Mobile Task Computing: Beyond Location-based Services and EBooks

John Liagouris[1], Spiros Athanasiou[1], Alexandros Efentakis[2], Stefan Pfennigschmidt[3], Dieter Pfoser[1,2], Eleni Tsigka[3], Agnès Voisard[3]

[1]Institute for the Management of Information Systems
Research Center Athena
G. Bakou 17, 11524 Athens, Greece
{liagos|spathan|pfoser}@imis.athena-innovation.gr
[2]RA Computer Technology Institute
Davaki 10, 11526 Athens, Greece
{efedakis|pfoser}@cti.gr
[3]Fraunhofer ISST
Steinplatz 2, 10623 Berlin, Germany
{stefan.pfennigschmidt|eleni.tsigka|agnes.voisard}@isst.fraunhofer.de

Abstract. Mobile devices are a promising platform for content delivery considering the (i) variety of attached sensors, (ii) widespread availability of wireless networks, (iii) even increasing screen estate and hardware specs. What has been missing so far is the adequate coupling of content to those devices and their users actions. This is especially apparent in the area of Location-based Services (LBS), which, with few exceptions (e.g., navigation), have not fulfilled their predicted commercial success in mobile environments due to the following reasons: (i) content in typical LBS applications is still narrow and static, (ii) available methods and interfaces in mobile handsets for the discovery of available content are at best cumbersome (e.g., keyword-type search), and (iii) existing structured content available in LBS applications is hard to reuse. In this work, we propose the concept of task computing to complement and extend LBS as a means to enable the intuitive and efficient re-purpose, discovery, and delivery of rich content according to the users needs. Further, we establish the theoretical foundations of task computing and its application in the LBS domain. We also present a fully functional prototype iPhone application structured around the concept of task computing.

Keywords: task computing, location-based services, ontologies, ebooks

1 Introduction

Despite the growth in the number of mobile users, the widespread adoption of ubiquitous wireless networks, and the ever increasing capabilities of mobile handsets, the market of mobile services is still dominated by simple infotainment services. This is especially apparent in the area of Location-based Services (LBS),

which, with few exceptions (e.g., navigation), have not fulfilled their predicted commercial success in mobile environments.

Location-based services have emerged a few years ago to allow end users to obtain information based on some location, usually the position of the user. Such services, for instance mechanisms to answer a query such as "Where is the nearest subway station?" or "What are the exhibitions in the city today?" are currently receiving a great deal of interest. They manipulate the common aspects of location and time but also more complex notions such as the profile of a user.

General LBS for mobile users are either extremely narrow and static in their content offerings, or address the needs of business users (e.g., fleet management). However, within the past few years, certain important arrivals of LBS systems for everyday, casual users have emerged. These products and services focus either (i) on expanding the reach of geospatial information, or on (ii) coupling social network activities on a geospatial domain.

In particular, a class of services adequately represented by Google Maps/ Google Earth for mobile devices [4], focuses on the simple task of delivering standard geospatial products (such as maps), along with "free text"-based search for web content. Other similar services come from Microsoft Bing Maps [1], Nokia Ovi Maps [5], and so on. Though truly fascinating upon their arrival, these services have not changed the way users discover and consume content. Discovery still requires the users input from a hardware or on-screen keyboard. At best, a catalogue of available content is presented to the user, according to his/her search parameters and location. So the basic paradigm of how users manipulate geospatial information has not been altered. Rather, it has been poorly ported to mobile devices, simply by re-purposing available interface and information retrieval concepts for smaller screens. In essence, Google Maps for mobile is a literal port of standard Google Maps for mobile devices. The limited attentional span of mobile users, required ease of interaction, necessary provision if accurate content, is not dealt with. For example, a simple query for desktop-based users, such as "Is there a nice Greek restaurant within 1km from location X that costs less than 10 euro per person?" requires several searches, and the gradual filtering/refinement of web content from several sources (e.g., maps, guides, blogs, social networks). A mobile user does not have the time, nor adequate hardware resources to perform this search. Further, since mobile phones are a de facto democratized medium, they appeal to less tech-savvy users as well. Even to users that do not own a computer (e.g. developing nations), and who do not have any reference points on how geospatial content discovery and delivery is performed.

The second class of products and services focuses on overlaying a geospatial mantle upon social networks and interactions. Users can annotate content with location attributes, whether it concerns their own interaction (e.g., geotag profile messages), or points of interest (e.g. leave a restaurant review). Such services include Facebook, Twitter, Yelp, Foursquare, Layar, and Google Buzz. They have also been embraced by the users with great interest, but they still remain a shallow means for general geospatial content discovery and delivery.

While users can see where their friends are, read and submit reviews on POIs, etc., the prevalent paradigm we discussed earlier is kept intact. Further, these services highlight the problem even further, since they essentially comprise isolated islands of geospatial knowledge. For example, unless one has a Facebook account, he/she cannot search through the available content. The need for a radically different approach on how geospatial content is organized, queried, and presented to mobile users is more than necessary. One has to take account of the special needs of mobile users, the specifications of mobile devices, and the mass of readily available geospatial content and services in the web to provide a persuasive solution. Rather than porting existing paradigms we must figure out new ones that are both intuitive and efficient for the task in hand.

Stemming from these observations, we introduce the principles of task computing as a viable alternative to the problem. Task computing at its simplest, is based on the admission that human beings tackle everyday problems through a simple and naive problemsolving procedure. In order to attain a goal, divide it into smaller, less complex problems, find a solution for each, and therefore accomplish the greater goal. Problems requiring a solution are called tasks, and sub-problems are called sub-tasks. In this respect, we propose the alignment of how geospatial content is structured and queried based on this basic human thought process. Instead of requiring users to perform the translation of their tasks to the digital domain, we convey the digital means closer to the user needs and cognitive skills.

The purpose of this work was to lay the theoretical foundation of task computing in mobile environments and develop a complete framework containing all the necessary programming tools, libraries, APIs, and authoring tools to provide modularity and simple integration with existing solutions. Before continuing, we emphasize that our approach is orthogonal to more traditional tourism applications which recommend sites based on people profiles (with a recommendation system that can be more or less elaborate, see for instance [13]). Additionally, it extends previous works based on the notion of task computing in that it addresses the problem of supporting applications with rich content. In a nutshell, the main focus of our work is the *dynamic discovery, delivery, and presentation of rich and personalized content to mobile users based on the tasks they want to perform.*

The remaining document is organized as follows. Section 2 describes the concepts of task computing. Section 3 presents the principles of our approach and the overall architecture of the implemented infrastructure. Section 4 presents the proof of concept through the TALOS[1] prototype. We conclude in Section 5 with directions for future work.

2 Task Computing

One major reason for the difficulties in searching, finding, and selecting suitable services is that User Interfaces (UIs) are currently designed from the view point of

[1] http://www.talos.cti.gr/

the domain. By using keywords, one has to follow the menu provided, "translate" what he/she wants to do in terms of the menu and, finally, reach the appropriate service [22].

A common sense implied in the previous paragraph is that users organize their everyday lives around solving problems (tasks) and thus both services and content should be structured around tasks in order for them to be easily discovered and assimilated. This idea is strongly encouraged by the enlightening evaluation of NTT DoCoMos task-based approach [20]. It is shown that the percentage of users reached the appropriate services by employing a keyword-type search through their handsets was no greater than 16%, whereas in the existence of a task-oriented search interface the corresponding percentage grew up to about 63%. According to the same test, it is also astonishing that 50% of the latter (one out of two) reached the services within five minutes, compared to just 10% (one out of ten) of the keyword-type search users.

Task computing [16, 14, 15] is a relatively novel concept in regards to the design, implementation, and operation of computing environments, aiming to fill in the gap between tasks (i.e., what users want to do) and services (i.e., functionalities available to the user). In contrast to the current, traditional computing paradigm, task-oriented computing environments are ideal for non-expert users, since they provide access to useful information in a goal-centric manner, requiring minimal user adaptation to the particularities of the user interface and device characteristics. Furthermore, task computing is ideal for pervasive and ubiquitous environments, i.e., computing applications aiming to help users in accomplishing their daily goals. In such environments, users expose high demands for minimal interaction and show limited tolerance to ineffectual content. The organization of content and services around tasks, offered in this discipline, has the potential to greatly improve the computing experience, since users receive timely and accurate information for the exact task in hand.

As already mentioned, although the research in this area is still in its early stages, there are systems and prototype applications which demonstrate persuasive evidence about the importance and benefits of task computing. Besides the system developed by NTT DoCoMo labs, which includes (i) a knowledge base of tasks that a mobile user performs in daily life, and (ii) a database of services that can be used to accomplish these tasks, another important work in this field is the Task Computing Environment (TCE) [23], a service discovery and composition framework for pervasive computing environments. TCE can reduce the cost of performing a task by displaying the tasks that could be performed by executing pervasive services located around the user.

2.1 Ontologies in a Task-oriented Environment

Precondition for providing automated task-based services is the formal description of the potential tasks. This procedure amounts to the construction of a task model or, in other words, a so-called *task ontology*. Up to now, task ontologies have been used in various fields ranging from Artificial Intelligence [28,29] and Expert Systems [19] to Geographical Information Systems [26] and UI Modeling

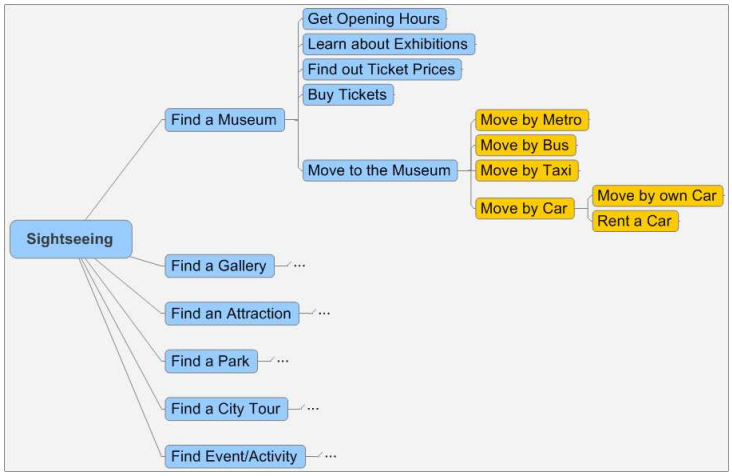

Fig. 1. Task Hierarchy Example

[25]. In general, the reason for their popularity lies in that they provide a flexible way for representing problem solving procedures, mainly because they facilitate sharing and reuse of knowledge along with automated reasoning capabilities [30].

In the context of the Semantic Web [11], the definition of *ontology* could be that it is "a formal specification of a conceptualization". Each ontology of the Semantic Web consists of two parts, (i) a vocabulary (intentional knowledge) that consists of concepts (classes) and relationships (properties or roles), and (ii) additional knowledge (instantiation) consisting of individuals, class and property assertions. A class assertion denotes that a specific individual belongs to a class, while a property assertion assigns a pair of individuals to a specific property. Ontologies in this context are used to provide order for a set of concepts and thus they are also referred to as *domain ontologies*.

Simirarly, task ontology is a term referring to a formal model of tasks. According to the domain of interest, a task may represent a software procedure (e.g., "Sort an array of integers"), a business process (e.g., "Review the proposals") or even a simple human activity (e.g., "Cook food"). In our scenario, each task ontology includes the specification of the task attributes and parameters (e.g., name, input, necessary and/or sufficient conditions for accomplishing a task) and the definition of the relations between different tasks, such as subsumption and temporal ordering.

Intuitively, building a task ontology in our case amounts to modeling what the user of a mobile handset may want to do, e.g. "Go to the Theater", "Visit a Museum" or "Eat at a Restaurant". The basic feature of such an ontology is that complex tasks like those mentioned before are broken into simpler subtasks, as shown in Fig. 1. Here, the hierarchy defined in the task ontology serves as a "task-oriented index" that is used for retrieving the appropriate content while guiding users to perform a task.

As shown in Fig. 2, users tasks are described in the form of task ontologies whereas domain and *context ontologies* contain some of the data needed for instantiating these tasks, i.e., for capturing specific users activities.

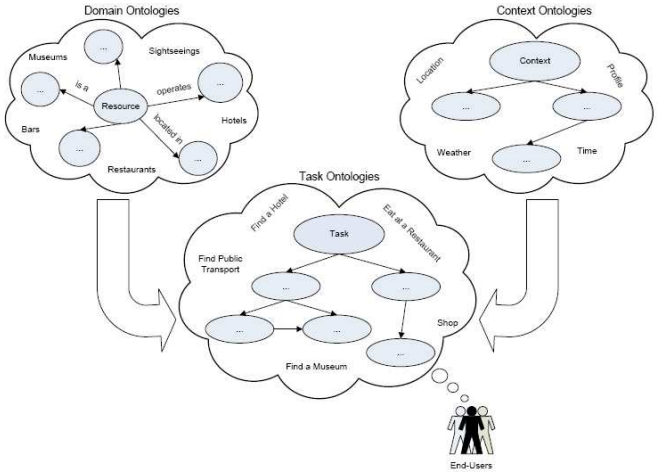

Fig. 2. Ontologies in a task-based service provision system

When defining a task, we may need to refer to one or more domain ontologies for concepts and definitions that are used to describe inputs, outputs, preconditions of the task and so on. Consider for instance the task "Find a Museum". Here, a necessary list of museums in the area can be generated from the instances of a respective Domain Ontology that is stored in a relational database. In exploiting the expressiveness of ontologies, we are able to perform the following:

- **Classification of resources**: Instead of just providing to the user a list of museums in the city that may prohibitively grow huge, especially for a screen of small size, the classification of museums according to their type can be very useful and time-saving when searching for the appropriate one.
- **Reuse of knowledge**: Already described knowledge about solving a task must be accessible for reuse in another task if suitable. Thus, by using ontology-based techniques, we are able to extract parts of solutions related to different tasks and combine them in creating solutions for new tasks. A representative example of this case is the knowledge about transportation as shown in Fig. 1. In this example, the highlighted group of tasks can be reused in more than one levels of the task ontology enabling the authors to easily define multiple "paths" for accessing the same resources depending on the tasks a user selects to perform.
- **Context-aware filtering**: The resources for accomplishing a task alter dynamically according to the current context, e.g. the users location. Thus, by also modeling context, we are able to use conjunctive query answering

techniques (including both context and structured content) for the efficient extraction of the most appropriate resources. A similar approach is followed in [21].

- **Access to various data sources**: By using the W3C standard languages for describing ontologies, i.e. the Resource Description Framework [6] and theWeb Ontology Language [8], we can import and process data from various data sources existing in the web. This can be done either in a forward-chaining manner, where data are processed and stored in the system database offline, or even directly through the mobile device when the user uses the application. A representative example of the second case is the dynamic retrieval of content described in RDF format from DBpedia [2]. One of the first attempts to store and efficiently manage voluminous RDF data in a mobile device with limited capabilities is presented in [27] with very promising results.
- **Consistency check**: When constructing a task ontology one has to pay attention not only to the syntax, but also to the semantics of the latter. The former, i.e. the syntax validation, is ensured by restricting authors to construct the ontology through a graphical interface and automatically interpret the graphical notations into XML. However, regardless the correct syntax, there may be declarations in the ontology that contradict one another (e.g. two tasks each one of them requiring the other to have been performed previously). From our experience, allowing IT-illiterate authors to arbitrarily define their own entities (tasks, relations etc.) will definitely result in various logical contradictions or redundancies. Unfortunately, these kinds of errors cannot be captured through a simple XML Schema validation and thus additional algorithms are required to ensure the consistency of the ontology. These algorithms are based on the firm semantics of the ontology description languages (e.g., OWL).

When faced with a particular type of content one has to be aware that not all content can be modeled using ontologies. For instance, the travel guide content that we use in our prototype is in the form of unstructured text. However, besides the part of the content, such as the POI-related data (e.g., museums, parks, etc.), which can be modeled using ontologies, unstructured text that is stored and retrieved into/from a relational database can still be used in instantiating the tasks of a task ontology. We address this issue in Section 4.3. At this point we emphasize that, in an ontology-based approach, POIs along with their properties (e.g., addresses, operating hours, specific features etc.) are regarded as pieces of well-structured information that is extracted from the overall available (unstructured) content and modeled using ontologies. This kind of POI-related information can be either static (extracted from a book) or dynamic material (retrieved from the Web). Context-related information (e.g., date, time, location, weather, users profile) can be modeled using context ontologies [12] and, similarly to the case of POIs, it can be either static (e.g., the users profile) or dynamic (e.g., the location of the user retrieved on-the-fly).

2.2 Task Modeling

A task reflects what an end-user wants to do in a high-level layer of abstraction, e.g., "Visit a Museum". Each task is accompanied by a set of properties (input, output, precondition etc.) and it is instantiated by context and content in order to become an activity. Following the object-oriented paradigm, a task can be regarded as a class of activities (instances) that share common types of attributes. For example, if we assume that the task "Visit a Museum" has the attributes "Museum" and "Date" defined as inputs, then it can produce activities like "Visit the Museum of Acropolis on 26/6/10" or "Visit the Louvre on 28/7/10" and so forth.

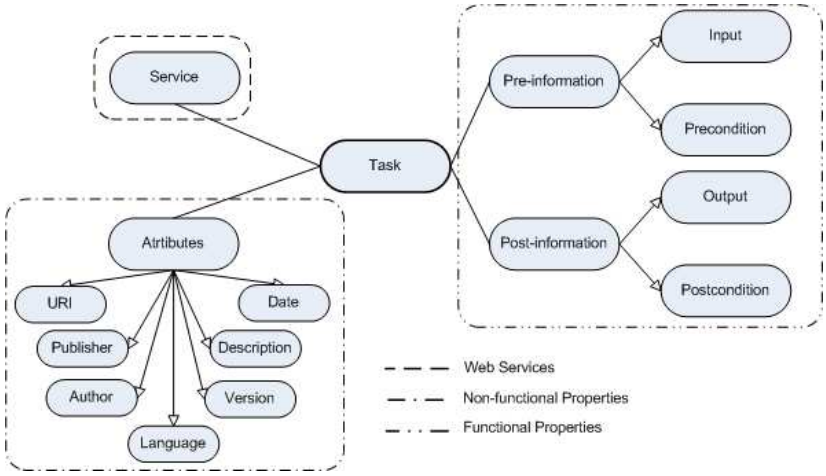

Fig. 3. Task Properties

Definition 1. *A Task T is a collection of well-defined attributes (or properties)* P_i. $T{=}P_i$.

As shown in Fig. 3, each property P_i of a task is classified under one of the following categories: *Non-functional Properties*, *Functional Properties*, and *Services*. The first includes the metadata of a task (name, description, version etc.) that are useful for the task authors. The second class includes all parameters taken into account when a task is performed by the user. This class is divided into two subclasses, *Pre-information* and *Post-information*, that are described in the following. Finally, each task in a task ontology may be realized by one or more web services, e.g. the task "Book a Hotel Room" that is realized by a web service provided at "www.booking.com". In our prototype, the URLs of the respective web services are assigned by the author a priori.

Pre-information includes all parameters needed in order for a task to be executed successfully. This class breaks down into two disjoint subclasses: *Input*

and *Precondition*. An input stands for a parameter needed for performing a task and it can be defined as optional. For instance, the task "Find a Restaurant" may take as alternative inputs the exact location of the user in the form of longitude and latitude, as long as an abstract location in the form of the city or neighbourhood name he/she is located in. In our model, task inputs can be (i) context-related parameters, e.g. the user's location, and (ii) POI-related parameters, e.g. the id of the specified mall in the task "Find Shops in the Mall" as retrieved from the user's local database (cf. Section 3.2). Note that when defining a task the author can specify groups of inputs that must be instantiated all together in order for the task to be performed.

Definition 2. *An input I of a task is a class defined along a hierarchy of concepts that belongs either in a Domain or in a Context Ontology.*

On the other hand, a precondition represents conditions on the task inputs that must hold in order for a task to be performed successfully. Preconditions are logical expressions applying to (i) context, and (ii) POI-related parameters that exist in the user's local database. Regarding the task "Visit an Open Market", a representative example of the first case is "Weather = Sunny". In the second case, the precondition "Hotel.Rank = 5" (with Hotel defined as an input of the respective task) can be used in filtering the hotels retrieved from the user's local database when looking for a luxurious one. In our prototype, such preconditions are specified either a priori by the authors or on-the-fly by the end-user in order to act as an (optional) filter when searching for the most appropriate resources or services. From the UI perspective, they indicate the existence of a screen where the user can specify his/her preferences on the available task inputs and/or POI attributes.

Definition 3. *A precondition P_I of a task defines a restriction in the instantiation of a task input.*

Post-information includes all parameters that are generated after executing the task. It breaks down into two disjoint subclasses: *Output* and *Postcondition*. An output describes information returned after performing a task. Similarly to the case of inputs, task outputs are classified under (i) context-related parameters, e.g. type of weather produced from the task "GetWeather Forecast" and (ii) POI-related parameters, e.g. the name and type of a POI that matches the specified input parameters. The difference with respect to inputs is that a task can produce as output a piece of content in the form of unstructured text that is retrieved either from the travel guide (static) or from the web (dynamic). Note that an output of a task (except the unstructured content) may be used as input in other tasks of the ontology introducing a dataflow. In our prototype, the default output of a task is the unstructured content retrieved from the travel guide (if available).

Definition 4. *An output O of a task is a piece of unstructured text or a class defined along a hierarchy of concepts that belongs either in a Domain or in a Context Ontology.*

A postcondition represents conditions that must hold after performing a task. Similarly to preconditions, postconditions are logical expressions applying to (i) context and (ii) POI-related parameters both defined as outputs of a task. An example of a postcondition regarding the task "Move to the Park" could be something like "User.Location = Park.Location" where the "User.Location" is a context-related parameter referring to the user's position (defined as output of the task) while "Park.Location" is the location of the POI, i.e. the specified park, also defined as an output of the task. From the mobile application perspective, such a postcondition is useful for recommending tasks to the user and also for helping the user (re)organize the schedule of tasks to perform during the trip.

Definition 5. *A postcondition PO of a task defines a restriction in the instantiation of a task output.*

Before continuing, we point out that each input and output of a task in our prototype is instantiated by one of the following modules: (i) the *Context Aggregator* that manages all context-related attributes, (ii) the user (through the UI), and (iii) the users local database.

As far as task relations are concerned, in our task modeling approach we define four kinds of relationships between tasks: (i) *SubTaskOf*, (ii) *OR*, (iii) *CHOICE*, and (iv) *Sequence* relation. The SubTaskOf relation introduces a task hierarchy and denotes that the parent task is accomplished by accomplishing all of its children in any order. The OR and CHOICE relations introduce a task hierarchy as well, but in the former case the parent task is accomplished by at least one of its children, while in the latter case the parent task is accomplished by exactly one of its children (exclusive option). Finally, a Sequence relation denotes that a task is performed always before another one and it is accompanied by at least one parameter passing (binding) from the first task to the second one. For instance, the tasks "Find a Hotel" and "Learn about Hotel Facilities" are representative examples of this case considering that the second task always needs as input the output of the first one, i.e. the specified hotel. From the application perspective, different types of relationships between tasks in the ontology layer define different functional properties of the user interface (display order, redirection etc.).

3 Background and Architecture

This section presents the modes of interaction with the system, the basic concepts, as well as the overall architecture of TALOS. For the ease of presentation, in the remaining document we address the use case of a (mobile) traveler but the overall approach can be easily applied to other scenarios as well where task computing plays a central role.

3.1 Modes of Interaction

Our task-based service provision system supports the following two modes of interaction between the end-user and the services provided:

– **Planning mode**: The user selects a number of available tasks from those existing (pre-defined by the authors) in the task knowledge base of the system and incorporates them in a schedule. In order to produce a schedule, the user has to provide (i) the order of the tasks he wants to perform during the trip and (ii) the duration of each task (start and end time). This procedure results in a number of planned activities, each of which stands for a single step of the users schedule. Besides the previous basic requirements, the user can also define context-related prerequisites (preconditions) for each step (e.g., preferred weather condition). When the user finishes with the plan, the tasks are instantiated and the prerequisites are automatically evaluated by the system (for instance, if a weather prerequisite has been specified, weather forecasts are used) in order to identify conflicts and help the user re-organize the schedule.
– **Trip mode**: The user has already made the schedule of the trip, which is downloaded on the mobile device, and now clicks on a planned activity while being on site. In order to perform the specified activity (e.g., visit the open market), the respective task is re-instantiated which means that the required context-related inputs and prerequisites are reevaluated, either in a *push* or in a *pull* mode, using the current context that is retrieved on-the-fly. If for example, in this case, the current weather conflicts with the precondition defined in the schedule (a priori by the user), then a new rescheduling process takes place where the system suggests a number of alternative tasks after informing the user about the conflict. The recommendation of alternative tasks is done according to (i) the users schedule, and (ii) the current context (location, time, weather, users profile etc.).

The creation of the schedule is done through a *Content Portal* that is described in Section 3.3. Regarding the Trip mode, we point out that the user is not restricted only in performing the tasks he/she (optionally) specified during the Planning mode. Instead, taking into account that the mobile application also provides a general hierarchy of tasks, the user can select additional tasks to perform (following this hierarchy) without having them added in a schedule. In any case, the instantiation of a task, the recommendation and rescheduling processes, and the retrieval of the appropriate content and services while the user is on site is based on the principles we describe in the following.

3.2 Basic Concepts

The overall mechanism relies on concepts coming from mobile applications but also from Event-based System (EBS) architectures, which are designed to provide a quick reaction when new information is coming to the system.

The context-related parameters required for instantiating a task (date, time, location, weather, users profile) are managed using the notion of the *situation*. In simple words, a user's situation consists of a set of context-related parameters which are valid during a certain time. As described in [17], a situation is a multidimensional concept defined as a triple (C, t_1, t_2) with C a collection of

(attribute, value) pairs and t_1 and t_2 the beginning and the end of the interval on which the collection C is valid. Each attribute of a pair stands for a class defined in a context ontology.

Following the *publish-subscribe* paradigm as often used in event-based systems, users subscribe to information stored in different sources (e.g., in a database of museums). These sources publish new information when there is a change (e.g., new opening hours). In a mobile, task-oriented application like ours, the need for new information occurs when there is a new location or a new task to consider. In general, an event is usually understood as "a happening of interest". In our case, an event is a a new task or a new (significant) location. This is detected on a pull mode but also on a push mode.

As far as *re-scheduling* is concerned, such a need occurs when something has changed either on the source side or on the user side (e.g., new location). On the source side, changes can occur on static sources when there is an update (e.g., new opening hours) or on dynamic sources, for instance, regarding weather prognosis that may affect the planned schedule. Note that if there is a change of plan, the system could notify the user of the exact changes as it is done in [18] when there is a discrepancy between what the user expects and what the system can now offer. However, these notification issues are not the focus of our current approach.

Finally, regarding the *recommendation* of tasks and content, the steps to be taken are depicted in the following:

1. *situation* ← get-user-situation()
   (situation includes location, date, time of day, season, weather, or profile)
2. *task-list* ← recommend-tasks(*situation*)
3. *task-selected* ← choose-task(*task-list*)
4. *content-list* ← recommend-content(*task-selected*; *situation*)
5. show-content(*content-list*)

3.3 Architecture

The architecture of the overall system is illustrated in Fig. 4. The system has a typical three-tier architecture composed of (i) the *Data Sources* tier where the task ontologies and the available content are stored, (ii) the *Services* tier that includes all services implementing the business logic, and (iii) various *Clients*. In addition, two kinds of tools, the Task Ontology Authoring Tool (cf. Section 4.2) and the Content Authoring Tool (cf. Section 4.3), are provided to serve as a means to import, edit, and update the data sources.

A *Task Author* (TA) is responsible for creating the task ontologies based on the model we described in Section 2.2. By the time a task ontology is created and uploaded on the server, it is regarded as an *Idle Task Ontology* (ITO). This means that the ontology cannot be downloaded and used by the end-users until content is assigned to its tasks. The latter is performed by the *Content Author* (CA). By the time a CA assigns content to the tasks of an ITO, then the task

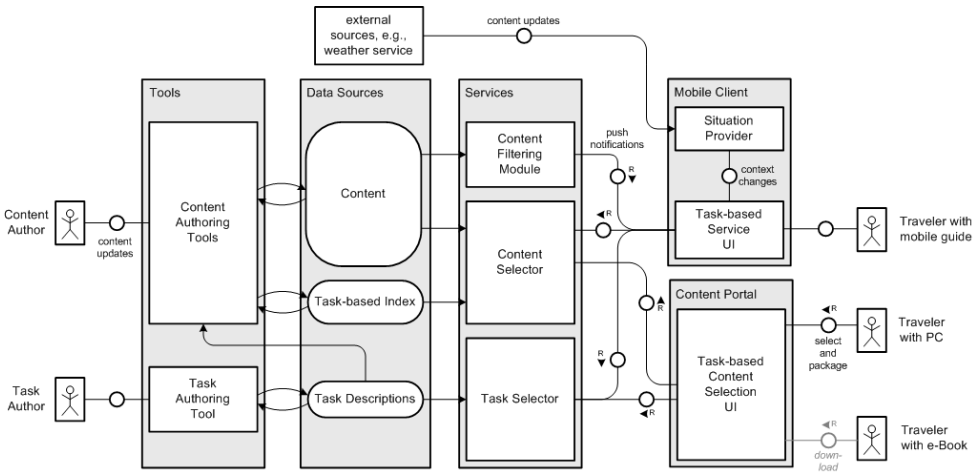

Fig. 4. System Architecture

ontology becomes an *Operational Task Ontology* (OTO) which means that it can now be downloaded (along with the assigned content) and used by the end-users. Obviously, besides the assignment of (structured and unstructured) content to tasks, the CA is also responsible for the overall management of content (import, edit, update, geocode etc.).

The *Data Sources* tier encompasses (i) the *Task Knowledge Base* (TKB) and (ii) the *Content Base* (CB). TKB keeps all versions of every task ontology. The main reason we keep all versions of task ontologies in TKB is because, when downloading an OTO, the user of the mobile device may decide to include only a part of the assigned content and not all of it. In such a case, the rest of the content can be downloaded gradually during the use of the application. Thus, a downloaded OTO cannot be overwritten in the server until all end-users have updated their local copies; otherwise, in case the downloaded OTO is out of date, incompatibility problems may occur. Apart from this, a version history of the authors work can be very useful when, for some reason, one needs to rollback to an older version.

All available content is stored in the *Content Base*. More specifically, CB includes (i) unstructured content in the form of text, (ii) geo-referenced (structured) content in the form of POIs and their attributes, and (iii) the ids of the (operational) tasks to which the content is assigned (*Task-based Index*). As mentioned in the previous sections, POI-related data are described using an ontological model, hence, they are stored in CB following a variation of the widely-adopted database representations presented in [24]. As shown in Fig. 4, the dynamic content retrieved from external sources (e.g., web pages) is accessed by clients directly.

The *Services* tier encompasses components for recommending (i) tasks with respect to a users situation (*Task Selector*), and (ii) content from the underlying

Content Base using a task selected by the user along with the users situation as parameters (*Content Selector*). Task and content selectors are used in planning mode (e.g., recommend content targeted to a backpack traveller or content related to events that take place during the trip), but also in trip mode when an automatic re-scheduling is needed. However, the recommendation of tasks can also be done offline by using the context-related task parameters defined in the operational task ontology. For example, in case the task author has specified a weather precondition for the task "Have a Boat Tour", then this precondition will be evaluated using the current situation of the user and in case there is a weather conflict, the task will be excluded from the recommended ones. At this point we emphasize that, although the task and content selectors support the use of content in a pull mode, a *Content Filtering Module* is used for notifying the user about just-in- time content updates in the server (push mode).

Regarding the *Clients* tier, there are two different clients that use the services of TALOS to provide task-based content delivery to the user. First, a *Content Portal* facilitates the pre-selection of content based on tasks planned by the user. In this case, the content can be compiled into a personalized travel guide eBook augmented with (i) additional information from the web, and (ii) a task-based index. Second, the content can also be compiled into a database (e.g., an SQLite file) that is downloaded and used as an offline source of content in the mobile device. In the latter case, the users local database contains a (personalized) portion of the content stored in the *Content Base* of the TALOS server that is coupled with the tasks of the downloaded task ontology. To access this information in a task-based and situation-dependent manner while traveling, besides the actual content and the task-based index, a situation provider component (the *Context Aggregator*) is included in the application. The situation provider uses various techniques to collect and aggregate context data or even anticipate user situations. Context changes are forwarded to the client logic that calls the services and adapts the task and content recommendations accordingly.

4 The TALOS Prototype

This section describes the TALOS prototype system which includes a number of authoring tools and services as well as a mobile travel guide for the Apple iPhone. The core objective in developing the prototype was not only to facilitate the end-user interaction through an intuitive and easy-to-use interface, but also to make task and content authoring as easy as possible.

4.1 General Information

The mobile travel guide provides a task-based interface where a mobile user can select tasks from a predefined task hierarchy and access relevant content. The *Context Aggregator* component gathers all data describing the users situation, i.e., the users current location, time, weather, and the users traveler profile. Depending on the users interaction with the UI, and the current context, content

and tasks are recommended to the user, allowing for personalized information provision. Apart from accessing information, the user can also use the planning functionality supported by the mobile guide in order to create a trip schedule.

Available content involves static content that consists of an existing travel guide for Brussels and a piece of geo-referenced content created by Michael Mller Verlag[2], as well as dynamic information from the Web. Dynamic information includes (i) map tiles provided by the OpenStreetMap project, (ii) scraped information from various web sites concerning POIs, and (iii) information obtained by different web services, e.g., the Yahoo [9] and Geonames [3] weather services. Data storage on the mobile travel guide is handled by an SQLite database [7]. A combination of the Apple Core Location Framework and Wireless Positioning Techniques [10] are utilized in order to capture the users location (indoors and outdoors) and to offer context-aware content. The prototype is developed as a stand-alone application for the iPhone platform, hence, objective-C and the iPhone SDK provided by Apple are used.

4.2 Task Authoring

The creation of task ontologies is done within the Task Ontology Authoring Tool (TOAT). TOAT provides an interactive 2D canvas where the authors can define the tasks of the ontology along with the relations among them by simply dragging elements from a palette. The result of this procedure is a Directed Acyclic Graph (DAG) where each node represents a task, and each edge represents a relation between two tasks. For ease of presentation, the underlying task parameters are not visualized in 2D, otherwise the graph would be quite difficult to handle. Task properties are defined form within a dynamic form shown in the right part of Fig. 5.

After finishing the authoring procedure, the author can get the XML file of the task ontology by clicking on the respective option in the menu. Before producing the XML representation of the ontology, TOAT performs a number of validations to ensure (i) that the file is valid according to a predefined XML Schema and (ii) that the ontology does not contain contradictions or redundancies. An example of the first case is when the author has defined that a task is optionally accomplished by only one task, i.e. there is a single OR relation between the parent and its children. In the latter case, a contradiction could be a subTaskOf cycle where two tasks are defined to be both the child and parent of one another. Finally, redundancies in the ontology are introduced when the user defines the same thing more than once, as for example two identical sequence relations between the same tasks.

The XML file generated by the editor is stored in the central task knowledge base. This XML file defines the structure and the basic functionality (inputs, preconditions, dataflow etc.) of the task-based UI. In this sense, it serves as an abstract model of the hierarchical user interface. The approach we follow here is reminiscent of the well-known Model-View-Controller (MVC) architecture

[2] http://www.michael-mueller-verlag.de/

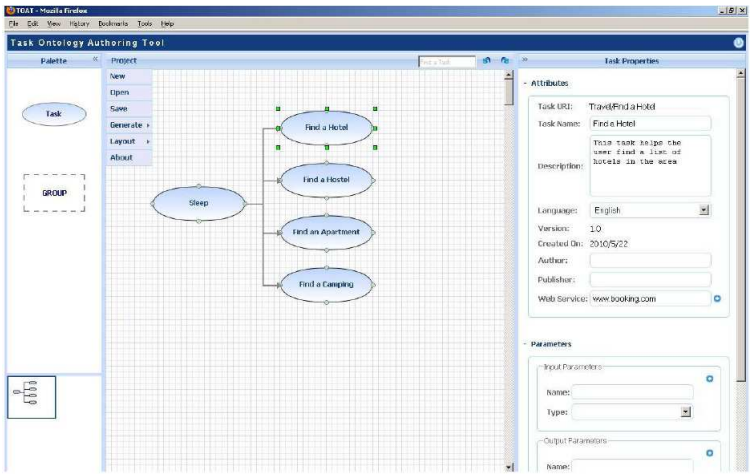

Fig. 5. Task Ontology Authoring Tool

paradigm where the view (what the user interacts with) is based on a generic model which changes (through the controller) according to (i) the users actions, (ii) the current context, and (iii) the available content. Note that having a generic model as the basis from which the view is automatically generated provides us with great flexibility in managing the UIs, because the users application can be easily updated by just changing the respective graph representations in TOAT.

In order to avoid software installation, TOAT is developed as a web browser application using open JavaScript frameworks. The created task ontologies are stored using AJAX-based techniques in the central task knowledge base of TALOS. Based on the fact that we deal with a real-world environment where many authors may work on the same task ontology, we have also implemented a simple versioning mechanism that ensures the consistency of the model in case an update or a rollback to an older version is made.

4.3 Content Authoring

A core objective in the current effort was to make existing rich location-relevant content, in our case travel guides, readily accessible through mobile device and task-based interfaces. To do so we (i) linked such content to task metadata (task annotation), (ii) geocoded the content to provide a map-based access, and (iii) added dynamic Web content to the mix by using Web scraping, i.e., linking content from third-party sources to our content (e.g., linking the current exhibitions of a museum from the respective Web page to the POI information stored in our content management system). The above task require a content management system, which in our case comprises a database (SQL Server 2008) and an interface. Since content authoring is to be conducted by writers, the annotation tool had to be easy to use. To also avoid software installation task, the

tool was implemented as a Web browser application, specifically using the Ruby on Rails open source web application framework and JavaScript to implement the client functionality. The following sections will give more details on the content authoring tasks mentioned above.

Geocoding To geocode the content, we relied on existing Web services such as the Google Maps and Yahoo Maps API, Geonames, and Open- StreetMap namefinder and developed an application wrapper that provides uniform access to any or all services depending on the users needs and licensing restrictions. Advocating a semi-automatic geocoding approach, a map-based interface is introduced that allows the user to update the automatic geocoding results by dragging markers on the map. Fig. 6 shows the map interface.

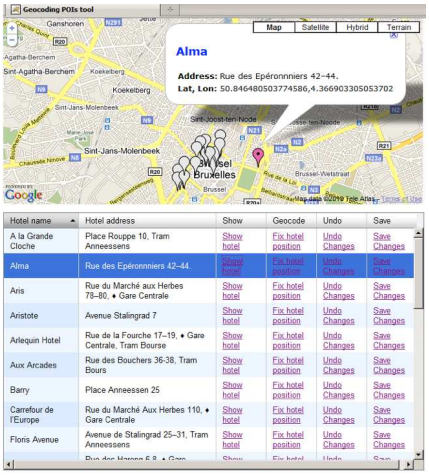

Fig. 6. Geocoding Interface

Task Metadata Using a task ontology as input, the content annotation tool provides a simple means of linking tasks to content. The content as stored in the content management system is shown in the Web interface along with the task ontology represented in a tree structure. Clicking on a section highlights the section and shows a new pop up window with the suggested task hierarchy as a tree view and a list of already linked tasks with the specific section. After selecting a portion of the content, tasks are linked by selection to the selected content. Multiple selection is supported and the linked tasks are visualized accordingly. Fig. 7 shows the interface.

Dynamic Web Content When considering electronic versions of print content, an important property is the possibility of frequent (and cheap) updates.

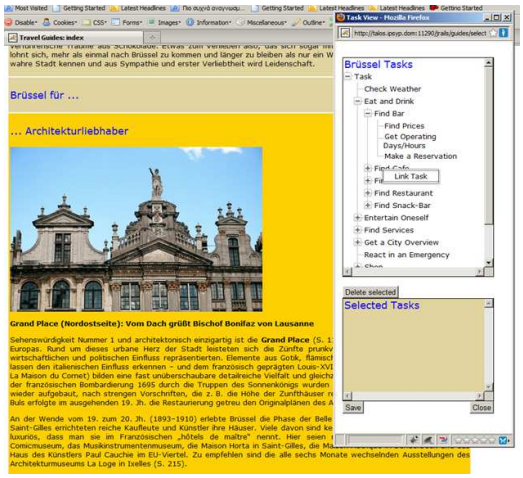

Fig. 7. Task Annotation Interface

To streamline publishing such content, we link third-party Web content to our content base. Examples, here are opening hours and changing exhibitions of museums. Once such a link is established, it is simple to check if the information at the remote site has changed. We have developed Web scraping tools, specifically a Firefox Web browser extension that allows one to mark content at a remote site and so to link dynamic Web content to authored content. Fig. 8 showcases the tool with its Firefox extension.

4.4 iPhone Interface

The user interface of the mobile travel guide includes four different modes, namely the (a) *Activities*, (b) *eBook*, (c) *Map*, and (d) *Diary* mode as shown in Fig. 9. We point out that all these modes are interlinked to one another. Each one of them provides a different way to access available information and thus it presents another dimension of it.

The activities mode shows either the predefined task hierarchy or a context-adapted one. It offers a task-based UI which is generated dynamically, as defined in the task ontology. A task selection leads the user to appropriate content which can be a piece of unstructured text or a list of POIs. The map mode offers a spatial view consisting of a full screen map showing the (geocoded) POIs. Selecting a POI reveals content and task recommendations.

The eBook mode is the content view, where unstructured content, such as text derived from existing travel guides, and structured content, such as POI metadata can be read. For each unit of content (which could be a section of the guide or a POI) relevant tasks and POIs are suggested. Finally, the diary view offers a temporal view, where the user's plans and memories are presented. The user selects tasks in order to plan his/her activities and stores bookmarks of all

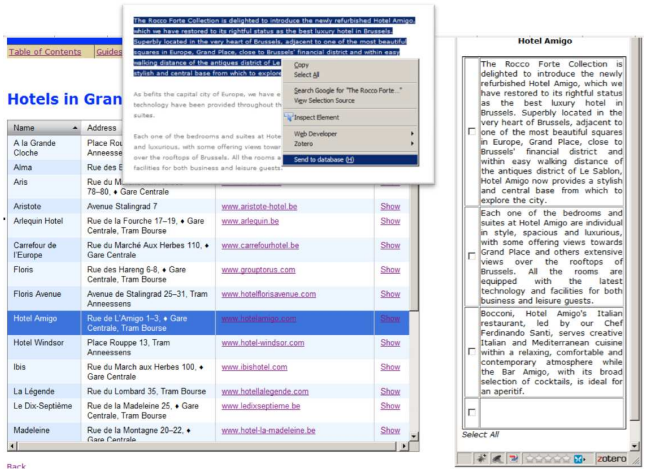

Fig. 8. Web Scraping Interface (plugin and web interface)

kinds of available content or personal pictures and notes to create trip memories, creating in this way a personal trip diary.

5 Conclusion

In this work, we investigated the potentials of dynamic discovery, delivery and presentation of rich content to mobile users based on the tasks they want to perform, an approach that leads to Mobile Task Computing. Starting from the theoretical foundation of the task computing paradigm in mobile environments, the contributions of this work also include a complete framework containing all the necessary programming tools, libraries, APIs, and authoring tools to provide modularity and simple integration with the existing solutions.

The core feature of our approach is an intuitive task model that can be used to describe the various activities of the end-users. All context-related parameters required for the dynamic discovery and personalization of the available content are integrated with this task model by exploiting the flexibility of ontological engineering. The first results from the evaluation of the prototype iPhone application (developed in the context of the TALOS project) clearly demonstrate the advantages of our approach.

Our future research directions in the field focus on collaborative task computing environments where users can share or recommend tasks to others and also on the combination of task and cloud computing techniques for retrieving resources and services available on the cloud by utilizing task-related knowledge.

Acknowledgments. This work was partially supported by the TALOS project, funded by the FP7 Research for SMEs work programme of the European Com-

Fig. 9. iPhone screenshots. (a) Activities Mode: the task-based UI allowing for content provision and planning (b) EBook Mode: the content view offering structured and unstructured content (c) Map Mode: the spatial view showing POIs on a full-screen map (d) Diary Mode: the spatial view showing users plans and memories

mission under contract number 222292. We would like to thank all partners in the TALOS project for their significant contributions in realizing this work.

References

1. Bing mobile. `http://www.discoverbing.com/mobile/`. Last accessed July 2010.
2. Dbpedia. `http://dbpedia.org/About`. Last accessed July 2010.
3. Geonames. `http://www.geonames.org/`. Last accessed July 2010.
4. Google mobile. `http://www.google.com/mobile/`. Last accessed July 2010.
5. Ovi maps. `http://betalabs.nokia.com/apps/ovi-maps-beta-for-mobile`. Last accessed July 2010.
6. Resource description framework. `http://www.w3.org/TR/REC-rdf-syntax/`. Last accessed July 2010.
7. Sqlite. `http://www.sqlite.org/`. Last accessed July 2010.
8. Web ontology language. `http://www.w3.org/TR/owl2-overview/`. Last accessed July 2010.
9. Yahoo weather. `http://weather.yahoo.com/`. Last accessed July 2010.
10. S. Athanasiou, P. Georgantas, G. Gerakakis, and D. Pfoser. Utilizing wireless positioning as a tracking data source. In Proc. of the 11th International Symposium on Advances in Spatial and Temporal Databases (SSTD), 2009.
11. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, 284(5):3443, 2001.
12. A. Bikakis, T. Patkos, G. Antonis, and D. Plexousakis. A survey of semantics-based approaches for context reasoning in ambient intelligence. In Proc. of the Artificial Intelligence Methods for Ambient Intelligence Workshop at the European Conference on Ambient Intelligence, 2007.
13. A. Hinze and A. Voisard. Location- and time-based information delivery in tourism. In Advances in Spatio-temporal Databases, volume 2750 of LNCS, Berlin/Heidelberg/New York, 2003. Springer Verlag.

14. R. Masuoka, Y. Labrou, and Z. Song. Semantic web and ubiquitous computing - task computing as an example. In AIS SIGSEMIS Bulletin, pages 2124, 2004.
15. R. Masuoka, B. Parsia, and Y. Labrou. Task computing - the semantic web meets pervasive computing. In Proc. International Semantic Web Conference (ISWC), 2003.
16. R. Masuoka and M. Yuhara, editors. Task Computing - Filling the Gap between Tasks and Services. FUJITSU, 2004.
17. U. Meissen, S. Pfennigschmidt, A. Voisard, and T. Wahnfried. Context- and situation-awareness in information logistics. In W. Lindner, M. Mesiti, C. Torker, Y. Tzikzikas, and A. Vakali, editors, EDBT Workshops, volume 3268 of LNCS, pages 335344, Berlin/Heidelberg/New York, 2004. Springer Verlag.
18. U. Meissen, S. Pfennigschmidt, A. Voisard, and T. Wahnfried. Resolving knowledge discrepancies in situation-aware systems. International Journal of Pervasive Computing and Communication (JPCC), 1(4):327336, Dec. 2005.
19. R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda. Task ontology for reuse of problem solving knowledge. In Proc. International. Conference on Building and Sharing Very Large-Scale Knowledge Bases, pages 4659, 1995.
20. T. Naganuma and S. Kurakake. Task knowledge based retrieval for service relevant to mobile users activity. In Proc. of the International Semantic Web Conference (ISWC), 2005.
21. T. Naganuma, M. Luther, M. Wagner, A. Tomioka, K. Fujii, Y. Fukazawa, and S. Kurakake. Task-oriented mobile service recommendation enhanced by a situational reasoning engine. In Proc. of the Asian Semantic Web Conference (ASWC), 2006.
22. M. Sasajima, Y. Kitamura, T. Naganuma, K. Fujii, S. Kurakake, and R. Mizoguchi. Task ontology-based modeling framework for navigation of mobile internet services. In Proc. of the Third IASTED European Conference on Internet and Multimedia Systems and Applications, 2007.
23. Z. Song, Y. Labrou, and R. Masuoka. Dynamic service discovery and management in task computing. In Proc. of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004.
24. Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representation of rdf/s stores. In Proc. of the International Semantic Web Conference (ISWC), 2005.
25. M. Van Welie. Task-Based User Interface Design. PhD thesis, Vrije Universiteit, Amsterdam, 2001.
26. S. von Hunolstein and A. Zipf. Towards task oriented map-based mobile guides. In Proc. of the International Workshop HCI in Mobile Guides, (Mobile HCI), LNCS. Springer Verlag, 2003.
27. C. Weiss, A. Bernstein, and S. Boccuzzo. i-moco: Mobile conference guide - storing and querying huge amounts of semantic web data on the iphone/ipod touch. In Billion Triples Challenge of the International Semantic Web Conference (ISWC), 2008.
28. B. Chandrasekaran, J. R. Josephson, The Ontology of Tasks and Methods. In Proc. of the AAAI Conference on Artificial Intelligence, 1997.
29. D. Rajpathak, E. Motta, and R. Roy. A Generic Task Ontology for Scheduling Applications. In Proc. of the International Conference on Artificial Intelligence (IC-AI), 2001.
30. A. Gómez-Pérez, and V. R. Benjamins, Applications of Ontologies and Problem-Solving Methods. In AI Magazine, Vol. 20 No 1, 1999.