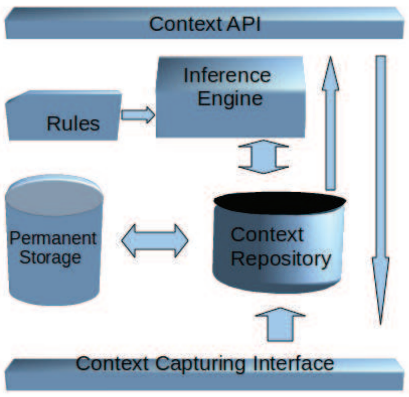


Fig. 2. System Architecture.

Finally, to better clarify the concepts above, we present three use cases to describe our approach to context assessment in rest of the paper.

1) Classify a tourist as an *ArtLover* if tourist has spent in overall more than 3 hours in art locations including churches, monuments and museums or if the tourist has spent more than 40% of her time visiting art places and it is more than 1 hour that she is visiting places.
2) Classify a tourist as a *BarLover* if she has spent in average more than 3 hours per day in bars or she has spent in overall more than 90 min. in bars during morning or evening times.
3) If the time is *Evening*, send *barAdvertiseMessage2* to tourists who are *BarLovers* and currently in *Bar* locations and whose preferred language is *Italian*.

III. Context Module

The Context module is responsible for acquiring, representing and storing context information, reasoning on high level context information and providing context-aware functionalities. The design of the Context Module follows the general layered structure architecture, common in the majority of context-aware systems [10], [11] which consists of the following abstract components (Figure 2)

A. Context Capturing Interface

The Context Capturing Interface is responsible for acquiring context data from logical and physical sensors, performing the required sensor data pre-processing and representing them in a suitable data structure to be stored in the context repository. The reasonable architectural approach here is to separate concerns between acquiring context data, and processing and using them [12]. There are 3 main data acquisition models [13]: Widgets, Network Services and Blackboard Architecture. Our design follows the blackboard model which is the most loosely coupled model and enables the complete decoupling of the Context Module from the actual data acquisition method, providing the most re-usability and extensibility of the Context Module. The Blackboard architecture adopts a data-centric rather than process-centric point of view in which processes post messages to a common shared message board and subscribe for relevant notifications.

B. Context Repository

The Context Repository is used to represent, manage and store context information. The choice of a context modelling approach determines the core design of the context-aware system. State-of-the-art approaches to context modelling and reasoning show that ontology-based approaches are the most promising ones for context modelling and reasoning in pervasive computing [3], [14]. Ontologies support the representation of complex relationships and dependencies among context data which is suitable for recognition of high-level context information. Besides, the availability of a set of development tools and standards for ontologies increases re-usability and ease of development. Specially, thanks to the Semantic Web, standard ontology languages and development tools have gained maturity over the past years.

In our system, context is modelled using Web Ontology Language(OWL) and stored using Protégé OWL repository which provides permanent storage of ontology in a database backbone. Using Protease OWL editor we are able to graphically present and edit the ontology. We also use the Protégé OWL JAVA API to manipulate the ontology (context information) in the application program. OWL is a W3C standard language for defining ontologies. It is based on Description Logics [17] which allows OWL to exploit the considerable existing body of DL reasoning such as class consistency and consumption. An OWL ontology may include descriptions of classes, properties and their instances.

Figure 3 depicts a part of our outdoor tourist guide ontology. The ontology contains more than 50 classes, 15 object properties and 30 datatype properties. In this ontology: *Agent*, *Tourist* and *Message* classes represent the corresponding concepts in the tourist guide scenario. Each tourist keeps track of her history of visited places and of the received messages using the *hasHistoryUnit* and the *hasMessage* object properties. The *PresentTourist* class represents the tourists that are currently present in the system, encoding information about which user is currently near which *Agent* and since when. The *HistoryUnit* class represents the history of tourists visiting an *Agent* (i.e., a location) or passing from one *Agent* to another *Agent* within the corresponding period of time. It also stores other information about the visit, such as the type of the locations visited and the time of the day (i.e., *Morning*, *Evening*, etc). The *MessageQueue* class represents a set of individuals, each encoding information about which *Message* should be delivered to which *Tourist* by which *Agent*. The *LocationType* class and its subclasses represent a hierarchy of location types corresponding to individual *Agent*s. A single individual of each of the leaf nodes of the *LocationType* class hierarchy is created and permanently stored in the repository. The set of these individuals forms the range of admissible values for the *hasLocationType* property. The *Time* class and its subclasses represent a hierarchy of the timing periods. Nodes of this hierarchy correspond to specific periods of time, such as *Morning* which is from 6 am to 11 am. A single individual of each of the leaf nodes of *Time* class hierarchy is created and permanently stored in the repository. The set of these individuals forms the range of admissible values for the *hasTimeOfTheDay* property. There is also an individual named *currentTime* which encodes information about the current timing period of the day: the Context Module sets its class type according to the current time (e.g. if the time is 7am which corresponds to *Morning* time, the *currentTime* individual is defined as an individual of the *Morning* class). Finally, there are a number of classes (e.g. *BarLover*, *ArtLover*, etc.) representing different tourist types. When the system infers that a tourist individual has a type, it will add the corresponding tourist type class to the asserted types of the tourist.

C. Context Reasoning and Querying

The Context Reasoning and Querying component is responsible for inferring new context based on the current contextual information in the context repository and the new contextual data acquired through the context capturing interface. We utilize SWRL and SQWRL for performing reasoning and querying on context information. We use Protégé editor for writing SWRL and SQWRL rules and utilize JESS[3] as our rule engine.

SWRL is an expressive OWL-based rule language which allows writing rules that can be expressed in terms of OWL classes, properties, individuals, and data values. It extends the set of OWL axioms to include Horn-like rules to provide more powerful deductive reasoning capabilities than OWL alone. SWRL offers, through built-ins, a number of mathematical, time related and comparison functions that are necessary for reasoning. It also offers support for composite relationships on instance level. SQWRL is a SWRL-based language for querying OWL ontologies. In addition to providing a core language which uses a SWRL rule antecedent as a pattern specification and replaces the rule consequent by selection and formatting operators, SQWRL provides a selection of collection operators that provide advanced grouping and aggregation functionality, and limited forms of negation as failure, and disjunction.

D. Context API

The context API: provides an interface for context-aware applications to actually utilize contextual information.

E. Context Module Work Flow

Considering the outdoor tourist guide scenario, the cyclic work flow of the Context Module is as follows: a) new context information provided by sensors is acquired (e.g., which *Tourist* is currently in proximity of which *Agent*); b) according to the new sensor data and the current time of the system, different segments of context information are updated including the class type of the *currentTime* individual, *PresentTourist* and *HistoryUnit* individuals; c) the previously inferred information describing tourist types are deleted, and a set of SQWRL are performed to query tourist histories and

[3]http://www.jessrules.com/

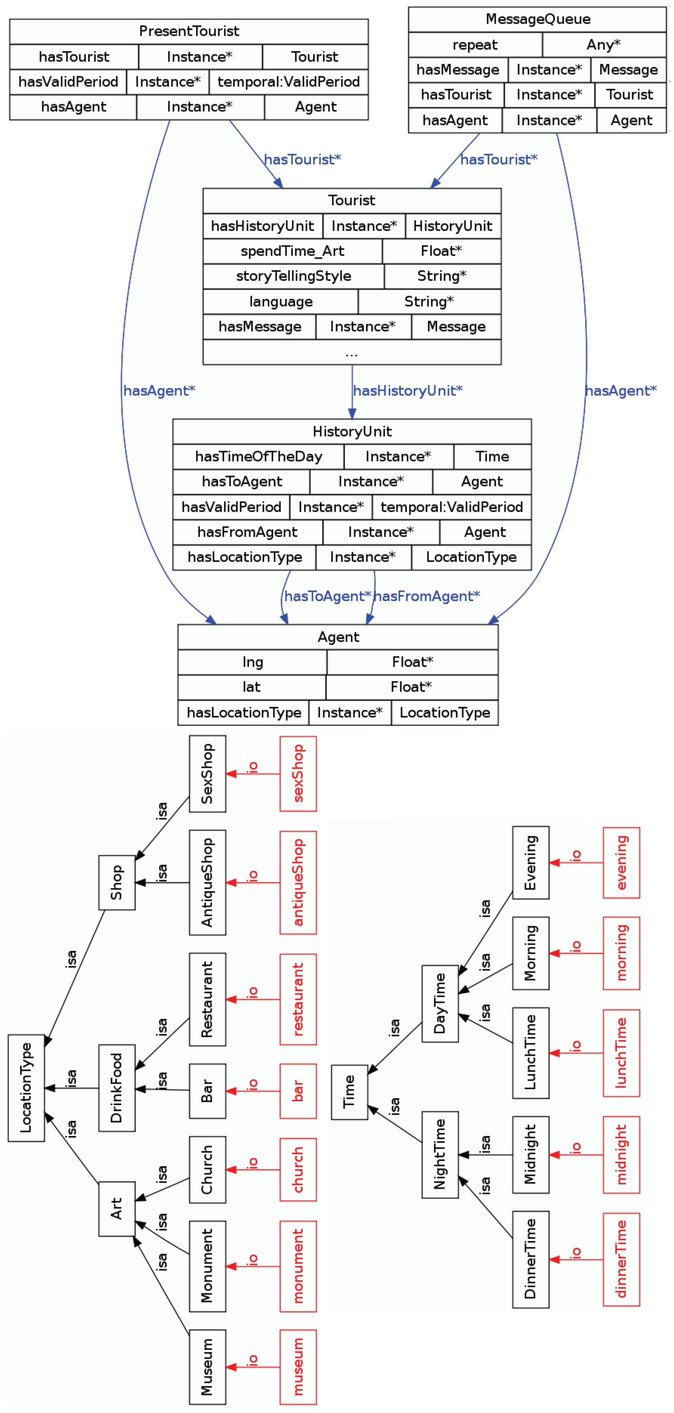


Fig. 3. Tourist Guide Ontology, depicted using Ontoviz

update the repository with corresponding information; d) A set of SWRL rules are performed which: infer tourist types based on the information obtained from querying tourists histories, infer which *Message*s should be sent to which *Tourist*s by which *Agent*s, generating a set of *MessageQueue* individuals encoding these information e) Context API are accessed to notify the corresponding *Agent*s to deliver the messages to the *Tourist*s.

IV. Challenges and Solutions

In this section we consider the case study described in Section 2 and discuss about different challenges of OWL-DL based context representation and reasoning.

A. *Utilizing time*

In many cases such as in use case 1, we need to reason upon the tourist history, which involves performing algebraic operations and comparisons on temporal data.

Ontology languages such as OWL typically provide minimal support for modelling the complex temporal information. OWL, for example, provides no temporal support beyond allowing data values to be typed as basic XML Schema dates, times or duration[4]. SWRL includes operators for manipulating these values, but its operators work at a very low level. There are no standard high-level mechanisms to consistently represent and reason with temporal information in OWL. To deal with this issue, We use a methodology and a set of tools introduced in [15] for representing and querying temporal information in OWL ontologies. The approach uses a lightweight temporal model to encode the temporal dimension of data. It also defines a library of SWRL built-ins to perform temporal operations on information described using this ontology. Using this temporal ontology, we are able to represent temporal information such as a time instance or a time period and perform different temporal operation such as computing duration of a time period, adding time periods, compare time instances or durations, etc.

B. *Reasoning on Conjunction of atoms*

In many cases, we need to perform reasoning on conjunction of atoms which is not supported by SWRL. For example, in use case 1, we want to write a query on the tourist history which sums up the time she has spent in art-related locations(i.e., *Church* or *Museum* or *Monument* locations). To get around this issue, we presents location types in a class hierarchy and create a single and permanent individual of each *LocationType* class in the repository. In the tourist history of visited places, location types of these places are encoded using the *hasLocationType* property whose value can be a generic individual of one of the classes derived by *LocationType*. This way of encoding location information allows for reasoning on conjunction of location types through subsumption, as an individual of a class is also an individual of the corresponding parent class.

C. *Open world assumption and monotonicity*

OWL-DL makes the open world assumption, which means that the truth-value of a statement is independent of whether or not it is known. It is the opposite of the closed world assumption, which states that any statement that is not known to be true is assumed to be false. Closely related to the open world assumption, OWL and SWRL supports monotonic inference only which means adding new information never

[4]http://www.w3.org/XML/Schema

falsifies a previous conclusion. These OWL and SWRL characteristics impose different concerns and issues when working with context information, some of which are described below.

Context information which changes in time: due to the monotonicity assumption of OWL-DL, the property values can not be overwritten. Therefore when assigning a new value to a property, the new value is added to the set of values for that property rather than overwrite them. The intuitive approach to treat a context information that changes in time is to add a time stamp to every of its value. In this way we are able to represent the values of a property as they evolve in time, and to reason about its actual value at a specific time. However, in some cases, we want for example to reason about the most recent value of the property, but the problem is that operators such as Min or Max are not available to be used in the body of SWRL or SQWRL rules. A possible solution for this problem is to treat such context information manually outside of the ontology, by overwriting old values with new ones. However to keep the consistency of the ontology, all the facts that have been inferred from the deleted facts should be also deleted. Such retraction might be sometimes difficult if not impossible!

Reasoning on the results of queries: in some cases such as in use case 1, we want to perform some queries and then reason on the results of such queries. But there is no way to do that in SQWRL, as query operators in SQWRL should always appear in the head of the rule. Moreover, SQWRL provides no way of accessing the information that it temporarily accumulates while executing a rule, and therefore the corresponding query results cannot be written back to the ontology. There is no way, for example, to insert the result of a computed aggregate count back into the ontology, since such a mechanism could invalidate OWL-DL's open world assumption and lead to non-monotonicity. To address this issue, we use an engineering solution at the implementation level by providing a mechanism to store the results of SQWRL rules back to the ontology. Although this approach is against OWL-DL semantic, it is safe when used in the controlled way in which a non-monotonic assumption is applied to a specific property, ensuring that it can have only one value and any new value rewrites the previous one. We have developed an ontology and a Java library to perform this operation in an automatic and safe way. The ontology is used to encode information about which property values should be updated based on the results of which queries. The Java library provides a function which reads this ontology, performs corresponding queries and write their results back to the ontology. Following this approach, the Context Module provides different context information about the summary of tourists history as explained in section 2.

Negation as a failure: A further consequence of SWRL's monotonicity is that negation as a failure is not supported. For example in use case 4 we want to reason on the context information and send appropriate messages to tourists, but avoid sending repeated messages. To do this, we store the history of sent messages. But it is impossible to write a rule which says: "send a message to a tourist if the message does not exist in the set of sent messages stored for that tourist". To get around this issue, we have defined the *hasRepeat* datatype property for the *MessageQueue* class and defined a rule which assign the *yes* value to this property, if the tourist has the same message in her sent message history. The Context API examine this value and only deliver messages of the *MessageQueue* individuals which do not have the *yes* value for their *hasRepeat* properties. It should be noted that in our approach, by writing rules to create and add *MessageQueue* individuals to the repository, we violate the safety condition of SWRL by using unbound variables in the head of the rules. Also, by writing the rule which assigns the *yes* value to the *hasRepeat* property of *MessageQueue* individuals we violate again the monotonicity assumption as before.

V. Rapid development and prototyping

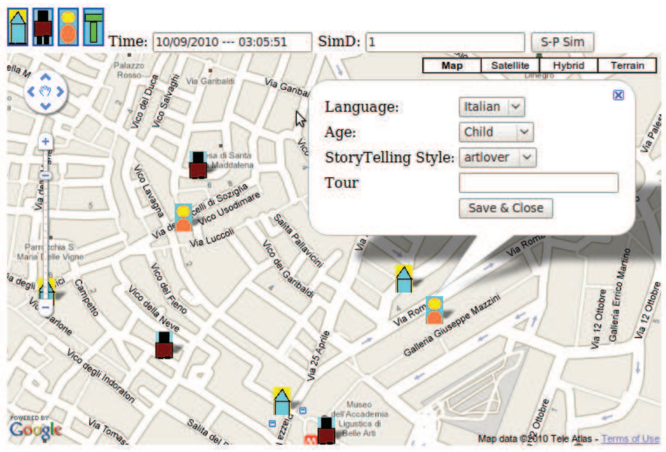
This Section presents a simulation tool and discusses about rapid prototyping and development of context-aware outdoor tourist guides using this simulator and the Context Module. To evaluate our approach, we have conducted a usability test for performing the following tasks: a) designing a tourist guide and testing it using the simulator, b) extending context-aware services of the Context Module, and c) extending the context recognition functionalities of the Context Module. Users have been provided with a 15 pages user guide of the system, explaining, in an easy to understand language for non technical users, how to perform the above tasks. Then they were asked to perform different assignments related to the above tasks, and finally they were asked to answer 32 questions, to evaluate the usability of the system. In the following, the simulator and the assigned tasks are first described, and then usability test results are presented and discussed.

A. Simulation tool

Figure 4 shows a snapshot of the simulator. It is a web-based tool, utilizing Google Map API to present and simulate the tourist environment. Using AJAX technology, it exchanges information with the Context Module in an asynchronous way and provides a user friendly GUI for the following functionalities.

- Adding and removing *Agent*s, *Tour*s and *Tourist*s on and from the map.
- Starting and pausing the simulator.
- Changing the simulation time and the simulation duration time.
- Providing draggable tourist icons to simulate and test the context-aware functionalities of the Context-Module

The first item deserves a discussion. By clicking on the *Agent* or *Tourist* icon on the top left side and then clicking on the map, a corresponding OWL-DL individual is created inside the context repository and also shown on the map. While creating these entities, a pop up window appears which allows the user to enter the corresponding information such as *Language*, *age*, *storyTellingStyle* and preferred *tourType* for the *Tourist*. Right-clicking on an entity will delete it. A tour can be defined by clicking on the corresponding icon, entering the tour name in the window which will pop-up and then clicking on a sequence of *Agents* (i.e., locations) and at the end clicking on the tour icon again. When creating a tour, the simulator creates necessary OWL-DL individuals and rules



Time: 10/09/2010 --- 03:05:51

SimD: 1

Language: Italian

Age: Child

StoryTelling Style: artlover

Tour

Fig. 4. Simulator window showing the historic centre of Genova.

to provide the tour guide functionality described in Section 2. This includes: encoding the tour information using individuals of a class named *TourUnit* (each representing a path segment from an *Agent* to another *Agent*), creating *Messages* (with text and pictures) containing walking directions, and a SWRL rule for sending these messages to the proper tourists.

B. Extending context-aware services of the system

This subsection describes how to write SWRL rules to utilize the available context information in the Context Module in order to provide the context-aware messaging service for tourists. Although in the tourist guide scenario messages are considered as multimedia information to be presented to tourists, in general, they can be seen as encoded information which can be interpreted by the context API to produce different context-aware functionalities and services for the applications.

Rule1 shows a SWRL rule which results in sending *Message1* to every tourist, present in the system, and can be interpreted as: 'send *Message1* to each present tourist *pt* which corresponds to tourist *t* and is near agent *a*'. In fact, this rule generates proper *MessageQueue* individuals encoding such information which are later interpreted by the Context API to perform the actual delivery of messages.

$$\begin{aligned}
&Rule1 : PresentTourist(?pt) \wedge hasTourist(?pt,?t) \wedge \\
&hasAgent(?pt,?a) \wedge Message(?m) \wedge \\
&sameAs(?m, Message1) \wedge \\
&swrlx : createOWLThing(?mq,?m) \wedge \\
&\rightarrow MessageQueue(?mq) \wedge hasMessage(?mq,?m) \wedge \\
&hasTourist(?mq,?t) \wedge hasAgent(?mq,?a)
\end{aligned}$$

The following describes how Rule1 can be easily extended, while fast-prototyping a tourist-guide system, to reason about different context information available in the Context Module.

- Sending a different *Message* than *Message1*: obviously to send a different message, for example *Message2*, *Message1* in the template Rule1 should be changed to name of the new message.
- Reason on tourist preferences: to reason on tourists who have the language *L*, its enough to add $\ldots \wedge language(?t,?x1) \wedge swrl : equal(?x1, "L") \wedge \ldots$ to the Rule1 template, which can then be read as 'send *Message1* to each present tourist *pt* which corresponds to tourist *t*, who speaks the language *x1* which is equal to *L*'. Writing rules for reasoning on tourist' *age*, *storyTellingStyle* and *tourType* is done in a similar way.
- Time: to reason on the current timing period of the day, we can reason on the class type of the *currentTime* individual. For example, to send a message to the present tourists if the current time is morning, we add $\ldots \wedge Morning(currentTime) \wedge \ldots$ to the body of Rule1.
- To reason on the type of a tourist, we should check the class type of the tourist in the body of the rule. For example to send the message to all *artLover* tourists, we add $\ldots \wedge hasTouristType(?t,?tt) \wedge ArtLover(?tt) \wedge \ldots$ to the body of Rule1, which can be read as 'send *Message1* to every tourist *t* who has a type *tt* which belongs to the class *artLover*'.

Rule2 shows an example of combining the above context information, by sending a *message3* to all present tourists whose language is *English* and are *AntiqueShopLovers* and *slowWalker*, if they are close to an *AntiqueShop* location and the time is *Evening*.

$$\begin{aligned}
&Rule2 : PresentTourist(?pt) \wedge hasTourist(?pt,?t) \wedge \\
&hasAgent(?pt,?a) \wedge Message(?m) \wedge \\
&sameAs(?m, Message3) \wedge language(?t,?l) \wedge \\
&swrl : equal(?l, "English") \wedge AntiqueShopLover(?t) \wedge \\
&SlowWalker(?t) \wedge hasLocationType(?a,?la) \wedge \\
&AntiqueShop(?la) \wedge Evening(currentTime) \wedge \\
&swrlx : createOWLThing(?mq,?m) \wedge \\
&\rightarrow MessageQueue(?mq) \wedge hasMessage(?mq,?m) \wedge \\
&hasTourist(?mq,?t) \wedge hasAgent(?mq,?a)
\end{aligned}$$

C. Providing more context recognition functionalities

As a complex context recognition use case, we consider the use case of inferring the tourist type based on the history of a tourist. The context information provided by the Context Module about the history of tourists in the system has been described in section 2. SWRL built-ins such as $swrlb : add$, $swrlb : divide$ and $swrlb : lessThanOrEqual$ can be used to perform the necessary mathematical operations on these information to reason about the tourists types. For example Rule3 and Rule4 classify a *BarLover* tourist as defined in section 2.

$$\begin{aligned}
&Rule3 : Tourist(?t) \wedge \\
&spendTimeBarMorningEvening(?t,?stbme) \wedge \\
&swrlb : greaterThan(?stb, 90) \rightarrow BarLover(?t)
\end{aligned}$$

$$\begin{aligned}
&Rule4 : Tourist(?t) \wedge spendTimeBar(?t,?stb) \wedge \\
&spendDays(?t,?sd) \wedge swrlb : divide(?astb,?stb,?sd) \wedge \\
&swrlb : greaterThan(?astb, 180) \rightarrow BarLover(?t)
\end{aligned}$$

D. Usability Test

The usability test has been conducted on 4 users, 2 of them having the knowledge of software development and 2 of them just having basic familiarity with using computers. None of the users had previous familiarity with context-aware systems. The purpose of the system and the requested tasks were clear for all users.

The software developers could perform all tasks using the tutorial. The other two could easily use the simulator but for the tasks of extending the systems functionalities and services needed more help. The simulator was ranked very user friendly by all users. All users considered the context information provided by the system for reasoning about tourist types, as well as the corresponding messaging service, highly useful. Learning how to extend context recognition functionalities and messaging service was easy for the software developers and relatively easy for the others.

Due to the lack of time, the usability test was conducted on a small number of users. Moreover it would have been more interesting to have the system tested by experts in the tourist domain, which has not been possible up to now. However the initial results clearly show the advantages and usability of the system.

VI. CONCLUSION

This paper has shown how different available technologies can be leveraged to enable the rapid prototyping and development of context-aware smart outdoor environments. We presented the Context Module and the simulation software we have developed for an outdoor tourist guide case study. The conducted usability test has shown the usefulness of the provided tools in enabling even non-technical users to design a context-aware outdoor tourist guide, to test its behaviour, and to extend its context-aware and context recognition functionalities.

Moreover the case study has shown how OWL-DL, SWRL and SQWRL can be used for context modelling, reasoning and querying to provide complex context-aware functionalities. Several issues related to the development of context-aware applications using these technologies have been discussed. The Context Module extensively uses the time and history context information to reason about high level context information. Thanks to the blackboard model of data acquisition and the ontology-based context modelling, the Context Module can be easily extended to utilize external context information such as the weather context made available by some Web Services. Moreover the Context Module enables push-based accessing to context information, in which the Context Module itself can revoke the proper context-aware functionalities when new context information is inferred.

An important issue which we did not address in the paper is the scalability of the system. One scalability concern is the amount of tourist history data, which can become large and should be handled properly. The second and the most important scalability concern is the performance issue of SWRL and SQWRL in reasoning on a big amount of context data [16]. In the presented scenario, there are no context individuals involving more than one tourist at the same time, which makes it easy to distribute the reasoning load over a number of Context Modules, each handling a limited number of tourists. But if the social context is considered, e.g., by integrating context data from different tourists, scalability poses a serious issue and using advanced distributed and cascade reasoning approaches will be inevitable.

REFERENCES

[1] Dey, A.K. and Abowd, G.D., 'Towards a better understanding of context and context-awareness', Proceedings of the CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness (2000).

[2] Schwinger, W., GrÃijn, C., PrÃűll, B., Retschitzegger, W. (2008) 'Context-awareness in Mobile Tourism Guides', Handbook of Research in Mobile Multimedia, 2nd edition, Khalil-Ibrahim Ismail (ed.), Chapter XXXVII, IGI Global, USA, September 2008, pp. 534-552.

[3] Strang, T. and Linnhoff-Popien, C., 'A Context Modeling Survey', First International Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp, 2004.

[4] OâĂŹConnor, M. J. and Das, A. K., 'SQWRL: A Query Language for OWL'. In Proc. of 6th OWL: Experiences and Directions Workshop (OWLED2009), 2009.

[5] Liu, C.-H. Chang, K.L. Jason, J.Y. Hung, S.C., 'Ontology-Based Context Representation and Reasoning Using OWL and SWRL', 8th Annual Communication Networks and Services Research Conference, Montreal, Quebec, Canada, 2010.

[6] Lee, K. C. Kim, J. H. and Lee, J. H., 'Implementation of Ontology Based Context-Awareness Framework for Ubiquitous Environment', Int. Conference on Multimedia and Ubiquitous Engineering, pp. 278 - 282, April 2007.

[7] De, S., Moessner, K., 'Ontology-based Context Inference and Query for Mobile Devices'. IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2008, pp. 1 - 5, 2008.

[8] Ricquebourg, V. Durand, D. Menga, D. Marhic, B. Delahoche, L. LogÃľ, C. JollyDesodt, A.M., 'Context inferring in the Smart Home: An SWRL approach', Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops, AINAW '07, Niagara Falls, Canada, May 21-23, 2007.

[9] Chaari, T., Ejigu, D., Laforest, F., Scuturici, V., 'A comprehensive approach to model and use context for adapting applications in pervasive environments, The Journal of Systems and Software, v. 80, pp. 1973-1992, 2007.

[10] Baldauf, M. Dustdar, S. Rosenberg, F. (2007) 'A Survey On Context-Aware Systems', International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 63-277, Inderscience Publishers, 2007.

[11] Schmohl, R. and Baumgarten, U., 'Context-aware computing: a survey preparing a generalized approach'. In IMECS, Proceedings of the International MultiConference of Engineers and Computer Scientists 2008. International Association of Engineers, 2008.

[12] Dey, A.K., 'Providing Architectural Support for Building Context-Aware Applications', PhD Thesis, Georgia Institute of Technology, Georgia Institute of Technology, USA, 2000.

[13] Winograd, T., 'Architectures for context', Human-Computer Interaction (HCI) Journal, Vol. 16, No. 2, pp.401-419, L. Erlbaum Associates Inc., 2001.

[14] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D., A survey of context modelling and reasoning techniques, Pervasive and Mobile Computing, Volume 6, Issue 2, April 2010, Pages 161-180, Elsevier.

[15] O'Connor, M. J. DasA, A. K., 'Method for Representing and Querying Temporal Information in OWL', Biomedical Engineering Systems and Technologies (Selected Papers), Springer-Verlag, Communications in Computer and Information Science 127, pp. 97-110, 2011.

[16] Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K., 'Ontology Based Context Modeling and Reasoning using OWL'. In Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004), pp.18-22, Orlando, FL, USA, March 2004.

[17] Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002.