Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 34 (2008) 1318–1332
Expert Systems with Applications
www.elsevier.com/locate/eswa

SAMAP: An user-oriented adaptive system for planning tourist visits

Luis Castillo [a,*], Eva Armengol [b], Eva Onaindía [c], Laura Sebastiá [c], Jesús González-Boticario [d], Antonio Rodríguez [d], Susana Fernández [e], Juan D. Arias [e], Daniel Borrajo [e]

[a] *Departamento de Ciencias de la Computación e I.A., ETSI Informática y Telecomunicaciones, Universidad de Granada, 18071 Granada, Spain*
[b] *Artificial Intelligence Research Institute (IIIA), Spanish Research Council (CSIC), Campus Universitat Autònoma de Barcelona, 08193 Bellaterra, Catalonia, Spain*
[c] *Departamento de Sistemas Informaticos y Computación, Universidad Politecnica de Valencia, Camino de Vera, s/n, 46022 Valencia, Spain*
[d] *aDeNu Research Group, Dpto. Inteligencia Artificial, ETSI Informática – UNED, Juan del Rosal, 16-3[a], E-28040 Madrid, Spain*
[e] *Departamento de Informática, Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911-Leganés, Madrid, Spain*

Abstract

In this paper, we present SAMAP, whose goal is to build a software tool to help different people visit different cities. This tool integrates modules that dynamically capture user models, determine lists of activities that can provide more utility to a user given the past experience of the system with similar users, and generates plans that can be executed by the user. This system is intended to work in portable devices (mobile phones, PDAs, etc.,) with internet connection. In this paper, we describe the architecture, the knowledge model that is shared among components using an ontology, and the three components of the tool: user module, case-based module and planning module.

Keywords: Planning; Machine learning; Case-based reasoning; User modelling

1. Introduction

Electronic tourism is one of the activities that have enjoyed of an important success in the Internet, not only from the commercial point of view but also from a social perspective. Many sites provide information about hotels, plane tickets, etc. There are also recommender systems that tell us which destination is more suitable according to our preferences (Delgado & Davidson, 2002; Fesenmaier, Ricci, Schaumlechner, Wöber, & Zanellai, 2003). Once a destination has been selected, we can find many sites which give us information about places to visit in a city, activities to do during the travel, restaurants, etc. But this information is static in most cases; that is, it is presented to all users in the same way. Also, the quantity of available information can be large and, therefore, the user must select manually which pieces of information are interesting for him/her. Finally, they do not automatically provide plans and schedules, according to user needs and sites schedules. So, from the user point of view, it is useful to have a recommender system that tells him/her which places may be interesting to visit in a certain city taking into account his/her profile, computing a tourist daily plan, with indications about which places to visit in the given timeframe, and also how to go from one place to another.

We have developed a system, SAMAP, which provides this functionality. It captures and updates a user model about different city visits dynamically, analyses past planning behavior of the user and similar users in the same type of visit, and selects a list of places that have a high probability to be interesting for the user through a case-based reasoning (CBR) approach. Then, taking into account distances, places timetables, etc., it computes a plan, and also shows how to go from one place to the next in the plan. In order to store all the information the system needs, we have defined an ontology.

* Corresponding author. Tel.: +34 958 240803; fax: +34 958 243317.
E-mail address: L.Castillo@decsai.ugr.es (L. Castillo).

0957-4174/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2006.12.029

Nowadays, the use of ontologies in the internet is increasing, specially motivated by the Semantic Web efforts. Ontologies are being also used when building multiagent systems, in order to share their common knowledge in a declarative way. SAMAP has been built as a multiagent system, consisting of three main agents: user modelling and interface agent, case-based agent, and planning agent. In Heflin and Muñoz-Avila (2002), the authors present another example of this kind of system that integrates HTN planning and Semantic Web ontologies for defining agents capable of solving complex information integration tasks. Other related work are the trip planning systems for booking flights or hotels, renting cars, etc. though most of this work does not report in-city planning/scheduling and/or the use of ontologies. They focus on data integration rather on scheduling (Ambite et al., 2002; Knoblock & Minton, 1998; Camacho, Aler, Borrajo, & Molina, 2005). There are other proposals of tourism ontologies, like for example Harmonise.[1] The goal of Harmonise is that participating tourism organisations keep their proprietary data format while cooperating with each other. Harmonise focuses on obtaining the interoperability between different organizations with different standards so that they can share information with others. In contrast, SAMAP does not solve the problem of travelling to a specific place. Therefore, it does not need to know anything about flights, travel agencies or other elements related to travelling from a city to another one. SAMAP focuses on facilitating the activities that a tourist can perform in a city, also considering the transport means that can be used to move within the city that is visited.

Section 2 introduces the SAMAP architecture. Then, Section 3 describes the ontology defined in SAMAP. Sections 4–6 present each of the three modules it consists of. We finish with some Section 7.

2. SAMAP

The goal of SAMAP is to compute a tourist plan for a user with Internet access via a ubiquitous device, such as a PDA or a mobile phone. This system needs to have access to related information, represented in an ontology:

- Information about the city, that is, the context that surrounds the visit (i.e. monuments and interesting places in a city, transport means, etc.).
- Personal data, interests and preferences of the user, that is, type of activities that this user likes to do when s/he visits other cities (user model).
- Places that other people similar to our current user liked when they visited the same city (plans of other users).
- Basic services that this user prefers when performing his/her activities (i.e. payment with credit card, use only taxi for movements, etc.).

Once we have all this information, SAMAP computes a plan that contains the following elements:

- A selection of the most interesting places for this user according to his/her model.
- Indications about which transportation means s/he should take to move between different places including walking.
- Recommendations about where to have lunch or dinner (restaurants, bars, etc.).
- Proposals of places of leisure such as cinemas or theatres.

Fig. 1 shows a high level view of the architecture of SAMAP. The first step consists of building the user model. This requires the user to enter information about him/herself, that is, personal data, interests and preferences about, i.e., art, monuments, meals, etc. Also, the user should specify which city is s/he is going to visit, under which schedule, etc. This information can be gathered by using any device with Internet connection. In order to obtain more interesting data about the user, the system (by means of machine learning techniques) can use past information about the same user (provided s/he has used the application before). This information is stored in the ontology.

The second step consists of the generation of a list of activities that s/he might like to perform in the current visit according to the preferences. As explained before, the activities might come directly from the user, or automatically generated by CBR from similar users plans in the city or similar cities. Each activity will also be described with the expected utility that performing this activity might have for the user. This utility can be directly specified by the user, or computed by SAMAP from knowledge about similar users.

The last step is the computation of the tourist plan by taking into account the previously computed list of activities. Another solution might have been to use a CBR mechanism throughout the system. We prefer to use a planning approach with "classical" operators because (a) there is not a CBR planning tool that allows to develop applications quickly and (b) in big cities where there are a number of different goals and ways to go from one place to another

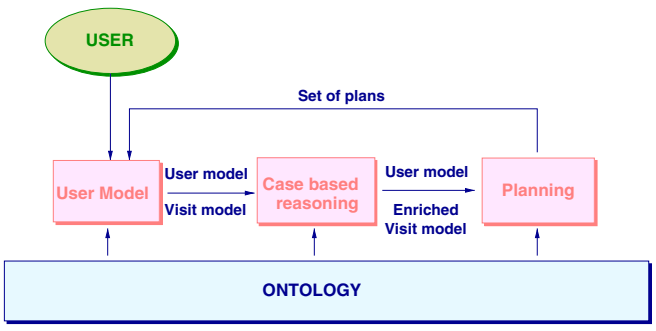

Fig. 1. High level view of the architecture of SAMAP.

[1] www.harmonise.org.

and scheduling the visits, we would need a very rich adaptation agent. Therefore, we think we have adopted a reasonable solution: the planner takes this information as input together with information about the city (streets and intersections, situation of each place, etc.) to compute the plan for the user.

3. SAMAP ontology

The system relies on an ontology that is shared by all agents, where each agent uses part of it. This ontology is the cornerstone of knowledge representation and exchange in the whole architecture, as it is shown in Fig. 1. Its main role is twofold. On the one hand, it is the basis of knowledge exchange among all agents in the architecture. As part of the CBR agent, it stores past visits and relates them to the user model that generated them. It also stores the information of user profiles generated by the user agent and this information will be used later by the CBR agent to retrieve past cases and to suggest the user a list of possible visits and their relevance according to his/her profile. It also contains both the relevant touristic information of the city that will be translated into the initial state of the planning agent and the set of plans found to satisfy the given goals. All this knowledge is given to the agents, which are based on different AI technologies, in a common language (CLIPS). Since it is a language that can easily be understood, and that maintains all the features of the ontology, it can later be translated into the specific languages of every agent like PDL or PDDL languages for the planning agent or the NOOS language for the CBR agent (Plaza, Arcos, & Martín, 1997).

On the other hand, the ontology allows an easy maintenance of the existing data by means of the standard PROTÉGÉ environment. Also, wrappers to web services or plain HTML web pages can be built that store the information in the ontology. Then, on one side, the development team can edit, validate and modify what is the common knowledge repository of the whole system. This is a critical issue to keep the knowledge updated about the city, like available visits, schedules and their features with a common language for all the agents. On the other side, automatic tools that access the information stored in the web, in the form of web pages or services, can easily maintain some knowledge chunks related to restaurants, cinemas, or shows.

The main classes of SAMAP ontology and their relationship appear in Fig. 2.[2]

In order to define this ontology we have used PROTÉGÉ which is an ontology editor developed at Stanford University.[3] This editor helps on the creation of ontologies and exports the information about created ontologies in many different ontology formats, such as CLIPS, which is the format used in SAMAP because its syntax is quite similar to LISP (we have developed all modules in LISP) and creating translators using rules is simple. However, other languages, such as OWL (Ontology Web Language), which is very powerful and considered now one of the standards in Web services, could be used in the future as PROTÉGÉ has a set of extensions or plugins that allow to load, edit and save ontologies in this format.

The SAMAP ontology consists on three main classes: user, activities and city information. The *User class* has attributes about the personal information of the user like name, profession, mobility, language, sex, etc., but it also references other classes where other user information is considered (see Fig. 2). These attributes as well as the captured user model are used for determining which activities the system should recommend to the user. For example, if SAMAP knows that a user has a reduced mobility (s/he needs a wheelchair), then the planner will not offer plans with places that are not prepared to receive visitors in a wheelchair. The same applies to other attributes. The user model with his/her preferences are also used for proposing the user plans with activities that were performed by people similar to the user on that same city.

The User class is related to the *User-Context class*. This class has context-dependent information, such as user localization, if the user has a car on that visited city, money available, residence, etc. The user localization is useful for planning the itinerary whereas knowing how much money is available is used for computing price-dependent plans. There are other attributes in this class as the ones referring to the type of connection device that help on performing device dependent interaction (we do not focus on this aspect in this paper). An important attribute in this class is *current-visit* that is a reference to one instance of the Visit class. The *Visit* class has the information on the city that is being visited, the reasons of the visit, the free time the user has, as well as the activities (plans) that the user performed in the past, the activities (plans) that s/he rejected, etc. The ontology also accepts storing multiple past visits of the same or different users, so it can deduce new user preferences by performing machine learning or case-based reasoning on past visits.

A city is represented by the *City class* together with classes that describe transport means, places that can be visited, as well as the streets that compose the map of a city. For instance, streets are represented in terms of intersections and street sections, and contain information about the district they belong to, type of street, or traffic in that section (in case SAMAP would connect to on-line traffic information resources in the future). Using these classes, we can define the graph that represents the map of a city, so SAMAP can perform path planning to know how to go optimally from one place to another, according to a given user defined quality metric.

The *Place class* defines specific sites within a city that can be visited, such as restaurants, museums, generic buildings, bars, open-spaces, or theaters. It contains information related to the user, such as accessibility, price, or methods

[2] The representation is not intended to be UML-complaint.
[3] http://protege.stanford.edu.

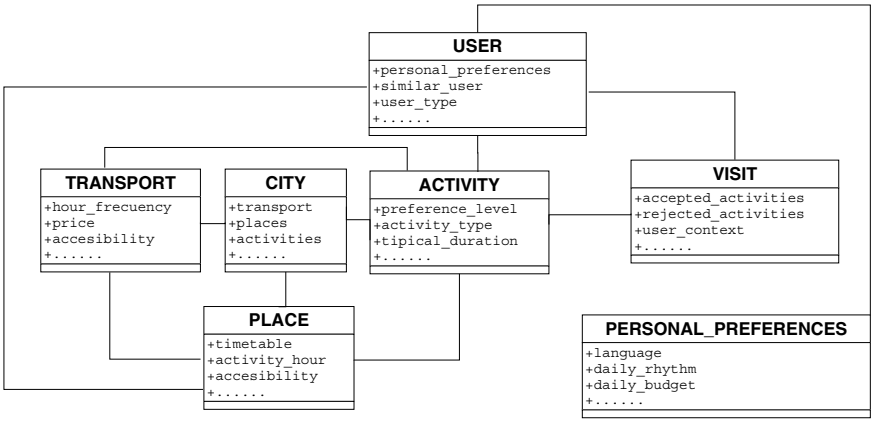

Fig. 2. High level view of the ontology developed for SAMAP.

of payment, as well as information needed for planning, such as timetables, which will be used to schedule the plan. The location in a street section is also specified for each place.

The transport means are represented by the *Transport class* and its subclasses, which denote specific transportation means: underground, railway, bus, taxi and walking. Some of them contain information on the graphs related to their itineraries.

Activities, which are represented in the *Activity class*, can be of two types: generic and specific. A generic activity is, for example, going to a museum, without specifying which one. This is useful when a user likes visiting museums in general. A specific activity relative to museums would be to visit the Hermitage museum. Therefore the system can recommend to visit several museums if "going to a museum" appears in the list of generic activities, or to visit an specific museum if it is in the list of specific activities. Planning goals will be automatically generated from the set of activities that the user has selected to perform within the city, as well as the ones that SAMAP has selected automatically from the user preferences and from previous visits of similar users to the same city. Examples of the activities that can be performed in a city are to visit a museum, a church, eat in a restaurant, go to a concert, attend a show, go to the cinema or theatre, etc. Each one of these types of activities is defined in a separate subclass.

While some of the attributes of the Activity class are useful for the planning process, others are important to select which places will be recommended to the user. For example, the attribute that determines the type of food helps the system to recommend a restaurant according to the user preferences. On the other hand, the attribute that indicates the typical duration of an activity helps to decide which activities could be performed taking into account the available time of the user.

The *Movement class* is a special subclass of Activity, given that it represents the action of going from one place to another. It provides a uniform way of representing the output of the planning agent; plans composed of activities that include visiting places, as well as going from one place to another. The system admits several types of transport means, including walking, and represents and computes several quality metrics related to them, as price, duration, distance, or utility for the user (related to preferences).

One of the advantages of having an explicit knowledge model such as the ontology we are using is that some elements of the ontology are recovered from Internet by means of wrappers. Information relative to cinemas, theaters, restaurants or museums is updated from Internet sites, which can improve the robustness of the plans. We plan to study in the future the relationship between the continuous access to updated information and reliability of plans.

4. The User Agent

The USER AGENT is the middleware between the system agents (i.e., user modelling and interface agent, CBR agent, and planning agent) and the user actions. Its main purpose is to accommodate system responses to users' behaviour, so it focuses in three functions: user interface; management of the information required by the rest of the system components; and, as its main contribution from the user perspective, building user models of needs and preferences.

With respect to the interface, there are some restrictions to be fulfilled in SAMAP. The system has been thought to work in portable devices (mobile phones, PDAs, etc.,) with Internet connection and to help different users to visit different cities. Under these conditions the main task for the interface is to provide the most relevant features from places, transport means, events, activities, etc., and to let the users manage their visits and their own personal data (see Fig. 3). With such variety and amount of data to be managed, simplicity has been chosen as the main principle (Alm, 2003) to guarantee an intuitive interface easy to use, understand, and navigate across. This is of particular importance when users are supposed to consult portable devices to make in-place decisions (Alberts, Michael, & Kim, 2002; Patch, 2001). In particular, we follow usability

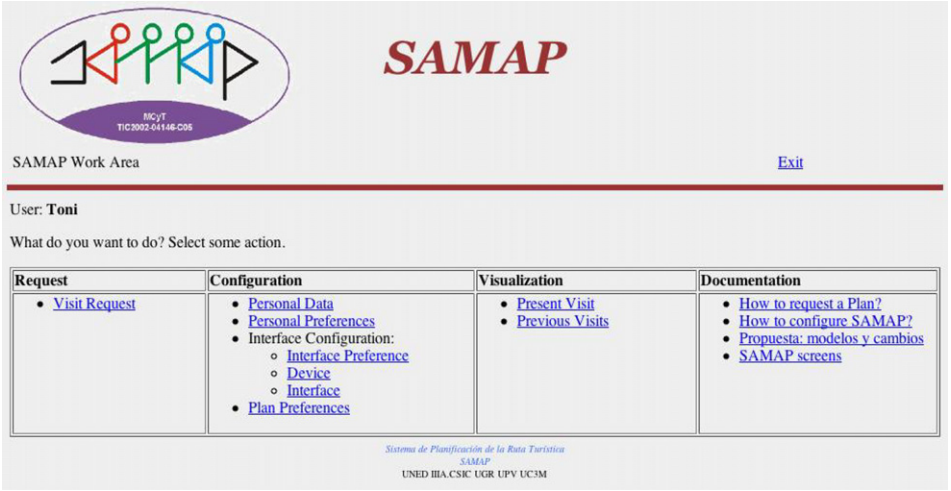

Fig. 3. Interface work area screen of SAMAP.

guidelines referring learnability, efficiency, memorability, errors, and user satisfaction (Nielsen, 1993). Mainly, we have focused on learnability because we have thought the most important characteristic from users' point of view is to learn and understand easily and quickly how to use the system and what the system can do. Other relevant feature we have considered is memorability because the system needs to manage all users' interactions, requests and answers for improving its replies, adaptations, etc. Users can manage their personal data, preferences and options. Let us consider a typical user of SAMAP. SAMAP interface starts asking for user identification to guide users to their specific working area where different system options are shown in a simple and organized way (see Fig. 3). If the user visits SAMAP for the first time, s/he must fill in her/his personal data to become a SAMAP user. From this working area the user can ask for a visit request, configure the system and interface, manage advised visits, and ask for more information when needed.

In more detail, the configuration section of the working area allows the users to update their personal preferences and data, and change their technical preferences of plans, which affect the plans that will be eventually advised. Likewise, users can configure some interface and device features depending on the particular situation in which the user is involved.

A visit request includes information about date (when the visit would be made), personal preferences (e.g., what activities would like to be made, or how much money the user has to spend), the user context (personal situations can change during the visit, e.g., if the user has a car, or s/he is hungry or thirsty, and they can alter the user preferences and options), and the user context (i.e., features related to the user situation, like weather conditions), which strongly affects user preferences and options. Once the user has entered the required data, the USER AGENT sends the relevant information to the CBR agent, which in turns filters the information to be provided to the planning agents. From the interface standpoint, the USER AGENT receives the system answer(s) which consist of one or more visit plans, where different activities are scheduled. These scheduled activities are shown to the user according to the interface restrictions in the visualization section of the work area, and the user can handle them according to his/her preferences, interests, etc. The user may choose some activities of the advised visit and reject others. The system can ask about these decisions and get feedback from the user. The user opinions about the advised visit are taken into account in the user modelling process, which provides the information to the rest of the system agents.

Therefore, the work of the USER AGENT is not only to manage the user interface but also to show the system functionalities, ask for the suitable information to the user, build a user model, transfer the user requests and information to others agents, obtain from other agents information for the user, show this information to the user, and allow the user to manage data which s/he has given to the system and s/he has got from the system.

This agent builds some models with information that is required by the rest of the system agents. In particular, it manages the *User Model* and the *Interface Model*. The former is in charge of grouping the relevant user information, including personal data and preferences, visit requests, and previous visits. The Interface Model groups data that the users have generated to configure for them SAMAP and their interactions.

Considering previous experiences in building adaptive user agents for educational environments (Santos, Boticario, Rodríguez, & Barrera, 2004) we are currently working on a USER AGENT version which considers similarities between users in a twofold approach. First, focusing on users with similar features and preferences. Here the USER AGENT will use personal data and preferences of the available users. These data are grouped in several attributes (most of them have literal values), which are used to classify the users in groups of users with similar needs

and preferences. To do so we are applying several machine learning algorithms (C4.5 Quinlan, 1993 and EM Dempster, Laird, & Rubin, 1977). Another area of interaction data we are considering is the user behaviour with respect to the system solutions. In this particular case, our focus is to compare comments and the score provided on chosen or rejected activities. The system provides advised activities to the users and the users may choose or reject them. The system asks the users the reasons for their acceptances or rejections. Moreover, SAMAP uses these answers to further come up with similarities among users who have been previously assigned to the same group. For this particular task the USER AGENT will use clustering algorithms. The final outcome will be a ranked list of similar users with respect to the particular target user. The highest position in the list corresponds to the most similar. Finally, this list will be send, with the rest of the data collected from the user visit request, to the CBR agent. Therefore, the CBR agent will be able to use that list to further investigate similarities among existing users.

5. The CBR agent

The goal of the CBR agent is to provide the planning agent a list of activities that an user could perform during a visit to a city. These activities are all consistent with the user's interests. Then, the planning agent will select a subset of them according to the availability and duration of the visit (see Section 6). Given a problem, the idea of CBR is to provide the solution (possibly adapted when necessary) of similar problems. In particular, the CBR agent will recommend to a tourist T with interests I a list of activities (i.e places to visit) performed by a subset of tourists with interests similar to T. Moreover, the CBR agent is capable of explaining why it recommends the current list of activities. In the following subsections, we explain the representation of the cases and places, the CBR method used for solving the problem, and how the CBR agent justifies the list of activities to the user.

5.1. Representation of cases and places

The CBR agent uses the part of the SAMAP ontology concerning the user-context, City and Place classes. In particular, the agent takes the information about user's interests, time availability and characteristics of the sites that can be visited (style, visit duration, etc). This information is useful for the CBR agent and is translated to an internal representation using the *feature terms* formalism (Armengol & Plaza, 2000). *Feature Terms* (also called feature structures or ψ-terms) are a generalization of first order terms. A feature term can be described as a labelled graph i.e. parameters are identified by name. Fig. 4a shows the representation of a tourist using feature terms. The root of this feature term is *T*-379 of sort *tourist* described by five features: `age`, `duration`, `kind-visit`, `rhythm`, and `interests`. The value of the feature interests is a feature

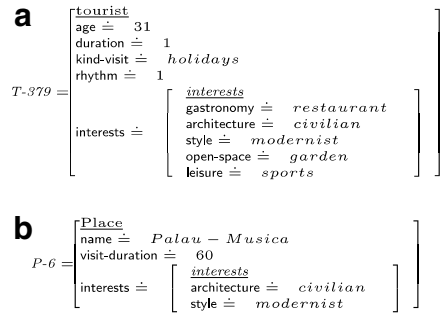

Fig. 4. Representation of a tourism case (a) and a city place (b) using feature terms.

term of sort *interests* described by five features: `gastronomy`, `architecture`, `style`, `open-space` and `leisure`. In particular, the tourist described in Fig. 4a is 31 years old, the visit is for holidays and it will take one day. The tourist is interested in gastronomy (particularly in restaurants), civilian architecture of modernist style, s/he likes open spaces (like gardens) and s/he also likes sports.

City places with some interest to be visited are also represented using feature terms. A feature term of sort *place* is described by three features: `name`, `visit-duration` and `interests`. In particular, the value of the feature `interests` is of the same sort that the feature `interests` in *tourist*. The reason is that both user interests and places are described using the same ontology. Fig. 4b shows the representation of a place, the `Palau de la Música`, that is interesting for its architecture; a civilian place of modernist style. The visit takes, in average, 60 min.

5.2. The CBR method

Given the description of a tourist T provided by the user agent (Section 4) the CBR agent uses the k-Nearest Neighbor (k-NN) algorithm (Dasarathy, 1991) to retrieve the k tourists most similar to T that already have visited the city. Most of similarity measures used by the *k-NN* compare objects represented as sets of attribute-value (propositional representation). However, this kind of representation has not enough expressive power to describe complex objects. For this reason, research on CBR dealing with relational representation of cases is currently growing. Relational cases are represented as a set of relations following the idea behind *Inductive Logic Programming*. In particular, it is necessary to define new similarity measures able to compare relational cases. Some attempts in that direction are the similarity measure used by the KGB system (Bisson, 1995), RIBL (Emde & Wettschereck, 1996), RIBL2 (Horváth, Wrobel, & Bohnebeck, 2001) and SHAUD (Armengol & Plaza, 2003). In particular, the CBR agent of SAMAP uses SHAUD, that compares relational cases represented using feature terms. SHAUD compares cases taking into account their structure, i.e. the part that is shared by the cases and the part that is not shared, using the ontology of cases.

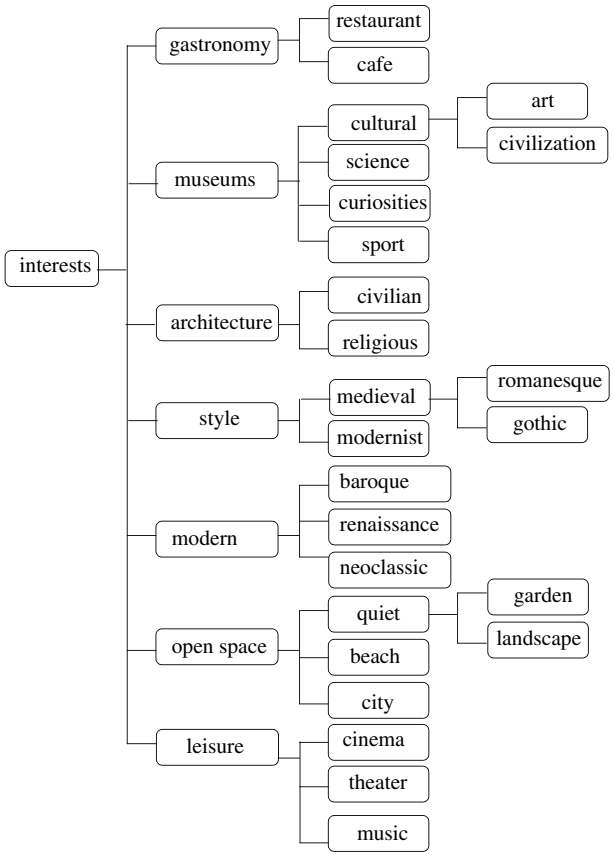

Fig. 5. Part of the ontology used to represent the user's interests.

The CBR agent focuses on the part of the SAMAP ontology concerning to interests of tourists and characteristics of the sites of a city (Fig. 5). In fact, each interest of a tourist (for instance, interest on modernist architecture) corresponds to, at least, one city place than can be visited (the `Palau de la Música` is modernist) in order to make the correspondence easy among user interests and places to visit. Interests are classified in six classes (sorts): gastronomy, museums, architecture, art style, open spaces and leisure. Each one of these sorts has subsorts; for instance, the sort *style* represents the art preferences of a user. The subsorts of *style* are the sorts *medieval* and *modernist*. In turn, the sort *medieval* has as subsorts *romanesque* and *gothic*. This ontology is used by SHAUD when assessing the similarity among the shared part of two cases. Thus, one tourist with interest on *romanesque* style is more similar to a tourist with interest on *gothic* than to another tourist with interest on *modernist* style. Given a problem p, a CBR method like *k-NN*, retrieves from the case base the k cases most similar to p. We call the set of these k cases the *retrieval set (RS)*. Then, during the reuse phase, the solution for p is built from the solutions of the cases in *RS*. Given a tourist T, the CBR agent of SAMAP uses *k-NN* with SHAUD as similarity measure to retrieve the k tourists (set *RS*) most similar to T. The reuse consists on selecting a subset of places $S \in RS$ to be visited ordered according to the percentage of visits. Let us suppose that each tourist $T_i \in RS$ has visited n_i places of a city, and let P be the set of places visited by at least one of the tourists in *RS*. Notice that the cardinality of P may be too high to be appropriate to the visit duration, i.e. T has a limited time of visit and s/he cannot visit the union of all the places visited by all the tourists in *RS*. For this reason, the CBR agent has to select a subset S of places in P in order to reduce the total number of options to be given to the planning agent. Fig. 6 shows the procedure followed by the CBR agent to build the set S. The idea of this procedure is to select the most visited places according to the tourist interests and also to adapt the number of places to the time available by the tourist. Moreover, places in S are ranked by the number of tourists with the same interests than T that have visited them. The planning agent takes into account this ranking when it elaborates the schedule of the visit.

5.3. Explanation of the result

Commonly, case-based systems justify their results by giving the user the capability to explore the cases considered as the most similar to the problem. The main shortcoming of this kind of explanation is that when the cases have a complex structure, the user can have some difficulties in understanding the solution of the problem at hand (Doyle, Tsymbal, & Cunningham, 2003; McSherry, 2005).

Procedure CBR (T):S
  T: knowledge on tourist, including *total-time* for the visit
  RS: set of the k tourists most similar to T
  P: places visited by at least one of the tourists in RS
  $S := \emptyset$ (subset of places to be visited)
  while(*total-time* > 0)
    order P according the number of tourists in RS that have visited it
    add the first element $p \in P$ to S
    $P := P - \{p\}$
    total-time := *total-time* − *place.visit-duration*
  return S

Fig. 6. Reuse oriented towards finding most relevant places to visit.

$$Explanation = \left[\begin{array}{l} \underline{\text{tourist}} \\ \text{age} \doteq 34.25 \\ \text{duration} \doteq 3 \\ \text{visit-type} \doteq familiar \\ \text{rhythm} \doteq 1 \\ \text{interests} \doteq \left[\begin{array}{l} \underline{interests} \\ \text{gastronomy} \doteq restaurant \\ \text{architecture} \doteq civilian \\ \text{style} \doteq medieval \\ \text{open-space} \doteq quiet \\ \text{leisure} \doteq leisure \end{array}\right] \end{array}\right]$$

Fig. 7. Example of an explanation given by the CBR agent.

In SAMAP the CBR agent is capable of justifying the result by means of a symbolic description (a feature term) that contains all that is shared by the problem and the cases retrieved. In Armengol and Plaza (2005), we proposed to explain the retrieval and the solution of a classification problem using the notion of *anti-unification*, i.e. the most specific generalization of a set of cases. The idea is to explain the retrieval to the user by showing all the structure shared by the problem and the cases retrieved, i.e. what is shared by the current tourist and the k tourists considered as the most similar to him/her.

Fig. 7 shows an example of explanation (represented as a feature term) given by the `cbr` agent to justify the retrieval of a particular set of cases. The explanation is a symbolic description composed only by the set of features common to the problem and all the retrieved cases. When the value of a common feature is symbolic, then it is generalized by the most specific generalization (the anti-unification); when the value is numerical then the value of the feature in the explanation is the average of all the values. In the example, the new tourist shares with the most similar tourists that all them are around 34.25 years old (age average), the visit has a duration of 3 days, and it is a familiar visit (i.e. with kids) at low rhythm. The common interests of all the tourists are the restaurants, the civilian medieval architecture, quiet open spaces. Also all tourists like leisure, but do not share a particular kind of leisure (for instance, not all like sports).

5.4. Recommendation of an equivalent visit

Let us suppose that the CBR agent has a extensive case base with activities performed by tourists in one city, namely C_1. Let us suppose now that a tourist wants to visit a city C_2, but there are not enough cases in the case base to recommend a good selection of places to visit. The CBR agent of SAMAP is capable of recommending a visit to C_2 based on the visits that similar tourists have performed to C_1. Equivalent places are based on both, the interest of the places and the percentage of visits received by the place. The idea is that when several places have the same features concerning tourist's interest, then the CBR agent recommends those places with higher percentage of visits. For instance, let us suppose that a tourist has visited the wax museum of Barcelona and that s/he wants now to visit Valencia (that does not have a wax museum). The wax museum is classified as a "museum of curiosities" and receives a percentage of 5.5% visits. The CBR agent searches Valencia looking for "museums of curiosities" having a percentage of visits close to 5.5%. In this example, the CBR agent will recommend a visit to the *Museo fallero* of Valencia since it has 6% of visits. This procedure is repeated for each place the tourist visited in his first trip to Barcelona to recommend similar places in Valencia. Finally in that case, the CBR agent also uses the anti-unification to explain the equivalence among the places.

6. The Planning agent

One of the inputs of the planning agent is the list of activities selected by the CBR agent. This list is not directly the goal of the planning problem, even if the CBR has ranked activities and made a selection, as it can still contain more places than the user will be able to visit because of schedule or movement constrains among places. Therefore, the planner must select which of them should be included in the plan. Moreover, the planner must schedule each visit according to the place timetable, deal with the city map, etc. This planning task has several features that make it hard for current planners:

- *Time management.* Each visit should be scheduled according to the opening hours of each place and the expected duration of the visit according to the user model. Also, it should consider the time to go from a place to another.
- *Preferences.* The ontology contains knowledge about user preferences or constrains such as time-to-eat, utilities of visiting, places, types of food, etc.
- *Management of numerical values.* The prices of the visits, meals and transports must not exceed the available budget.
- *Locations.* The planner should indicate how to go from one place to another, that is, which transport the user should take. In case it is preferable to walk, the planner should specify which route the user should follow.
- *Goals.* We can specify three types of goals:
  - totally instantiated goals, i.e., visit a specific museum,
  - partially instantiated goals, i,e., generic goals like visiting any museum,
  - a metric indicating that the plan must maximize its utility.

Moreover, not all the available places must be visited, that is, not all goals will be achievable, because of scheduling constrains related to timetables (one cannot enter Prado's museum at night) or to the available time of the user (s/he cannot visit five places if s/he only has time to visit two). This problem is related to the oversubscription problem in planning (Smith, 2004) and scheduling (Kramer & Smith, 2005).

Some of these features can be handled by some temporal planners (IXTET HSTS:.). However, we will focus on the ones that are more specific:

1. *Each visit is scheduled according to the opening hours of each place.* This implies that we need an explicit management of the current time. In addition to the restrictions that existing temporal planners take into account when scheduling an action, our system must consider the current time because a place cannot be visited if it is closed. This also causes that there can be empty time points, that is, time points when no action is executed (from the point of view of the user, an empty time point is free time). This free time is not handled by most temporal planners since every time point has an associated action.
2. *The plan indicates how to go from one place to another.* This task can be performed (theoretically) by any planner. But our system has to deal with all the transport means (subway, bus, taxi,...) in a city and the detailed routes, street by street, one can take to walk from one place to another. If this is handled by the planner also, it can yield to a very complex planning task.
3. *The objective of the problem contains partially instantiated goals.* This means that the planner must select only one specific place whose type is equal to the partially instantiated goal.
4. *The plan is computed in order to maximize the utility of the visits.* In a general planning problem, even when we deal with durative actions and numerical values, we specify the goal as a set of literals that must be true at the end. However, in our case, there is not such a set of literals; our goal is to maximize the utility of the plan. This kind of problems were introduced in the International Planning Competition held in 2003 in the hard numeric track, but only few planners (MIPS Edelkamp, 2002, SHOP2 Nau et al., 2003 and TLPlan Bacchus & Ady, 2001) were able to solve them, the last two of them requiring manual coding of the domain.

As it is difficult to use any of these planners for solving the whole planning problem, we propose a hybrid system composed by a planner and some other modules, based on the ideas in Fox and Long (2001). As we said above, the input of the system is ranked list of activities that represents the places of interest of the user (including eating and leisure places) together with a number indicating their utility, i.e the *satisfaction degree* that visiting such site will potentially provide the user (computed by the CBR agent). Each activity can be totally or partially instantiated, as in `visit museum of Modern Art` or `visit a museum`. The output is one or more tourist plans which contain a list of scheduled visits along with the indications about how to move from one place to another. The system also computes the cost of the plans trying to maximize its quality according to the established metric. By default, the quality metric is to maximize the total utility, but it could be to diminish the cost, a combination of both or any other one.

The final step of the planning system is to store the generated plans into the ontology. Fig. 8 shows an schematic view of the architecture of the planner agent. It is composed of five modules:

- *Translator* module: it transforms the original list of activities into the predicates required by the planner. The output is the initial state of the problem, the goals (list of activities) and the transport graph, i.e, all the information related with the transportation (buses, underground, users preferences ...). It also transforms the generated plans into the corresponding instances of the ontology. This module will be described in more detail in Section 6.1.

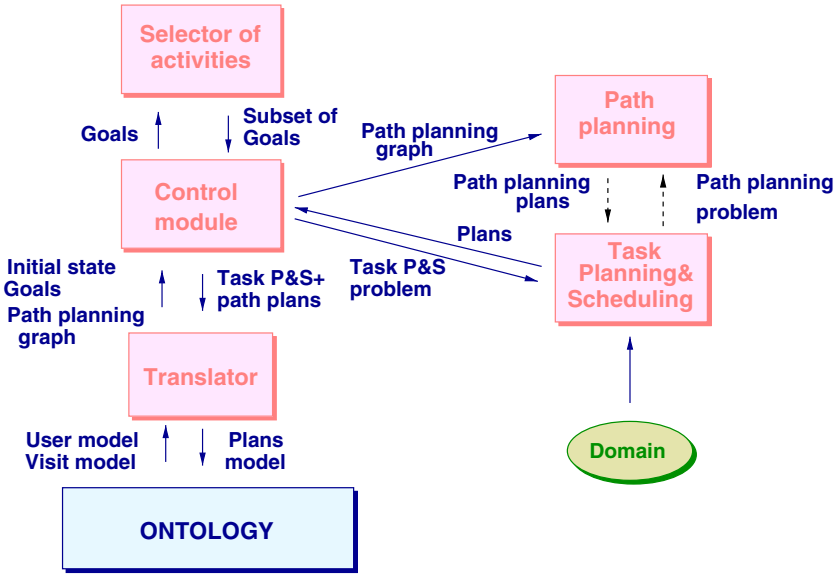

Fig. 8. Planning system architecture.

- *Control* module: it coordinates the rest of modules providing the input that they need in the suitable format and building the solutions from their outputs. It can invoke the Transport module or let the Planner module do it.
- *Selector of activities* module: the aim of the system is to maximize the utility of the plan and/or to satisfy some specific or generic goals, but not to fulfill all the available activities. This module selects the most appropriate actions to be solved by the planner each time. This module has been designed to have a similar behaviour to a tourist. For example, it can select first those activities which are near places with the highest utility. This module will be explained in detail in Section 6.2.
- *Planner* module: it generates the plans. Its input is a domain theory (set of operators and types hierarchy), planning problem (initial state and goals, list of activities), a quality metric, a cost bound and a time bound. It generates one or more plans that solve the problem in the given time bound, taking into account that the cost of the solutions cannot be greater than the cost bound, according to the quality metric. There can be more than one planner module each with a different planning technique (see Section 6.3). The Control module compiles all the results. This module can directly communicate with the Transport module.
- *Transport* module: it receives an origin and a destination point and returns the transport subplan for moving a person from the origin to the destination. It also returns the cost and time of the itinerary.

6.1. Translator

We describe here the interfaces needed to translate the knowledge contained in the ontology that is relevant for planning, and translate back the resulting set of plans into the ontology, for further use by other components. Fig. 9 shows how the translation is performed in SAMAP.

The input knowledge to the planner component is stored in the ontology in different classes as it was described earlier. In particular, from the point of view of planning, the most relevant knowledge contained in the ontology refers to the city and the user and his/her visit. Every specific city contains instances of city related classes in a CLIPS file. The planning agent is interested in those classes that refer to the places the user can visit, to the list of activities recommended by the CBR reasoning agent, and to the topology of the city and transports; that is, streets, intersections, buses, underground, places, etc. There are two files with the former information: CITY. PINS and VISIT. PINS. The knowledge about streets and transports is translated into a path planning problem (how to go from one place of the city into another one). The path planning problem can be solved by the planner at the same time as solving the planning problem (what activities to perform and when), or solved by the transport module described in the previous section.

On the other side, the translator is also in charge of building the initial state and goal of the planning problem. The initial state contains all those predicates that refer to the situation, schedule, price, ..., of each place together with the available schedule and the current location of the user, etc. The goal of the problem represents, in our context, the list of activities that the previous two modules have selected as relevant (useful) for the user. However, as we explained above, the existing planners try to satisfy all the predicates specified in the goal. For this reason, the selector of activities first shrinks the list of recommended activities to the most appropriate ones according to some other criteria (proximity, etc.). We have used CLIPS for this

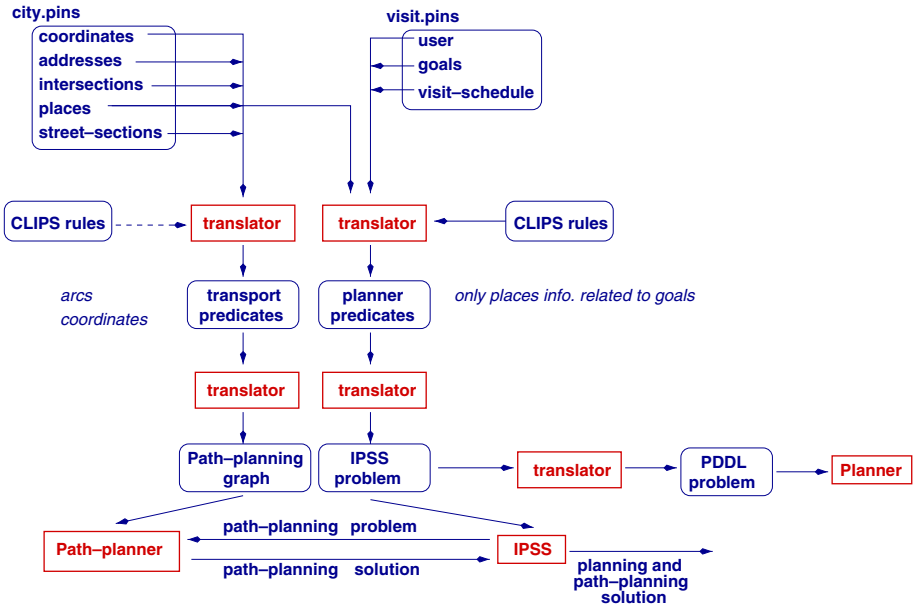

Fig. 9. Translation between ontology instances and the planners within SAMAP.

translation given that it provides a declarative and easy way of performing the translation. However, the transport problem translator has been re-programmed in Common-Lisp due to the size of the path planning graphs (a city such as Valencia (Spain) has over 7600 nodes and over 22,000 arcs only for walking, without considering any transport network), although it could have been easily done with more appropriate tools, such as LEX and BISON.

6.2. Selector of activities

This module is in charge of selecting the activities that the planner will attempt to achieve in a plan, that is the list of goals. Notice that the CBR agent computes a list of activities potentially interesting for the user, but it is very unlikely all of them can be performed within the available time of the user. For this reason, the selector shrinks this list of activities in two steps:

1. The first step uses a clustering algorithm to group the activities according to their geographical situation. That is, those activities that are separated by a given walking distance (places that are close enough to walk among them) are grouped together into the same cluster.
2. The second step computes the set of activities A that maximizes the following function:

$$F = \alpha \times \sum_{a_i \in A} \text{utility}(a_i) - \beta \times \left(\text{AT} - \sum_{a_i \in A} \text{duration}(a_i) \right) - \gamma \times \text{NC}$$

where AT is the available time of the user and NC represents the number of different clusters the activities in A belong to. The goal of this function is to reduce the user free time and the time used in moving among places as much as possible. This way the planner will be able to allocate the maximum number of visits. This behaviour can be modified by changing the values of α, β and γ.

6.3. Planning module

Once the transport and planning translators generate the initial state and goals of the planning problem (possibly also a path planning problem), task planning can start. We are using now three planners, METRIC-FF (Hoffmann, 2003), IPSS (Rodríguez-Moreno, Borrajo, Cesta, & Oddi, 2007) and SIADEX (Castillo, Fdez-Olivares, García-Pérez, & Palao, 2006) (HTN planning). The translators generate PDL (PRODIGY Description Language) problems for IPSS, but we have also built a PDL to PDDL translator, so we can also use FF and SIADEX for solving planning problems.

6.3.1. FF

The first solution we adopted was using a planner such as FF (Hoffmann & Nebel, 2001). The main problem it has in relation to SAMAP is that calling the path-planner from each state can yield to an exponential growth of successor nodes in the search tree, since from each node in the path planning problem a tourist can go to many others (all intersections in a city). So the whole transport (or path planning problem) gets translated into PDDL (Fox & Long, 2002) as part of the initial state. In this case the path planning graph (nodes are intersections, and addresses and arcs are transport means that move one person from one node or another) is represented with two predicates: `coordinates` that represent where nodes in the graph are; and `arc` that contains information about the type of arc (walk, bus, underground, ...), origin and destination, utility (for some people walking is better than taking the bus, but worse than taking the underground), distance, time, price, and cost (we can represent and reason about any other type of cost).

Under this approach, the path planning problem is defined inside the task planning problem (planning of activities). Move actions require an arc between two nodes in the path planning graph and the planner deals with both problems (activities and path) simultaneously. The advantage of this approach is that the planner is informed of the path planning results when planning the activities. The disadvantage is that it greatly increases the complexity, making the problem solving intractable even for simple problems.

Heuristic planners, such as FF (Hoffmann & Nebel, 2001), easily adapt their behaviour for solving path planning problems. This results from the way in which their heuristics usually work. Solving a relaxed problem means, in the presence of explicit path planning graphs, that they will find the minimum number of move actions that should be applied in order to solve the path planning part of the problem. The main difficulty arises when handling different cost metrics. Although there exists a version of FF that handles cost metrics, METRIC-FF (Hoffmann, 2003), it is not possible to define different metric measures for the activities planning and the path planning part. In our particular domain, we would be interested in finding the best solution paths in terms of time (we prefer to go as fast as possible from one place to another), but we would be interested in obtaining the best planning of activities in terms of another cost metric, such as utility.

In short, the approach of having the transport module inside the planner itself brings some advantages, but also a few disadvantages. From our experience, when dealing with such large path planning graphs the best option is to have a separate transport module just devoted to solve the path planning problems.

6.3.2. IPSS

The second solution (a separate transport problem) is the one we use with IPSS, given that it is quite easy to plug-in new execution components to this planner. In this case, the translator generates a path planning graph from the knowledge about the city in the ontology as seen in Section 6.1. Then, at planning time, every time IPSS tries to

apply a transport operator (such as walking from A to B), it calls a path planner (for instance, an A^* algorithm) with the origin and destination. And the path planner will use the planning graph to compute a solution, which returns to the planner. An advantage of being a backward chaining planner is that it only tries to call the path planner when it needs to go to a given place.

Fig. 10 shows the interaction between the task planner and the A^* planner. When the planner needs to know whether an instantiation of the `move` operator can be used to achieve a goal of having the user at some place, for a visit, then it calls the path planner that returns a value for a variable.

Fig. 11 shows the implementation of the `move` operator in pdl. The function `gen-from-pred` captures the value of variables from things that are true in the current state of the search. So, for instance, (`gen-from-pred (at` ⟨u⟩ ⟨pu⟩)) retrieves the value of the unbound variable ⟨pu⟩ from the place the user currently is at. The function `path-planning` calls the path planner with the current place of the user, and the place of where it should go (bound by the goal (at ⟨u⟩ ⟨p⟩)). This function returns a valid value for the ⟨duration⟩ variable; that is the time to move from the initial to the final place using the shortest path in terms of time (we can use any other metric). It then checks whether it has still time to move given the maximum time available for that day and user. Finally, it captures the price for that movement (in case of using a public transport or taxi) and the new time (after moving).

6.3.3. SIADEX

The third solution uses SIADEX (Castillo et al., 2006) a temporal HTN planner to explore a different perspective with the following features:

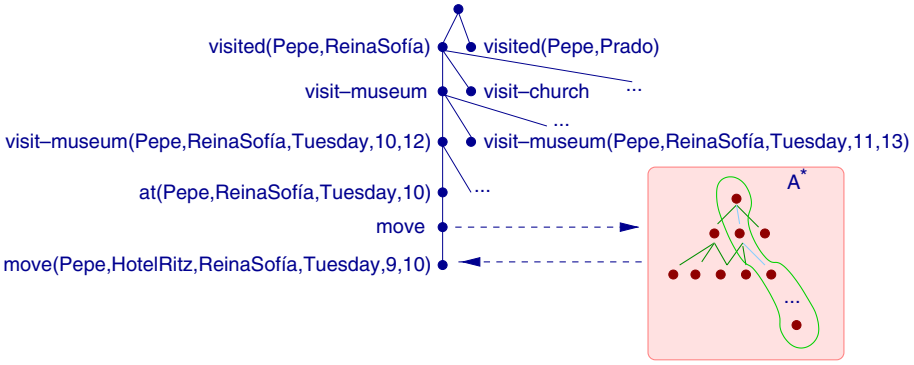

Fig. 10. Example of interaction between IPSS and a path planner implementing a A^* search.

```
(OPERATOR MOVE
   (params <u> <pu> <p> <day> <time> <new-time> <duration> <price> <utility>)
   (preconds
      ((<u> USER)
       (<day> DAY)
       (<p> PLACE)
       (<pu> (and PLACE (gen-from-pred (at <u> <pu>))))
       (<time> (and TIME (gen-from-pred (current-time <u> <day> <time>))))
       (<duration> (and DURATION (path-planning <pu> <p>)))
       (<end-ava> (and TIME (gen-from-pred (end-time-available <u> <day> <end-ava>))
                           (in-time <duration> <time> <end-ava>)))
       (<money> (and MONEY (gen-from-pred (money-available <u> <money>))))
       (<price> (and MONEY (get-transport-cost) (<= <price> <money>)))
       (<utility> (and UTILITY (get-transport-utility)))
       (<new-time> (and TIME (increase <time> <duration> <new-time>))))
      (at <u> <pu>))
   (effects
      ((<new-money> (and MONEY (decrease <money> <price> <new-money>))))
      ((del (money-available <u> <money>))
       (add (money-available <u> <new-money>))
       (del (at <u> <pu>))
       (add (at <u> <p>))
       (del (current-time <u> <day> <time>))
       (add (current-time <u> <day> <new-time>))))
   (costs ()
      ((PRICE <price> )
       (UTILITY <utility>)
       (MAKESPAN <duration>)
       (OTHER-METRIC (compute-metric <utility> <price>)))))
```

Fig. 11. Operator that implements the move operation.

- This approach also uses a separate transport module, so that the planner designs the visits, while the transport module takes into account the best way to go from one place to another, including several transportation means like buses, taxis or walking.
- The use of a hierarchy of tasks (Nau et al., 2003) allows to define more types of top level goals. Apart of having defined totally instantiated goals (visit a specific museum) and partially instantiated goals (visit any museum), we can define abstract goals with different levels of granularity, like perform any type visit (for example, I have some free time, what could I visit in the neigborhood?) or do a cultural visit (museums, collections, or shows).
- SIADEX uses a Simple Temporal Network to store all the underlying temporal knowledge. This allows an easy encoding of opening hours as timed initial literals (Edelkamp & Hoffmann, 2004) and does not need to explicitly handle empty time points, nor free time since actions are placed in a timeline according to the temporal constraints imposed by the opening hours.

After the planning process, the planners can either return no solution (when all goals cannot be scheduled in the given time frame), or a list of plans (in the case of FF only one). If no solution was found, then the control module calls the activities selector module to return a new sublist of goals and the planner is called again with the same initial state and the new subset of goals. If a set of solutions was found, it is translated back to ontology instances in a solution file. Every planning solution is a list of instantiated operators. These operators can be either visit some place, attend some event (cinema, theatre, concert, etc.) or perform a move operator (walk from one place to another, take a bus from one stop to another, etc.). If one of the routes includes walking from one point to another, a detailed explanation of how to get to the final destination is included. We have also created an external application

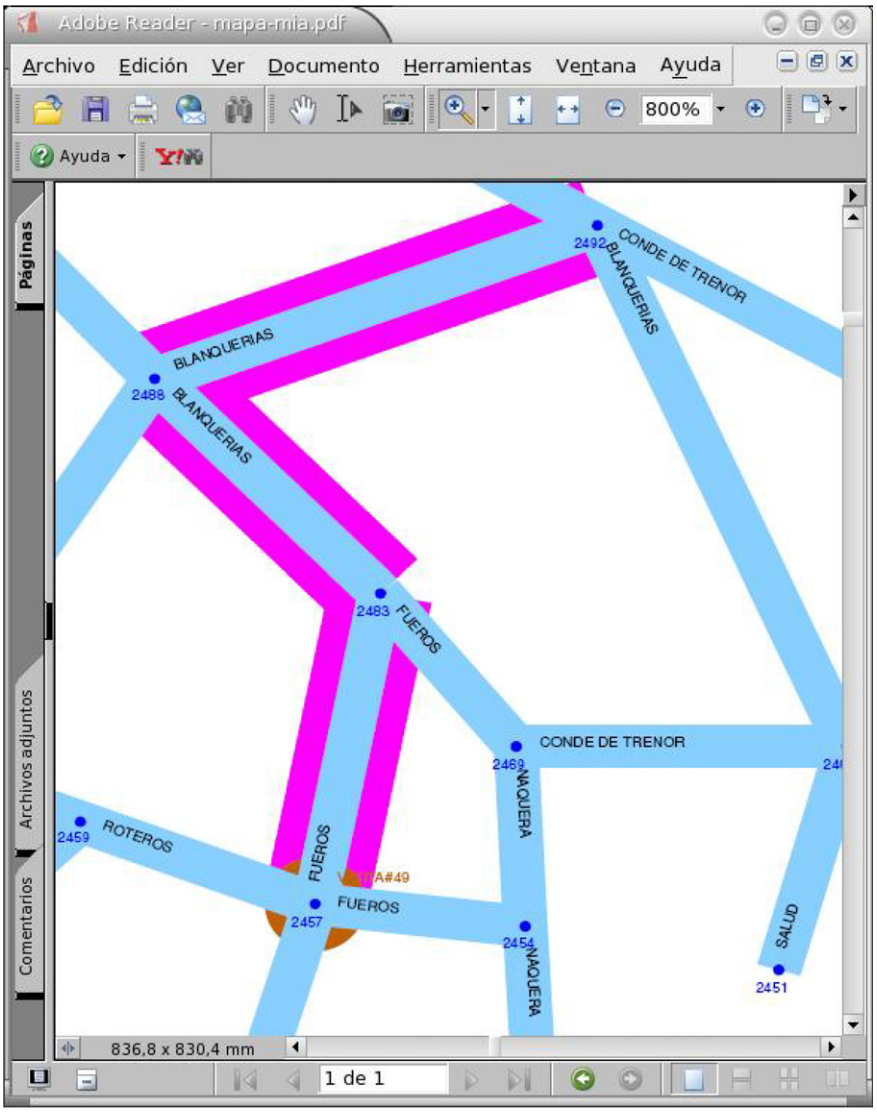

Fig. 12. Guided walk by downloading detailed maps in PDF or JPG.

that translates this explanation into a georeferenced map that may be delivered in JPEG or PDF formats to users and displayed in their device, either a PDA or a mobile phone (see Fig. 12 for an example of part of a map). The solutions file is sent to the user module for further processing (selection of a plan by the user, modification of conditions and replanning, or execution of the plan).

7. Conclusions

The application of AI Planning & Scheduling to real world problems is a growing area that attracts more researches each day. Also, the development of Internet, together with new hardware devices with good connection capabilities, is causing a social and economic impact in many aspects. Here, we have described an application for assisting any tourist for planning the visit to a city, using a PDA or a third generation mobile phone. These plans are adapted to the user preferences proposing only a list of activities that could be really interesting and achievable for him/her.

SAMAP has been built as a multiagent system, consisting of three main agents: user modelling and interface agent, CBR agent, and planning agent. In order to facilitate the information exchange among these agents, we have developed an ontology. It stores information about the user and his/her preferences, the activities that can be performed in a city and the city itself.

Acknowledgement

This work has been partially supported by the Spanish MCyT under Project TIC2002-04146-C05.

References

Alberts Michael, J., & Kim, L. (2002). An overview of web design issues for personal digital assistants. *Technical Communication, 49*(1), 45–60.

Alm, I. (2003). Designing interactive interfaces: theoretical consideration of the complexity of standards and guidelines, and the difference between evolving and formalised systems. *Interacting with Computers, 15*, 109–119.

Ambite, J. L., Barish, G., Knoblock, C. A., Muslea, M., Oh, J., & Minton, S. (2002). Getting from here to there: interactive planning and agent execution for optimizing travel. In *Fourteenth innovative applications of artificial intelligence conference (IAAI)*. Edmonton, Alberta, Canada, AAAI: AAAI Press.

Armengol, E., & Plaza, E. (2005). Using symbolic similarity to explain case-based reasoning in classification tasks. In *Explanation-aware computing. AAAI fall symposium. Technical report FS-05-04*, AAAI Press, pp. 1–9.

Armengol, E., & Plaza, E. (2000). Bottom-up induction of feature terms. *Machine Learning, 41*(1), 259–294.

Armengol, E., & Plaza, E. (2003). Relational case-based reasoning for carcinogenic activity prediction. *Artificial Intelligence Review, 20*(1–2), 121–141.

Bacchus, F., & Ady, M. (2001). Planning with resources and concurrency: a forward chaining approach. In *International joint conference on artificial intelligence (IJCAI-2001)* (pp. 417–424).

Bisson, G. (1995). Why and how to define a similarity measure for object-based representation systems. In *Proceedings of 2nd international conference on building and sharing very large-scale knowledge bases (KBKS)* (pp. 236–246). Enschede (NL): IOS press.

Camacho, D., Aler, R., Borrajo, D., & Molina, J. M. (2005). A multi-agent architecture for intelligent gathering systems. *AI Communications, 18*(1), 15–32.

Castillo, L., Fdez-Olivares, J., García-Pérez, O., & Palao, F. (2006). Efficiently handling temporal knowledge in an htn planner. In *Proceedings of the sixteenth international conference on automated planning and scheduling, ICAPS'06*.

Dasarathy, B. (1991). *Nearest neighbor (NN) norms: NN pattern classification techniques*. IEEE Computer Society Press.

Delgado, J., & Davidson, R. (2002). Knowledge bases and user profiling in travel and hospitality recommender systems. In *Proceedings of the ENTER 2002 conference* (pp. 1–16). Innsbruck, Austria: Springer Verlag.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of Royal Statistical Society, 39*, 1–22.

Doyle, D., Tsymbal, A., & Cunningham, P. (2003). *A review of explanation and explanation in case-based reasoning. Technical report TCD-CS-2003-41*, Department of computer Science, Trinity college, Dublin.

Edelkamp, S. (2002). Symbolic pattern databases in heuristic search planning. In *AIPS'02*. AAAI Press.

Edelkamp, S., & Hoffmann, J. (2004). *The language for the 2004 international planning competition*. <http://andorfer.cs.uni-dortmund.de/~edelkamp/ipc-4/pddl.html>.

Emde, W., & Wettschereck, D. (1996). Relational instance based learning. In L. Saitta (Ed.), *Machine learning – proceedings 13th international conference on machine learning* (pp. 122–130). Morgan Kaufmann Publishers.

Fesenmaier, D. R., Ricci, F., Schaumlechner, E., Wöber, K., & Zanellai, C. (2003). DIETORECS: travel advisory for multiple decision styles. In *Proceedings of the ENTER 2003 conference* (pp. 29–31). Helsinki, Finland: Springer Verlag.

Fox, M., & Long, D. (2001). Hybrid stan: identifying and managing combinatorial optimisation sub-problems in planning. In *Proceedings of IJCAI'01*.

Fox, M., & Long, D. (2002). *PDDL2.1: An extension to PDDL for expressing temporal planning domains*. Durham, UK: University of Durham.

Heflin, J., & Muñoz-Avila, H. (2002). LCW-based agent planning for the semantic web. In Ontologies and the semantic web. *Papers from the 2002 AAAI workshop WS-02-11* (pp. 63–70), Menlo Park, CA, AAAI Press.

Hoffmann, J. (2003). The Metric-FF planning system: translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research, 20*, 291–341.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research, 14*, 253–302.

Horváth, T., Wrobel, S., & Bohnebeck, U. (2001). Relational instance-based learning with lists and terms. *Machine Learning Journal, 43*(1), 53–80.

Knoblock, C. A., & Minton, S. (1998). The ariadne approach to web-based information integration. *IEEE Intelligent Systems, 13*(5).

Kramer, L. A., & Smith, S. F. (2005). Maximizing availability: a commitment heuristic for oversubscribed scheduling problems. In J. K. Shlomo Zilberstein & S. Koenig (Eds.), *Proceedings of ICAPS'05* (pp. 272–280). Monterey, CA, USA: AAAI Press.

McSherry, D. (2005). Explanation in recommender systems. *Artificial Intelligence Review, 24*(2), 179–197.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Mur, J. W., & Wu, D. (2003). SHOP2: an HTN planning system. *Journal of Artificial Intelligence Research, 20*, 379–404.

Nielsen, J. (1993). *Usability engineering*. San Francisco: Morgan Kaufmann.

Patch, K. (2001). PDA interface keeps a low profile. *Technology Research News*.

Plaza, E., Arcos, J. L., & Martín, F. (1997). Cooperative case-based reasoning. In G. Weiss (Ed.), *Distributed artificial intelligence meest machine learning, number 1221 in Lecture notes in artificial intelligence* (pp. 180–201).

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Rodríguez-Moreno, M. D., Borrajo, D., Cesta, A., & Oddi, A. (2007). Integrating planning and scheduling in workflow domains. *Expert System with Applications, 33*(2), 389–406.

Santos, O.C., Boticario, J., Rodríguez, A., & Barrera, C. (2004). Support to learners based on implicit collaborative interactions. In *IAIC 04. Workshop artificial intelligence in computer supported collaborative learning*.

Smith, D.E. (2004). Choosing objectives in over-subscription planning. In *Proceedings of ICAPS'04*, pp. 393–401, Whistler, Canada.