Utilizing Semantic Web and Software Agents in a Travel Support System

Maria Ganzha
Elbląg University of Humanities and Economy, Elbląg, Poland

Maciej Gawinecki
Adam Mickiewicz University, Poznań, Poland

Marcin Paprzycki
SWPS University, Warsaw, Poland

Rafał Gąsiorowski, Szymon Pisarek, Wawrzyniec Hyska
Warsaw University of Technology, Warsaw, Poland

Introduction

Let us consider a business traveler who is about to leave Tulsa, Oklahoma for San Diego, California. Let us say that she went there many times in the past, but this trip is rather unexpected and she does not have time to arrange travel details. She just got a ticket from her boss' secretary and has 45 minutes to pack and catch a taxi to leave for the airport. Obviously, she could make all local arrangements after arrival, but this could mean that her personal preferences could not be observed and also that she would have to spend time at the airport in a rather unpleasant area where the courtesy phones are located or to spend long time talking on the cell phone (and listen to call-waiting music) to find a place to stay etc. Yes, one could assume that she could ask her secretary to make arrangements, but this would assume that she does have a secretary (which is now a rarity in the cost-cutting corporate world) and that her secretary knows her personal preferences well.

Let us now consider another scenario. Here, a father is planning a family vacation. He is not sure where they would like to go, so he spends countless hours on the Web, going over zillions of pages, out of which only few match his preferences. Let us note here, that while he will simply skip pages about beauty of Ozark Mountains – as his family does not like mountains, but he will "have to" go over a number of pages describing beach resorts. While doing this he is going to find out that many of possible locations are too expensive, while others do not have kitchenettes that they like to have – as their daughter has special dietary requirements and they prefer to cook most of their vacation meals themselves.

What do we learn from these two scenarios? In the first case, we have a traveler who, because of her unexpected travel, cannot engage in e-business as she does not have enough time to do it, while she could definitely utilize it. Yes, when in the near future airplanes will have Internet access, she will possibly be able to make proper arrangement while traveling, but this is likely going to be an expensive proposition. Furthermore, the situation when traveler is spending time on the plane to make travel arrangements is extremely similar to the second scenario, where user is confronted with copious volumes of data within which (s)he has to find few pertinent gems.

What is needed in both cases is creation of a travel support system that would work as follows. In the first case, it would know personal preferences of the traveler and on their basis, while she is flying and preparing for the unexpected business meeting, would arrange accommodations in one of her preferred hotels, make a dinner reservation in one of her favorite restaurants and negotiate a "special appetizer promotion" (knowing that she loves the shrimp cocktail that is offered there). Upon her arrival in San Diego, results would be displayed on her PDA (or a smart cell phone) and she could go directly to the taxi or to her preferred car rental company. In the second case, the travel support system would act as an interactive advisor – mimicking the work of a travel agent – and would help select travel destination by removing from considerations locations and accommodations that do not fit the user profile, and personalizing content delivery further – by prioritizing information to be displayed and delivering one that would be predicted to be most pertinent first. Both these scenarios would represent an ideal way in which e-business should be conducted.

The aim of this chapter is to propose a system that, when mature, should be able to support needs of travelers in exactly the above described way. We will also argue that, and illustrate how, Semantic Web technologies combined with software agents should be used in the proposed system. We proceed as follows. In the next section we briefly discuss current state-of-the-art in agent systems, semantic web and agent based travel support systems. We follow with a description of the proposed system illustrated by UML diagrams of its most important functionalities. We then discuss how to work with ontologically demarcated data in the world where such resources are practically nonexistent. Finally, we show how RDF demarcated data is to be used to support personal information delivery. We conclude with description of current state of implementation and plans for further development of the system.

Background

There are two main themes that permeate the scenarios and the proposed solution presented above. These are: *information overload* and need for *content personalization*. One of the seminal papers that addresses exactly these two problems was published by P. Maes in 1994 (Maes, 1994). There she suggested that it will be intelligent software agents that will solve the problem of information overload. In a way it can be claimed that it is that paper that grounded in computer science the notion of a *personal* software *agent* that acts on behalf of its user and autonomously works to deliver desired personalized services. This notion is particularly well matching with travel support, where for years human travel agents played exactly the role that personal agents are expected to mimic. Unfortunately, as it can be seen, the notion of intelligent personal agent, even though extremely appealing, does not seem to materialize (while its originator has moved away from agent research into a more appealing area of ambient computing).

What can be the reason for this lack of development of intelligent personal agents? One of them seems to be the truly overwhelming amount of available information that is stored mostly in a human consumable form (demarcated using HTML to make it look "appealing" to the viewer). Even a more recent move toward the XML as the

demarcation language will not solve this problem as XML is not expressive enough. However, a possible solution to this problem has been suggested, in the form of semantic demarcation of resources or, more generally, the Semantic Web (Berners-Lee, 2001, Fensel 2001). Here it is claimed that when properly applied, demarcation languages like RDF (RDF, 2005), OWL (OWL, 2005) or DAML (DAML, 2005) will turn human-enjoyable Internet pages into machine-consumable data repositories. While there are those who question validity of optimistic claims associated with the Semantic Web (Zaslavsky, 2004, Orłowska, 2005) and see in it only a new incarnation of an old problem of unification of information stored in heterogeneous databases – problem that still remains without general solution – we are not interested in this discussion. *For the purpose of this chapter we assume that the Semantic Web can deliver on its promises and focus on how to apply it in our context.*

In our work we follow two additional sources of inspiration. First, it has been convincingly argued that the Semantic Web and software agents are highly interdependent and should work very well together to deliver services needed by the user (Hendler, 1999, 2001). Second, we follow the positive program put forward in the highly critical work of Nwana and Ndumu (Nwana, 1999). In this context we see two ways of proceeding for those interested in agent systems (and the Semantic Web). One can wait for all the necessary tools and technologies to be ready to start developing and implementing agent systems (utilize ontological demarcation of resources), or one can start to do it now (using available, however imperfect, technologies and tools) – among others, to help develop new generation of improved tools and technologies. In our work we follow Nwana and Ndumu in believing that the latter approach is the right one. Therefore, we *do not* engage in the discussion *if* concept of a software agent is anything more but a new name for old ideas; *if* agents should be used in a travel support system; *if* agent mobility is or is not important, *if* JADE (JADE, 2005), JENA (JENA, 2005) Raccoon (Raccoon, 2005) are the best technologies to be used etc. Our goal is to use what we consider top of the line technologies and approaches to develop and implement a complete skeleton of an agent based travel support system that will utilize semantically demarcated data as its centerpiece.

Here an additional methodological comment is in order. As it was discussed in (Gilbert, 2004, Harrington, 2003, Wright, 2003) there exist two distinct ways of managing information in an *infomediary* (Galant 2002d) system like the one discussed here (with possible intermediate solutions). Information can be *indexed* – where only references to the actual information available in repositories residing outside of "the system" are stored. Or, information can be *gathered* – where actual content is brought to the central repository. In the original design of the travel support system (Angryk 2002, Gilbert, 2004, Harrington, 2003, Wright, 2003) we planned to follow the indexing path, which is more philosophically aligned with the main ideas behind the Semantic Web. It can be said metaphorically, that in the Semantic Web *everything is a resource* that is located somewhere within the Web and can be found through a generalized resource locator. In this case indexing simply links together resources of interest. Unfortunately, current state of the Semantic Web is such that there are practically no resources that system like ours could use. To be able to develop and implement a working system "now" we have

decided to gather information. More precisely, in the central repository we will store sets of RDF triples (*tokens*) that will represent travel objects (instances of ontologies). We will also develop an agent-based data collection system that will transform Web-available information into such tokens stored in the system.

Obviously, our work is not the only one in the field of applying agents and ontologies to travel support, however, while we follow many predecessors, we have noticed that most of them have ended on a road leading to nowhere. In our survey conducted in 2001 we have found a number of WWW sites of agent based travel support system projects that never made it beyond the initial stages of conceptualization (for more details see (Paprzyckii, 2001a, Paprzycki, 2001b) and references presented there). The situation did not change much since. A typical example of the state-of-the-art in the area is the, EU funded, CRUMPET project. During its funded existence (between approximately 1999 and 2003) it resulted in a number of publications and apparent demonstrations, but currently its original WWW site is gone and it is really difficult to assess which of its promises have been truly delivered on.

Summarizing, there exists a large number of sources of inspiration of our work, but we proceed with development of a system that constitutes a rather unique combination of agents and the Semantic web.

System description

Before we proceed describing the system let us stress that what we describe in this chapter is the core of a much larger system that is in various stages of development. In selecting the material to be presented we have decided first, to focus on these parts under development that are finished or almost finished. This means that a number of interesting agents that are to exist in the system in the future and that were proposed and discussed in (Angryk, 2002, Galant, 2002c, Gordon, 2005a) will be omitted. Furthermore we concentrate our attention on these parts of the system that are most pertinent to the subject area of this book (Semantic Web and eBusiness) while practically omitting issues like, for instance, agent–world communication (addressed in (Galant, 2002b, Kaczmarek, 2005)) and others.

In Figures 1 and 2 we present two distinct top level views on the system. The first one depicts basic "interactions" occurring in the system as well as its main subsystems. It also clearly places the repository of semantically demarcated data in the center of the system. More precisely, starting from right to left, we can see that *content* has been divided into (a) *Verified Content Providers* (*VCP*) that represents sources of trusted content that are consistently available and format of which is changing rarely and not "without a notice" and (b) *other sources* that represents all of the remaining available content. Interested readers can find more information about this distinction in (Angryk, 2002, Gordon, 2005a).

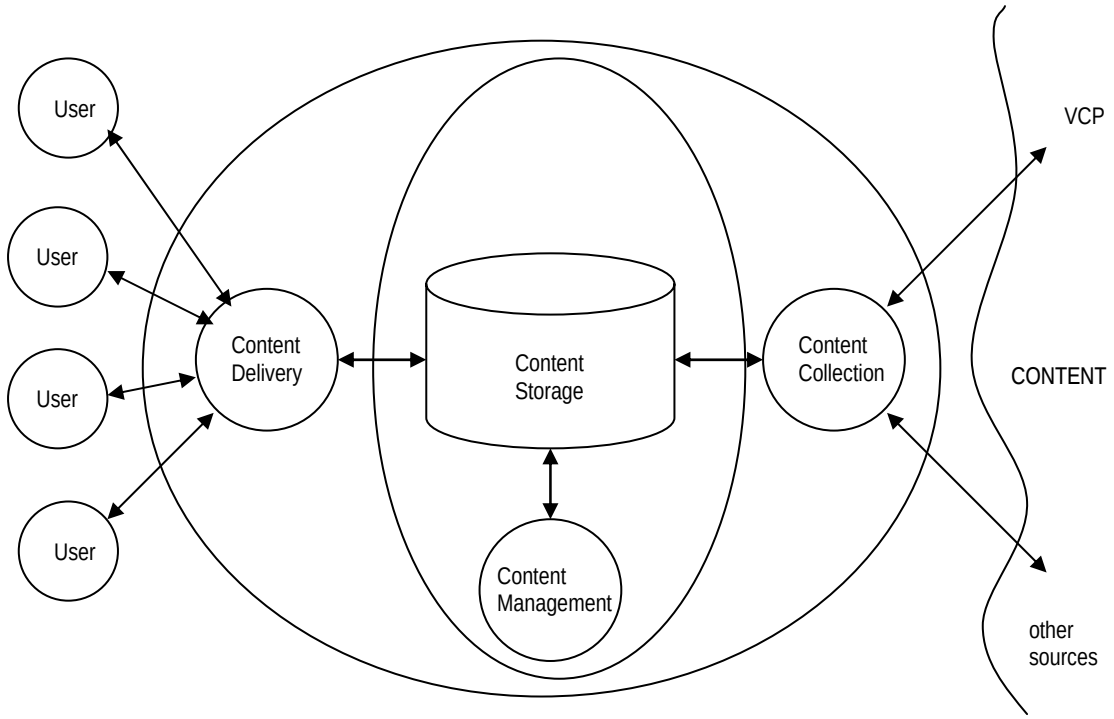

Figure 1. Top level view of the system.

While the dream of the Semantic Web is a beautiful one indeed, currently (outside of a multitude of academic research projects) it is almost impossible to find within the Web large sources of clean explicitly ontologically demarcated content (in particular, travel related content). This being the case, it is extremely difficult to find actual data that can be used (e.g. for testing purposes) in a system like the one we are developing. Obviously, we could use some of existing text processing techniques to classify pages as relevant to various travel-topics, but this is not what we attempt to achieve here. Therefore, we will, for the time being, omit the area denoted as *other sources* that contains mostly weakly structured and highly volatile data (see also (Nwana, 1999) for an interesting discussion of perils of dealing with dynamically changing data sources). This area will become a source of useful information when the ideas of the Semantic Web and ontological content demarcation become widespread.

Since we assume that *VCP*s carry content that is structured and rarely changing its format (e.g. website of Hilton Hotels), it is possible to extract from them information that can be transformed into a form that is to be stored in our system. More precisely, in our system, we store information about travel objects in the form of instances of ontologies, persisted in a JENA repository. To be able to do this, in the *Content Collection Subsystem* we use *Wrapper Agents* (*WA*) designed to interface with specific WWW sites and collect information available there (see also Figure 2). Note that currently we have no choice but to create each of *WA*s manually. However, in the future, as semantic demarcation becomes standard, the only operation required to adjust our system will be to replace our current "*static WAs*" with "*ontological WA*s". This is one of the important strengths of agent-based system design, pointed to in (Jennings, 2001, Woolridge, 2002).

As mentioned, the *Content Storage* is the JENA repository, which was designed to persist RDF triples (RDF is our semantic demarcation approach of choice). The *Content Management Subsystem* encompasses a number of agents (considered jointly as a *Data Management Agent* (*DMA*)) that work to assure that users of the system have access to the best quality of data. These agents are to, among others deal with: time sensitive information (such as changes of programs of movie theaters), incomplete data tokens or inconsistent information (Angryk, 2002, Gordon, 2005a).

Content Delivery Subsystem has two roles. First it is responsible for the format (and syntax) of interactions between users and the system. However, this aspect of the system, as well as agents responsible for it, is mostly outside of scope of this chapter (more details can be found in (Galant, 2002b, Kaczmarek, 2005)). Second, it is responsible for the semantics of user-system interactions. Here two agents play crucial role. First, the *Personalization Infrastructure Agent* (*PIA*) that consists of a number of extremely simple rule-based "RDF subagents" (each one of them is a class within the *PIA*) that extend the set of travel objects selected as a response to the original query to create a *Maximum Response Set* (*MRS*) that is delivered to the *PA* for filtering and ordering. Second, the *Personal Agent* (*PA*) that utilizes *user profile* to filter and hierarchically organize information obtained from the *PIA* as the *MRS*. It is also the *PA* that is involved in gathering explicit user feedback (see below) that is used to adjust user profile.

In Figure 2 we represent, in the form of an UML use case diagram, the above mentioned agents as well as other agents that are a part of the central system infrastructure. This diagram should be considered together with the system visualization found in Figure 1.

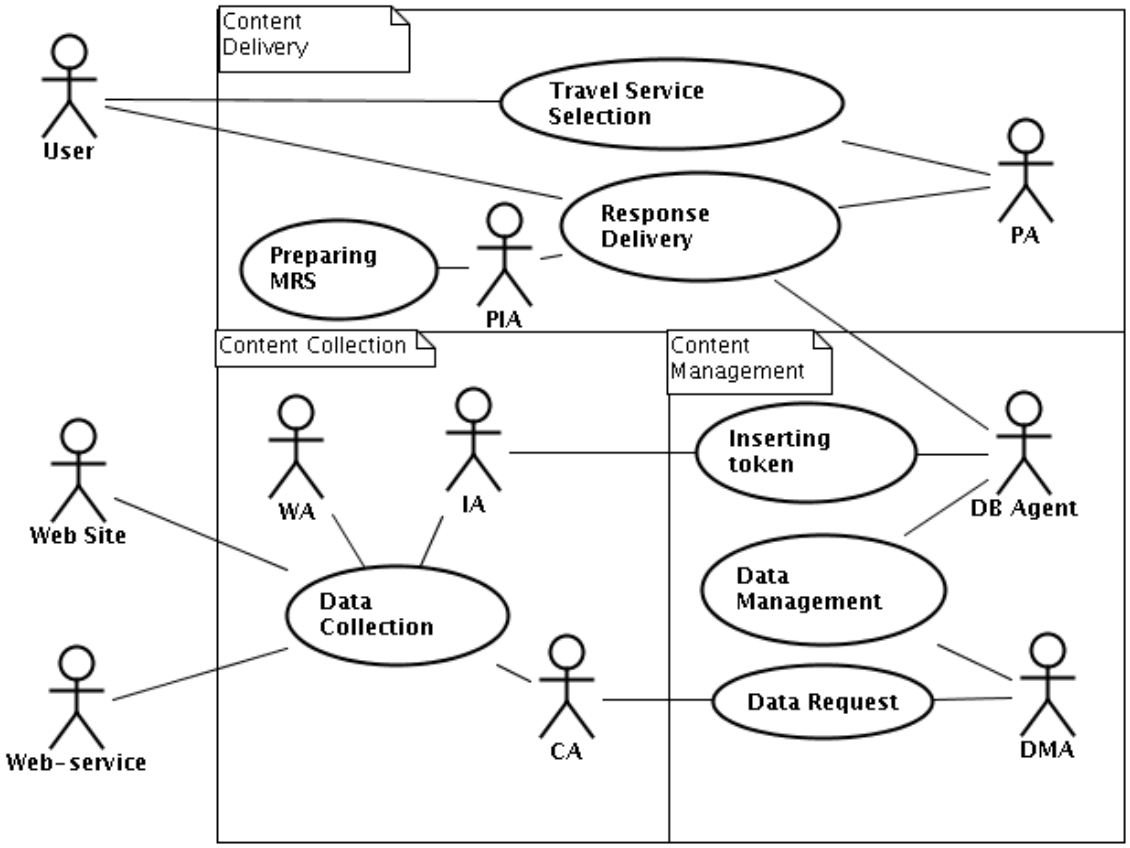

Figure 2. Top level use case diagram.

Since we had to abandon, hopefully temporarily, *other sources*, in Figure 2 we depict only Web Sites and Web Services that belong to the *VCP* category. They are sources of data for the function *Data Collection* that is serviced by *Wrapper Agents* (*WA*), *Indexing Agents* (*IA*) and a *Coordinator Agent* (*CA*). The *IA* communicates with the *DB Agent* when performing *Inserting tokens* function. Separately, the *CA* receives *Data request*s from *Data Management Agents* (*DMA*). These *Data request*s represent situations when data tokens were found to be potentially obsolete or incomplete (as a part of the *Data Management* function) and a new token has to be delivered by an appropriate *WA* to refresh / complete data available in the system. The *DMA* and the *DB Agent* (*DBA*) are the only agents that have a direct access to the JENA database. In the *Content Delivery Subsystem* we have three functions specified. The *Travel Service Selection* function is related to *User*(s) querying the system (information flow from the *User* to the central repository), while the *Response Delivery* function involves operations taking place between the time when the initial response to the query is obtained from JENA and when the final personalized response is delivered to the user (information flow from the central repository to the *User*). During this process the *PIA* performs the *Preparing MRS* function. Let us now discuss in some detail agents and their interactions. Before we proceed let us note that we omit a special situation when the system is initialized for the very first time and does not have any data stored in the JENA repository. While this situation requires agents started in a specific order, since it is only a one-time event it is

not worthy extra attention. We therefore assume that there is already data stored in the system and focus on interactions taking place in a working system.

Wrapper Agent (*WA*) interfaces with WWW sites, mapping XML- or HTML-demarcated data into RDF triples describing travel objects (according to the ontology used in our system (Gawinecki, 2005a, 2005b, Gordon, 2005b)). It is created by the *Coordinating Agent* (*CA*) on the basis of a *configuration file*. The configuration file may be created by System Administrator and send to the *CA* as a message from the *GUI Agent* or may be contained in a message from the *DMA* that wants to update one or more tokens. Each completed token is time-stamped and priority-stamped and send back to the *CA*. Upon completion of its work the (or in the case of an error) *WA* sends an appropriate message to the *CA* and self-destructs. A new *WA* with the same functionality is created by the *CA* whenever needed. Note that to simplify agent management we create instances of *WA* for each "job," even though they may produce tokens describing the same travel resource. For instance, when one *WA* is working on finding information about *all* Westin Hotels in Central Europe (task assigned by the System Administrator), another *WA* may be asked to find information about Westin Hotel in Warszawa (job requested by the *DMA*). It is the role of the *Indexing Agent* to assure that the most current available token is stored in the repository (see below). An UML statechart of the *WA* is contained in Figure 3.

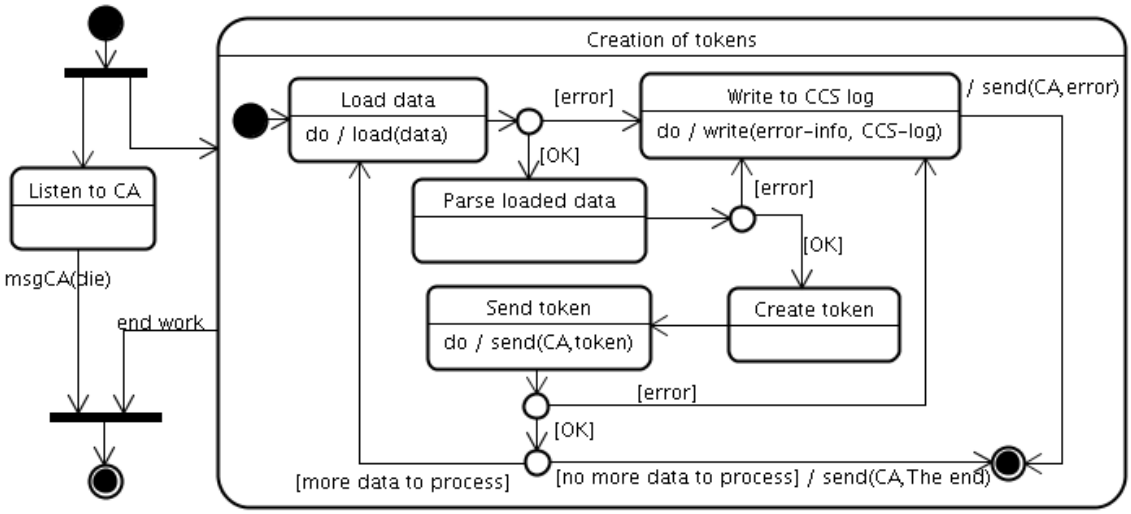

Figure 3. Statechart of the *Wrapper Agent.*

Coordinator Agent (*CA*) manages all activities of the *Content Collection Subsystem.* When started, it creates a certain number of *Indexing Agents* (*IA*) (specified by the System Administrator – *Servicing agent management request* function in Figure 4) and enters a listening state. There are six types of messages that may be received. (1) A self-destruction order received from the *GUI Agent* (send by the System Administrator) – resulting in the *CA* killing all existing *WA*s and *IA*s first, and then self-destructing. (2) Message from the *WA* that it encountered an error or that it has completed its work and will self-destruct – resulting in appropriate information being recorded. (3) Message from the *WA* containing a token – to be inserted into the priority queue within the *CA*. (4)

Message from one of the *Indexing Agent*s (*IA*) requesting a new token to be inserted into the repository – which results in the highest priority token being removed from the priority queue and send to the requesting *IA*. When the queue is empty, a message is send to the *IA* informing about this fact (as seen in Figure 5, *IA* will retry requesting token after some delay). (5) Message from the *DMA* containing a request (in the form of a configuration file) to provide one or more tokens – resulting in creation of an appropriate *WA* (or a number of *WA*s). And, finally, (6) message from the *GUI Agent* ordering adjustment of the number of *IA*s in the system. A complete statechart of the *Coordinator Agent* is depicted in Figure 4.

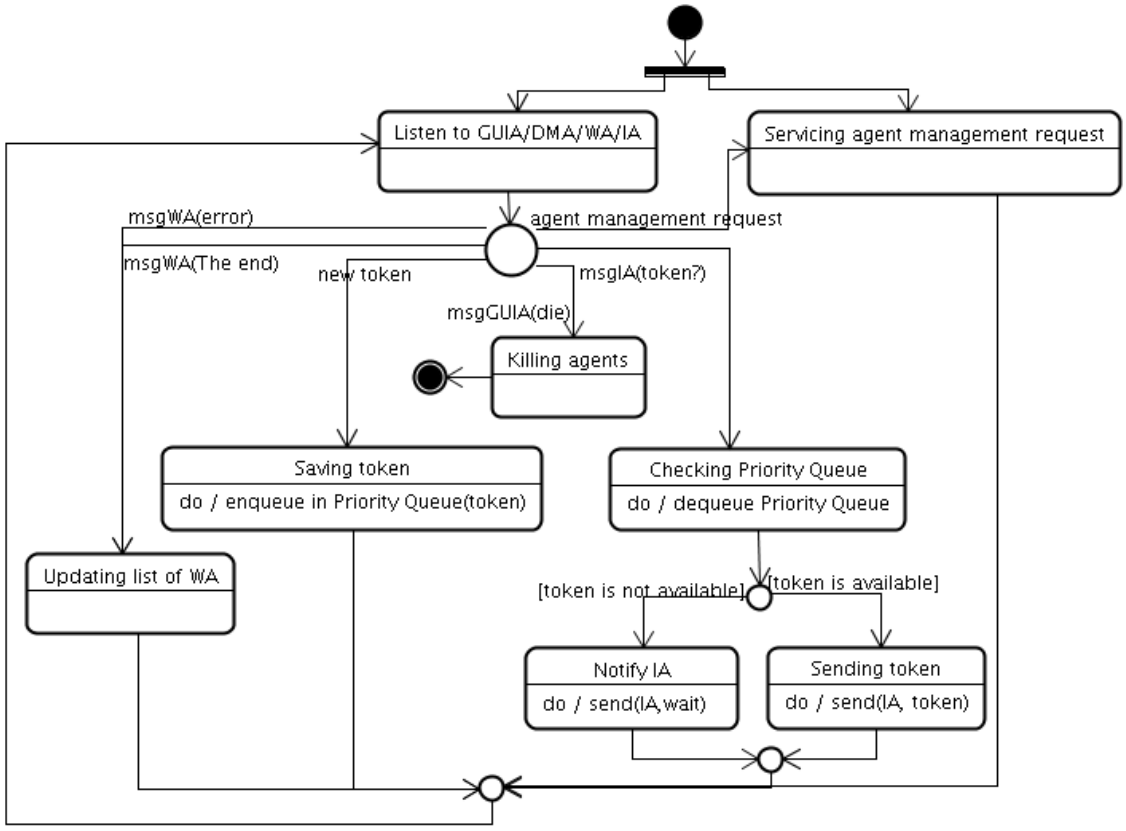

Figure 4. Statechart of the *Coordinator Agent*

Indexing Agent (*IA*) is responsible for inserting tokens into the central repository as well as initial pre-processing of tokens to facilitate cleanness of data stored in the system. For the time being the *IA* performs the following simple checks: (1) time consistency of tokens to be inserted – since it is possible that multiple *WA*s generate token describing the same travel resource (see above), the *IA* compares time stamps of the token to be inserted with that in the repository and inserts its token only when it is newer; (2) data consistency – token to be used to update / append information has to be consistent with the token in the repository (e.g. the same hotel has to have the same address); inconsistent tokens are marked as such and they are to be deconflicted (Angryk, 2002). In the case when the priority queue is empty, request will be repeated after delay *T*. The statechart of the *IA* is represented in Figure 5 (top panel presents the overall process flow, while the bottom panel specifies processes involved in servicing tokens).

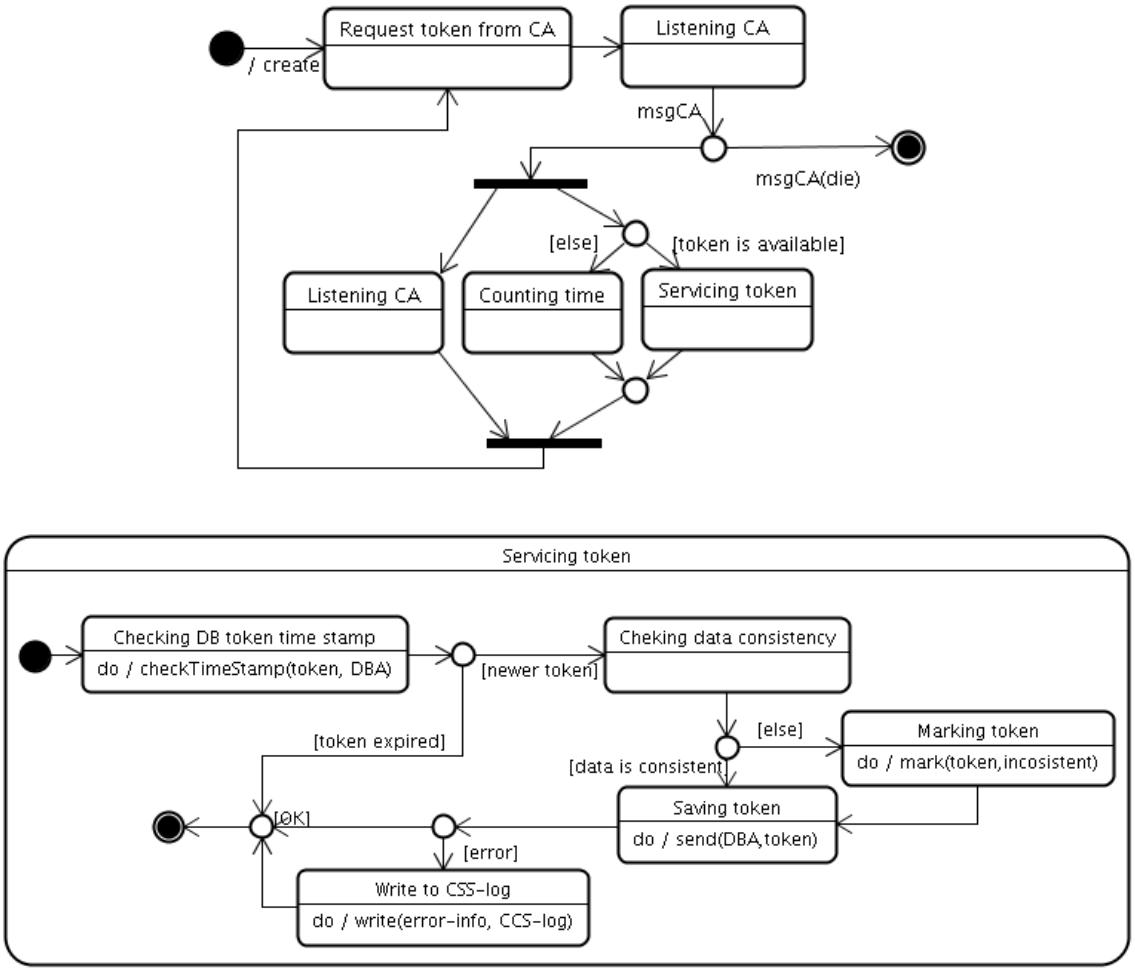

Figure 5. Statechart of the *Indexing Agent*

Let us now briefly describe the next three agents visible in Figure 2. The *DB Agent* (*DBA*) represents interface between the database (in our case the JENA repository) and the agent system. It is created to separate an agent system from an "outside technology" in such a way that in case of changes in the repository all other changes will be localized to that agent, while the remaining parts of the system stay unchanged.

In the current system the *Data Management Agent* (*DMA*) is a simple one. A number of agents of this type, responsible for different travel objects, are created upon system startup. Their role is to "traverse" the repository to find outdated and incomplete tokens and request new / additional ones to be generated to update / complete information stored in the repository. To achieve this goal *DMA*s generate a configuration file of an appropriate *WA* and send them to the *CA* for processing. In the future *DMA*s will be responsible for complete management of tokens stored in the repository to assure their completeness, consistency and freshness.

The *Personalization Infrastructure Agent* (*PIA*) consists a *manager* and a number of "RDF subagents" (*PIA workers* in Figure 6). Each of these sub-agents represents one or

more of simple rules of the type "Irish pub is also a pub" or "Japanese food is Oriental food." These rules are applied to the set of RDF triples returned by the initial query. Rule application involves querying the repository and is expected to expand the result set (e.g. if user is asking for Korean restaurant then other Oriental restaurants are likely to be included). The *PIA* subagents operate as a team passing the result set from one to the next (in our current implementation they are organized in a ring) and since their role is to maximize the set of responses to be delivered to the user no potential response is removed form the set. Final result of their operation is the *Maximal Response Set* (*MRS*) that is operated on by the *Personal Agent.* Action Diagram of the *PIA* is depicted in figure 6.

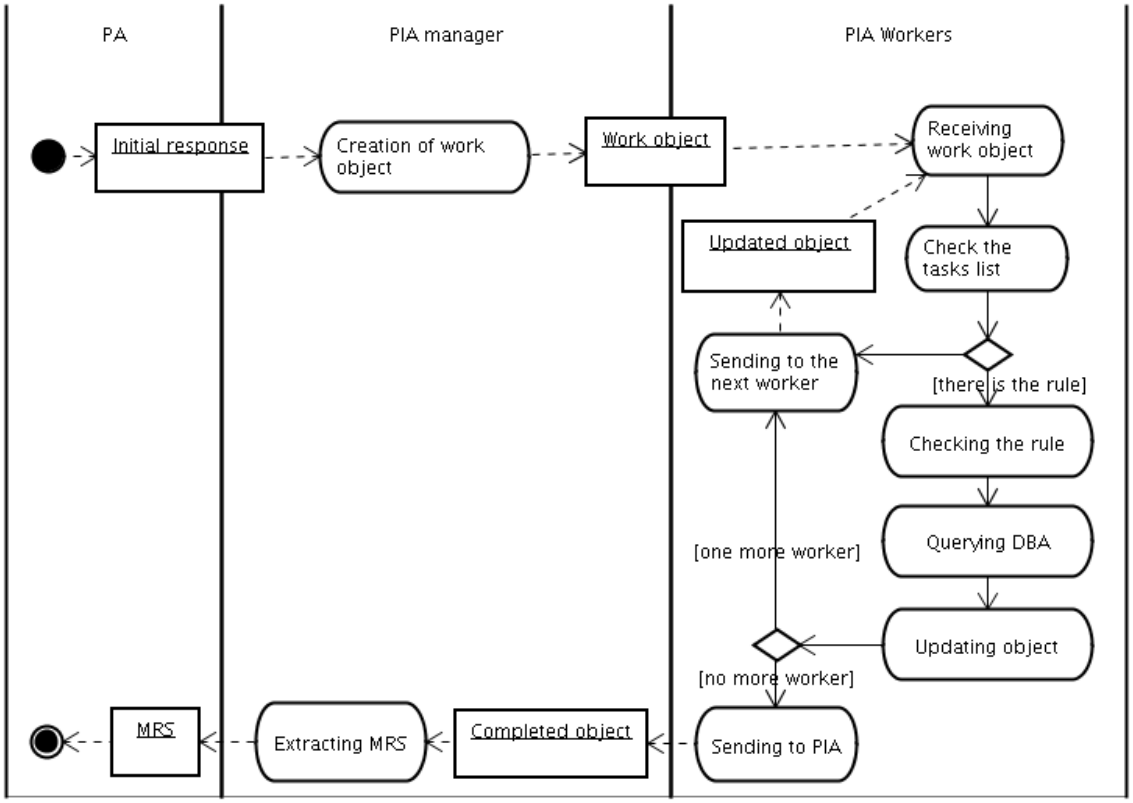

Figure 6. Action Diagram of the *PIA*

A separate *Personal Agent* (*PA*) will be created for each user and will play two roles in the *Content Delivery Subsystem.* First, it is the central coordinator – for each user query it directs it from one agent to the next, constantly monitoring processing progress. Second, it utilizes *user profile* to filter and order responses that are to be sent to the user. More precisely, the user query, after being pre-processed and transformed into an RDQL query (see (Kaczmarek, 2005) for more details) is being sent to the *DBA.* What is returned is the *initial response* consisting of a number of tokens that satisfy the query. This response is being redirected (by the *PA*) to the *PIA* to obtain the *MRS.* Then the *PA* utilizes the user profile to: (1) remove from the set responses that do not belong there (e.g. user is known to be adversely inclined toward Italian food, and pizza in particular, and thus all of the Italian food serving restaurants have to be excluded); (2) order the remaining selections

in such a way that these that are believed to be of most interest to the user will be displayed first (e.g. if user is known to stay in Hilton Hotels, they will be displayed first). The statechart diagram of the *PA* is contained in Figure 7.

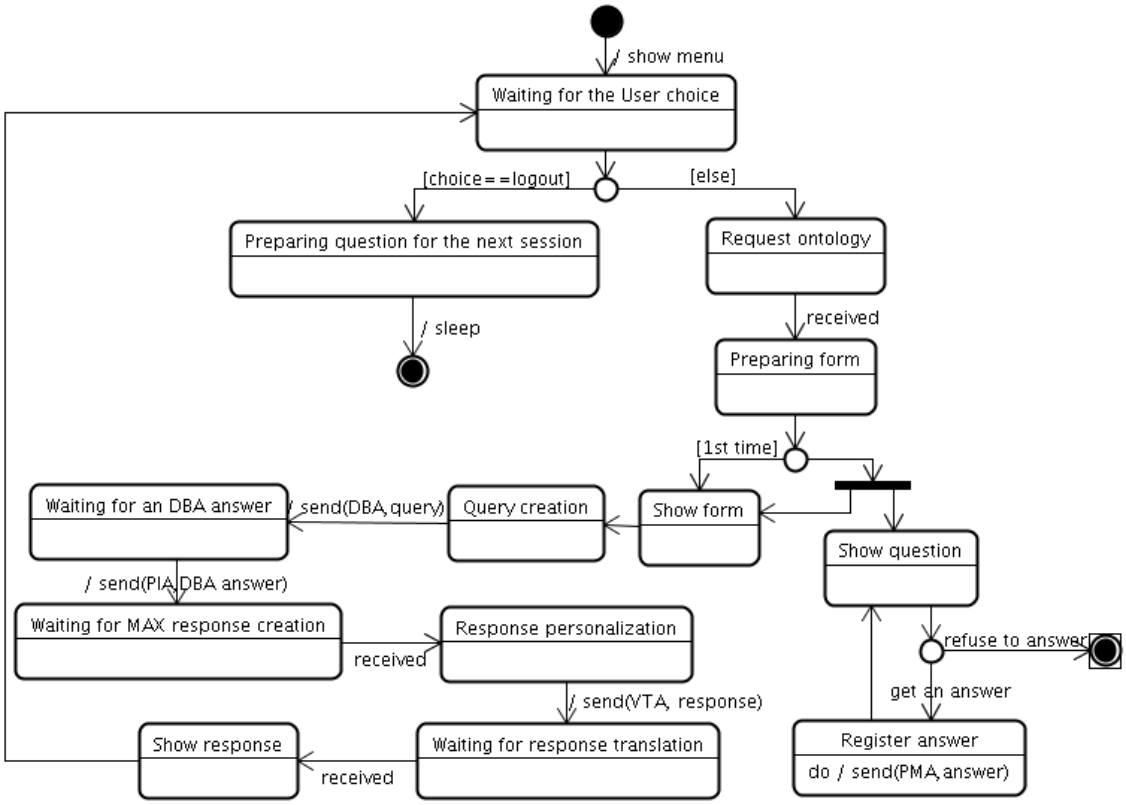

Figure 7. Statechart of the *Personal Agent*

As we can see the *PA* behaves differently depending if the user is using the system for the first time or if it is a returning user. In the latter case, the *PA* will attempt at gathering explicit feedback related to the information delivered to the user during the previous session. This will be done through generation of a questionnaire that will be shown to the user, who may decide to ignore it (see also (Galant, 2002a)). Obtained responses will be used to adjust the user-profile. We can also see how the *PA* plays the role of response preparation orchestrator by always receiving responses from other agents and forwarding them to the next agent in the processing chain. We have selected this model of information processing so that "worker agents" like the *DBA* or the *PIA* know only one agent to interact with (the *PA*). Otherwise, an unnecessary set of dependencies would be introduced to the system making it substantially more difficult to manage (any change to one of these agents would have to be propagated to all agents that interact with it – while in our case only a single agent needs to be adjusted).

Replacing Semantic Web with a Semantic Database

As noted before, currently the Semantic Web is an attractive idea that lacks its main component – large repositories of semantically demarcated (in particular travel-related) data. This was one of important reasons to change the design of our systems from data indexing into data gathering. As a result we are able to create our own "mini Semantic Web" (in the form of a semantic database) and store there information that in the future will allow us to extend our system beyond the basic skeleton described here, and start experimenting with its true projected functionalities – like content personalization.

Let us describe how the HTML-demarcated information available on the Web is turned into semantic-tokens representing travel objects in our repository. Before proceeding let us discuss briefly ontologies utilized in the system. As reported in (Gawinecki, 2005a, 2005b, Gordon, 2005b) while there exists a large number of attempts at designing ontologies depicting various aspects of the world, we were not able to locate a complete ontology of most basic objects in the "world of travel" such as a *hotel* and a *restaurant.* More precisely, there exists an implicit ontology of restaurants utilized by the ChefMoz project (ChefMoz, 2005), but it cannot be used directly as a Semantic Web resource due to the fact that data stored there is infested with bugs that make its automatic utilization impossible without pre-processing that also involves manual operations (see (Gawinecki, 2005a, Gordon, 2005b) for more details).

This being the case we have proceeded in two directions. First, as reported in (Gawinecki, 2005a, Gordon, 2005b) we have reverse engineered the restaurant ontology underlying the ChefMoz project and cleaned data related to Polish restaurants. Separately we have proceeded with designing hotel ontology using a pragmatic approach. Our hotel ontology is to be used to represent, manipulate and manage hotel information actually appearing within Web-based repositories (in context of travel; i.e. not hotels as landmarks, or sites of historical events). Therefore we have studied content of 10 largest Internet travel agencies and found out that most of them describe hotels using very similar "vocabulary." Therefore we used these common terms to shape our hotel ontology and the results of this process have been reported in (Gawinecki, 2005a, 2005b, Gordon, 2005b). As an outcome we have two fully functional, complete ontologies (of a hotel and of a restaurant) that are used to shape data stored in our JENA repository.

In this context, let us illustrate how we transform the *VCP* featured data into travel tokens. As an example we will utilize WWW site belonging to Hilton Hotels (www.hilton.com). More precisely, let us look at some of the information that is available at the WWW site of Hilton Sao Paulo Morumbi depicted in Figure 8.

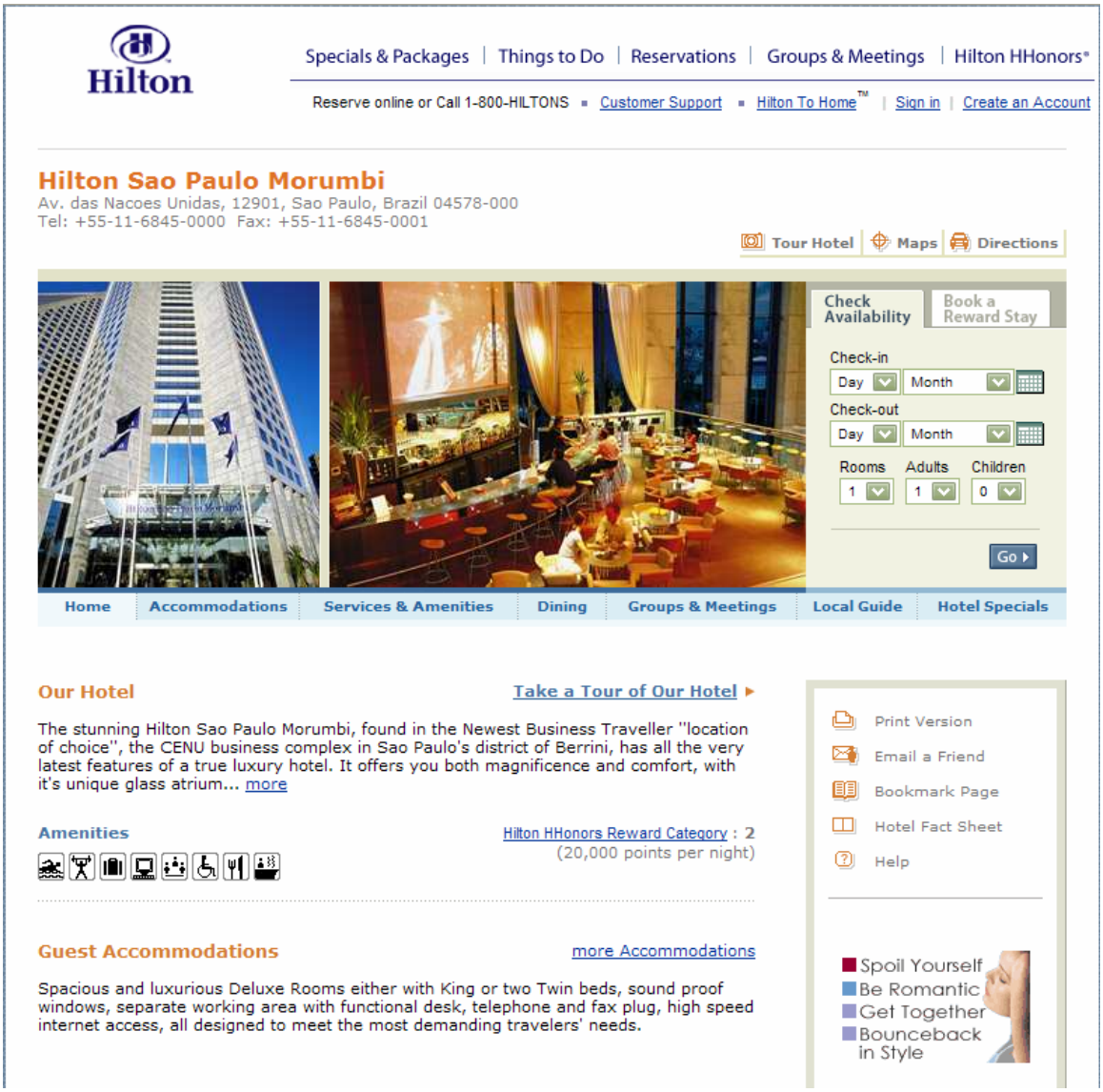

Figure 8. Hilton Sao Paulo Morumbi main page.

As clearly seen, from this page we can extract information such as the hotel name, address and phone numbers. This page would also have to be interacted with in case we planned to utilize our travel support system to make an actual reservation (which is only in very long-term plans and out of scope of this chapter). To find the remaining information defined by the hotel ontology requires traversing the WWW site deeper. Therefore, for instance, the *WA* has to go to the page contained in Figure 9, to find information about hotel amenities.

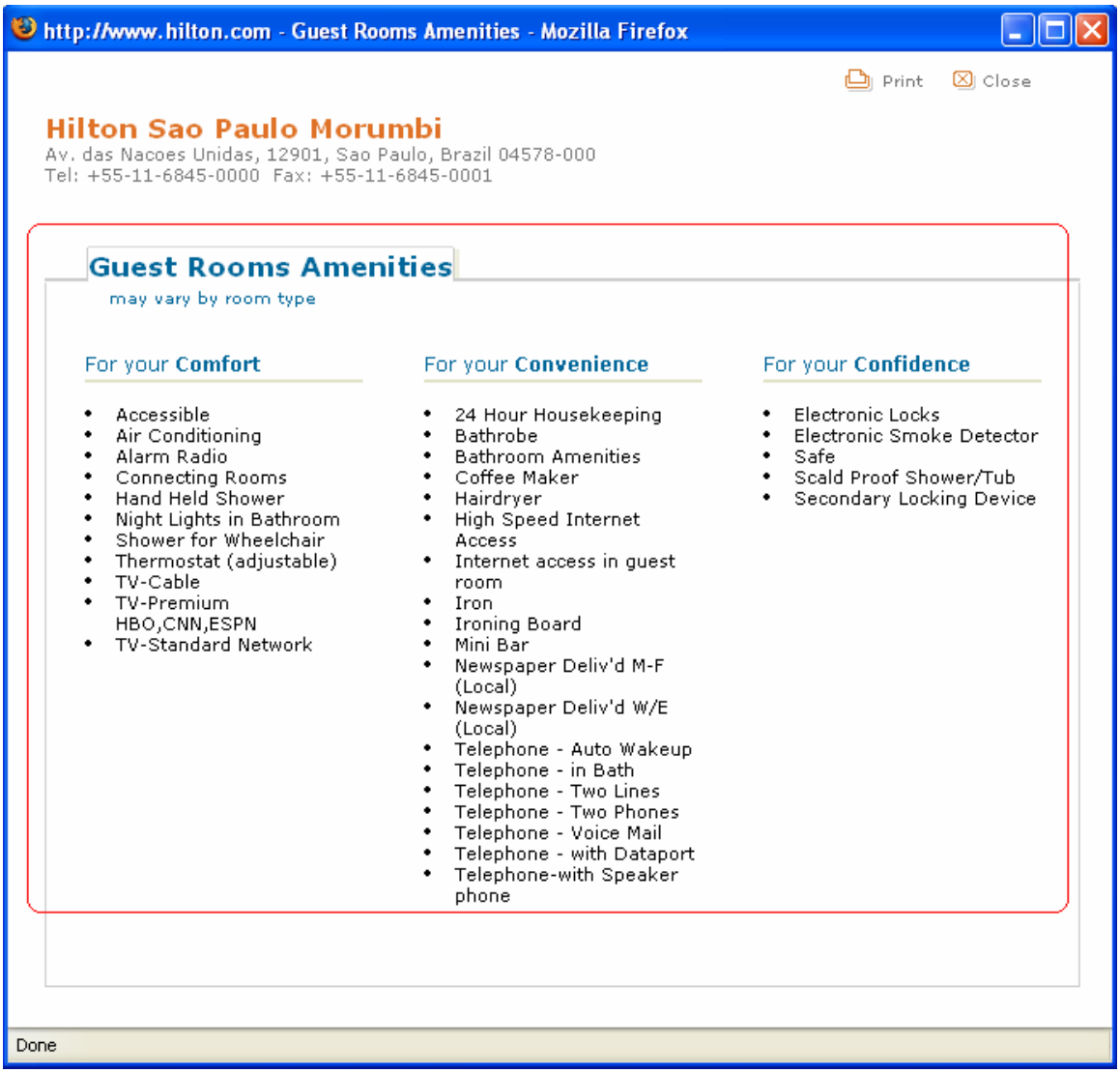

Figure 9. Hilton Sao Paulo Morumbi amenities page.

As a result the following set of RDF triples (in XML-based notation) will be generated:

```
<rdf:Description
       rdf:about="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI">
  <j.1:roomAmenit rdf:resource="http://.../hotel.rdf#AccessibleRoom"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#AirConditioning"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#ConnectingRooms"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Shower"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#CableTelevision"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#CNNavailable"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Bathrobe"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#BathroomAmenities"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Coffee_TeaMaker"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Hairdryer"/>
  <j.1:roomAmenity
       rdf:resource="http://.../hotel.rdf#HighSpeedInternetConnection"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#InternetAccess"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Iron"/>
```

```
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#IroningBoard"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Minibar"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Newspaper"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Wake-upCalls"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Two-linePhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#VoiceMail"/>
  <j.1:roomAmenity
       rdf:resource="http://.../hotel.rdf#TelephoneWithDataPorts"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#SpeakerPhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#SmokeDetektors"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Safe"/>
</rdf:Description>
```

These RDF triples represent a part of our hotel ontology, but this time they became its instance representing a given Hilton Hotel (values of various aspects of the hotel are filled-in). Our *WA* will then continue traversing the hotel site to find, for instance, information about fitness and recreation as well as check-in and check-out times. An appropriate page belonging to the same hotel is depicted in Figure 10 while the resulting set of RDF triples follows.

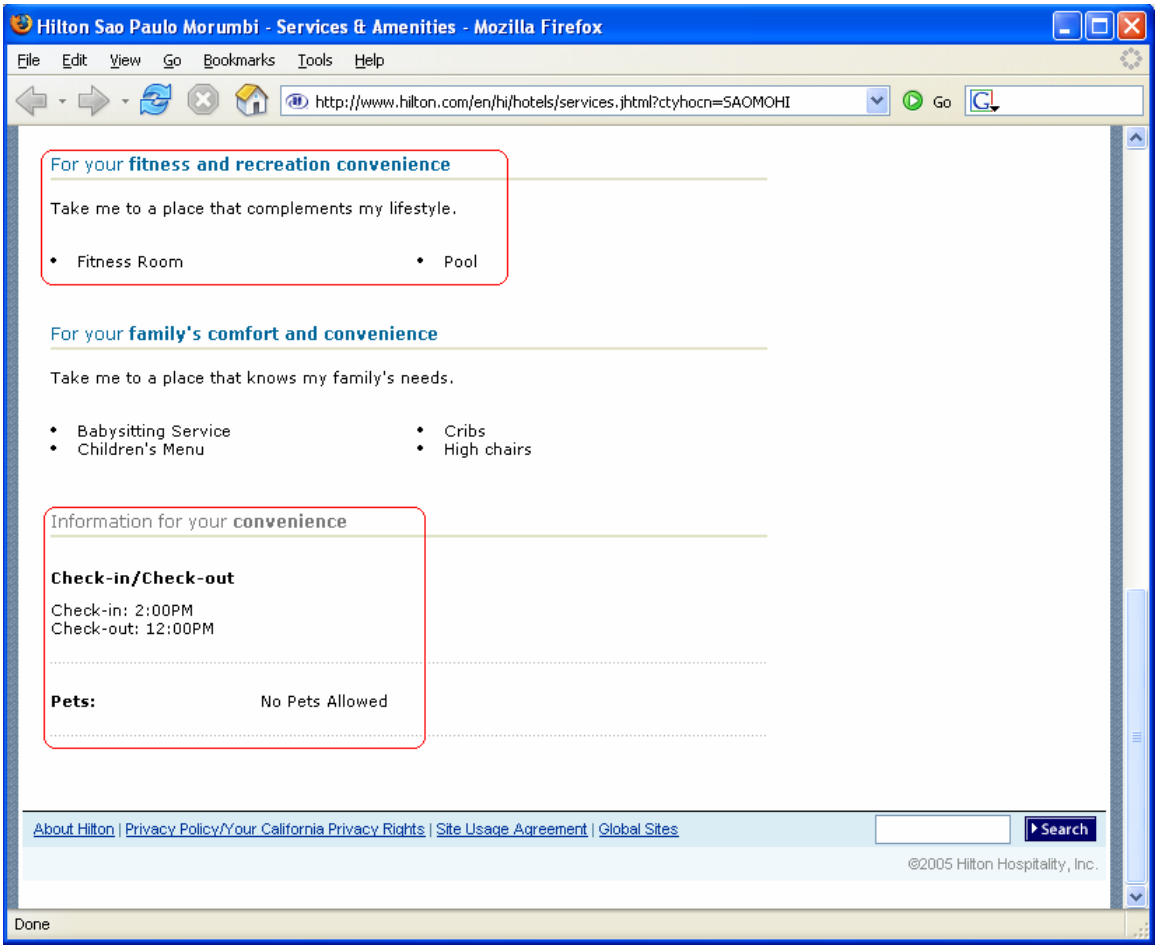

Figure 10. Hilton Sao Paulo Morumbi fitness and recreation and check-in & check-out information.

```
<rdf:Description
       rdf:about="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI">
  <j.1:recreationService
       rdf:resource="http://.../hotel.rdf#FitnessCenterOnsite"/>
  <j.1:recreationService
       rdf:resource="http://.../hotel.rdf#IndoorOrOutdoorConnectingPool"/>
  <j.1:petsPolicy rdf:resource="http://.../hotel.rdf#NoPetsAlowed"/>
  <j.1:additionalDetail
       rdf:resource="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI/CheckI
       n-CheckOut"/>
</rdf:Description>

<rdf:Description
       rdf:about="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI/CheckIn-
       CheckOut">
  <j.1:detail>Check-in: 2:00PM, Check-out: 12:00PM</j.1:detail>
</rdf:Description>
```

In this way the *WA* processes all necessary pages belonging to the Hilton Sao Paulo Morumbi and as a result obtains a set of RDF triples that constitute its complete definition (from the point of view of ontology utilized in our system). This set of RDF triples is then time and priority level stamped, packed into an ACL message and send to the *CA* that inserts it into the priority queue – to be later inserted, by the *IA,* to the semantic database. Depending on the assignment the *WA* may continue producing tokens of other Hilton hotels or, if work is completed, it informs the *CA* about this fact and self-destructs. In this way in our system, by manually creating *Wrapper Agents* for a variety of travel information sources we can collect real-life data representing actual travel objects.

RDF data Utilization – Content Personalization

Let us now discuss how the data stored in the system is used to deliver personalized responses to the user. While our approach to user profile construction and utilization is based on ideas presented in (Burke, 2002, Fink, 2002, Galant, 2002a, Kobsa, 2001, Montaner, 2003, Rich, 1979, Sołtysiak, 1998), however utilization of these methods in the context of ontologically demarcated information is novel and was proposed originally in (Gawinecki, 2005c).

To be able to deliver personalized content to the user of the system, we have to be able to represent the user in the system first – define user-profile. Furthermore, the proposed user-profile has to be created in such a way to simplify interactions in the system. Since our system is oriented toward processing of ontologically demarcated data, it is very natural to represent user preferences in the same way. Thus we adapted an *overlay model* of user profile, where opinions are “connected” with appropriate concepts in the domain ontology. This approach is also called a *student model*, since it has been found useful to describe knowledge of the student about specific topics of the domain (Greer, 1994). Basic tenets of the overlay model are depicted in Figure 11.

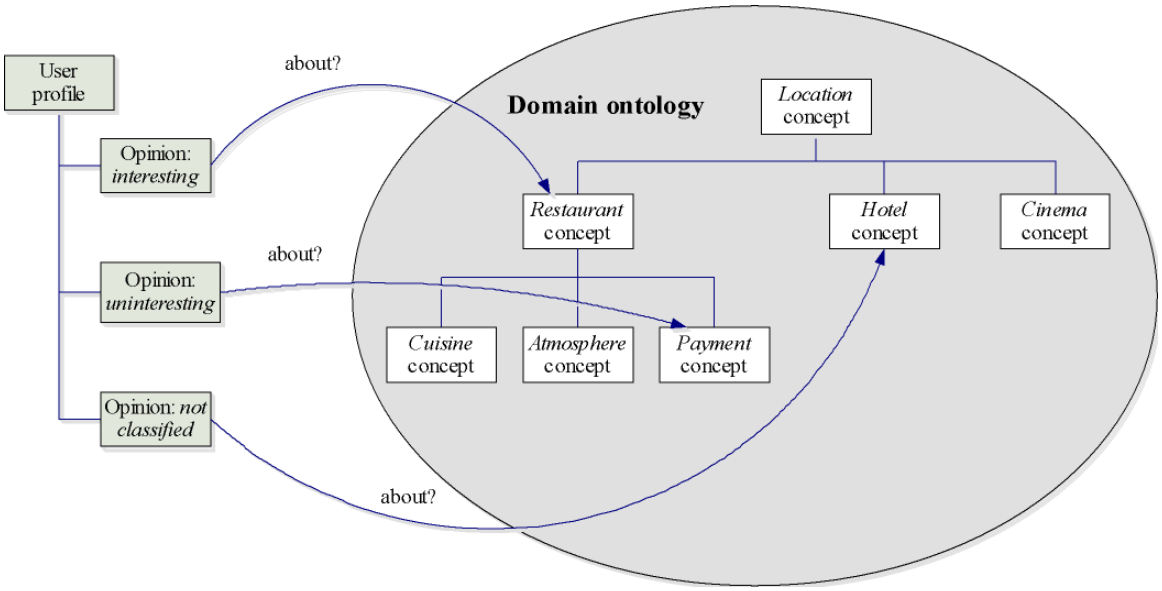

Figure 11. Overlay model utilized to represent user profile

For instance, let us consider our hotel ontology and assume that user likes to stay in hotels that have both *pool* and *fitness center*. Both these features are subclasses of the concept *amenities*. We can represent user interest by assigning weight to each amenity (the larger the weight the more important is given feature to the user). In case of our hypothetical customer, both pool and exercise room will be assigned large weights, while features that user is not particularly interested in (e.g. availability of ironing board – see Figure 9) will be assigned small weight – the lesser is the interest the closer to 0 the value will be. In the case of features about which we do not know anything about users' preferences, no value will be assigned (see Figure 11). Let us observe that in this approach we are mimicking the notion of probability – all assigned values are from the interval (0, 1). This means that even in the case of strong counter-preference towards a given feature we will assign value 0 (there are no negative values available). Proceeding in thus described way we will create a special instance of hotel ontology, one that represents *user-hotel-profile*. The following fragment of an instance of hotel ontology (this time represented in an N3 notation) depicts user (Karol) profile as it is represented in our system:

```
:KarolOpinions    a sys:OpinionsSet;
  sys:containsOpinion
    [sys:about hotel:Pool;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.89].
    [sys:about hotel:ExerciseRoom;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.84].
    [sys:about res:AirConditioning;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.89].
    [sys:about hotel:BathroomAmenities;
      sys:hasClassification sys:Interesting;
```

```
      sys:hasNormalizedProbability 0.73].
    [sys:about hotel:IroningBoard;
      sys:hasClassification sys:NotInteresting;
      sys:hasNormalizedProbability 0.11].
    [sys:about hotel:Iron,
      sys:hasClassification sys:NotInteresting;
      sys:hasNormalizedProbability 0.15].
```

The above *hotel-profile* of Karol, tells us that he likes to stay in hotels with swimming pool and exercise room, while availability of an iron and ironing board is inconsequential to him.

Obviously, somewhere in the system we have to store, in some form, information about the user. To assure consistency across the system, this is done in a form of a simplistic *user-ontology*. Below we present a fragment of such ontology:

```
: wasBorn a rdf : Property ;
      rdfs: range xsd : Date ;
      rdfs: domain : UserProfileData.

: hasAge a rdf : Property ;
      rdfs: range : Age ;
      rdfs: domain : UserProfileData .

: hasWealth a rdf : Property ;
      rdfs: range : Wealth ;
      rdfs: domain : UserProfileData .

: hasProfession a rdf : Property ;
      rdfs: range : Profession ;
      rdfs: domain : UserProfileData .
```

Let us now assume that Karol is a 24 years old painter, who has enough money to feel rich and whose dressing style is a natural one, than his profile would be represented as:

```
:KarolProfile a sys:UserProfile;
  :hasUserID          14-32-61-3456;
  :hasUserProfileData :KarolProfileData;
  :hasOpinionsSet     :KarolOpinions.

:KarolProfileData a :UserProfileData;
  :hasAge             24;
  :hasWealth          sys:Rich;
  :hasDress           sys:NaturalDress;
  :hasProfession      sys:SpecialistFreeLancer.
```

Rather than keeping them separate, we combine instances of user ontology with the above described user-profile into a complete ontological description – a comprehensive user-profile. This user-profile is then to be stored in the JENA repository.

One of the important questions is that all recommender systems have to address is, how to "introduce" new user to the system (Galant, 2002a). In our system we use stereotyping (Rich, 1979). Obviously, we represent stereotypes the same way we used to represent

user-profiles, with the difference that instead of specific values representing preferences of a given user, we use sets of variables of nominal (to represent categories – e.g. profession), ordinal (e.g. low income, medium income, high income) and interval (e.g. age between16 and 22) types. For values of nominal and ordinal types we have established sets of possible values, while for the values of interval types, we defined borders of intervals considered in the system. Using results of a survey and expert knowledge, we were able to create restaurant-related stereotypes (one instance of restaurant ontology of each identified stereotype). To illustrate such a case, here is a fragment of artistic profile in the area of restaurants:

```
:ArtistStereotypeOpinions    a sys:OpinionsSet;
  sys:containsOpinion
    [sys:about res:CafeCoffeeShopCuisine;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 1.0].
    [sys:about res:CafeteriaCuisine;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.75].
    [sys:about res:TeaHouseCuisine;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.9].
    [sys:about res:WineBeer;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.8].
    [sys:about res:WineList;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 1.0].
    [sys:about res:HotDogsCuisine;
      sys:hasClassification sys:NotInteresting;
      sys:hasNormalizedProbability 0.0].
```

In this stereotype we can see, among others, that an *artist* has been conceptualized as a person who likes *coffee houses* a bit more than *tea houses* and is willing to eat in a *cafeteria*, likes *vine* (a bit more than *beer*), but does not like *hot dogs* (*fast food*). Other stereotypes have been conceptualized similarly and their complete list, and a detailed description of their utilization can be found in (Gawinecki, 2005d).

When a new user logs to the system she will be requested to fill a short questionnaire about age, gender, income level, occupation, address (matching user features defined by the user-ontology), as well as questions about her travel preferences. While the basic user-ontology based data will be required, answering questions about travel preferences will be voluntary. Personal data collected through the questionnaire will be used to match a person to a stereotype. More precisely, we will calculate a distance measure between user-specified characteristics and these appearing in stereotypes defined in the system and find one that matches her profile the closest. To achieve this we will use the following formula:

$$d(\hat{S},\hat{u}) = \frac{\sum_{f=1}^{k} w^f \delta^f_{\hat{S}\hat{u}} d^f_{\hat{S}\hat{u}}}{\sum_{f=1}^{k} w^f \delta^f_{\hat{S}\hat{u}}}$$

Where: w^f – weight of attribute, $d^f_{S,u}$ – distance between values of the attribute in the stereotype S and user's data u, $\delta^f_{S,u}$ – Boolean flag that informs whether attribute f appears in both: stereotype's data (S) and user's data (u).

To illustrate this, let us consider Karol, the painter, again. In the table below we present Karol's data and the *artist* stereotype data and show how the closeness between Karol and that stereotype is calculated.

Attribute (f)	Attribute weight (w^f)	Data of artist stereotype (comma means OR relation): (S)	Karol's Data: (u)	Distance between value of attribute: ($d^f_{S,u}$)	Weighted distance: ($w^{f*} d^f_{S,u}$)
Age	2	20-50	24	0.00	0.00
Wealth	4	Not Rich, Average Rich	Rich	0.33	1.33
Dress	1	Naturally, Elegantly	Naturally	0.00	0.00
Profession	2	Student/Pupil, Scientist/Teacher, Specialist/FreeLancer Unemployed/WorkSeeker	Specialist/FreeLancer	0.00	0.00
			COMBINED		1.3(3) / (2+4+1+2)= 0.14(6)

The same process is then repeated comparing Karol's data against all other stereotypes to find the one that fits him the best. In the next step this stereotype is joined with his user-data to become his *initial profile*. In the case when he answered any domain-specific questions (recall, that he may omit them), this data will be used to modify his user-profile. For example, let us assume that he has been identified as *student stereotype*, but he has also specified that he does not like coffee houses (while in the *student stereotype* coffee houses have been assigned a substantial positive weight). Obviously, in *his* profile, this positive value will be replaced by zero – as explicit personal preferences outweigh these specified in the stereotype (see also (Nistor, 2002)):

```
:KarolOpinions a sys:OpinionsSet;
  sys:containsOpinion
    [sys:about res:CafeCoffeeShopCuisine;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.0].
```

Observe that as soon as the system is operational we will be able to store information about user behaviors (Angryk 2003, Galant, 2002a, Gordon, 2005). This data will be then used not only to modify individual user-profiles, but also mined (e.g. clustered) to obtain information about various group-behaviors taking place in the system. This information can be used to verify, update or completely replace our initial stereotypes. Such processes

are based on the so-called implicit relevance feedback (Fink, 2002, Kobsa 2001). As described earlier (see Figure 7) we will utilize also explicit feedback based on user responses to subsequent questionnaires. Currently as explicit feedback we utilize only a single question: "did you like our main suggestion presented last time?" but a more intricate questionnaire could also be used. Specifically, at the end of each user-system interaction, on the basis of what was recommended to the user, a set of questions about these recommendations could be prepared. When user returns to the system, these questions would be then asked to give him opportunity to express his direct opinion. Both implicit and explicit feedbacks are used to adjust user-profile (see also Gawinecki, 2005c). Note here, that in most recommender systems stereotyping is *the* method of information filtering (demographic filtering); thus making such systems rather rigid – in this case individual user preferences cannot be properly modeled and modified (Kobsa, 2001). In our system we use stereotyping only to solve the cold-start problem – and modify them over time – and thus avoid the rigidity trap.

User-profile is utilized by the *PA* to rank and filter travel objects. Let us assume that after the query, the response preparation process has passed all stages and in the last one the *PIA* agent has completed its work and the *MRS* has been delivered to the *PA*. The *PA* has now to compute a *temperature* of each travel object that is included in the *MRS.* The temperature represents the "probability" that a given object is a "favorite" of the user. This way of calculating the importance of selected objects was one of the reasons for the way that we have assigned importance measures to individual features (as belonging to the interval [0,1]). Recall here that the *DBA* and the *PIA* know nothing about user-preferences and that the *PIA* uses variety of general rules to increase the response set beyond that provided as a response to the original query.

To calculate the temperature of a travel object (let us name it an *active object*) three aspects of the situation have to be taken into account. First, features of the active object. Second, user interests represented in the user-profile – if a given feature has no preference specified then it cannot be used. In other words, for each token in the *MRS* we will crop its ontological graph to represent only these features that are defined in user profile. Third, features requested in user query. More specifically, if given keywords appear in the query (representing explicit wishes of the user), e.g. if the query was about a restaurant in Las Vegas, then such restaurants should be presented to the user first. Interactions between these three aspects are represented in Figure 12.

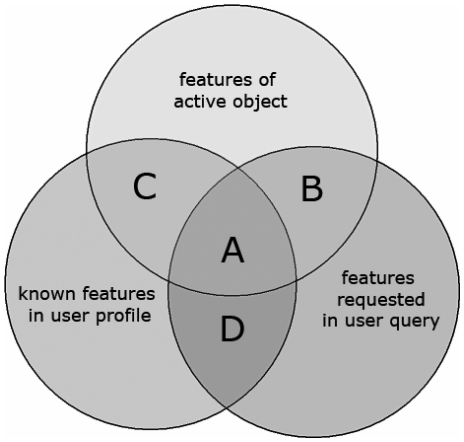

Figure 12. Construction of final response – interactions between features.

Here we can distinguish the following situations:

A – features explicitly requested by the user, that appear in the active object as well as in the user-profile,
B – features requested by the user and appearing in the active object,
C – features not requested, that are a part of user-profile and that appeared in the active object,
D – features that do not appear in the active object (we are not interested in them).

Ratings obtained for each token in the *MRS* represent what the system believes are user preferences and are used to filter out these objects temperatures of which are below a certain threshold and rank the remaining ones (objects with highest scores will be displayed first). We will omit discussion of a special case when there is no object above the threshold. The *MRS* is processed in the following way:

1. Travel objects are to be returned to the user in two groups (buckets)
   a. Objects requested explicitly by the user (via the query form) – Group I
   b. Objects not requested explicitly by the user but predicted by the system to be of potential interest to the user – Group II

   Thus, for each *active object* we divide features according to the areas depicted in Figure 11. Objects for which at least one feature is inside of either area A or B belong to Group I, objects with all features inside area C belong to Group II, while the remaining objects are discarded.
2. Inside of each bucket travel objects are sorted according to their temperature computed in the following way: for a given object *O* its temperature

   $$temp(O) = \sum_{f \in O} temp(f)$$

   where $temp(f) = 1$ if $f \in A \cup B$, or $p_n(f)$ if $f \in C$, while $temp(f)=temp(f) - 0.5$. This latter calculation is performed to implicate that these features that are not of interest to the user (their individual temperatures are less than 0.5) reduce the overall temperature of the object. Function $p_n(f)$ is a normalized probability of feature *f*, based on the user-profile.

Let us consider Karol, who is interested in selecting a restaurant. In his query he specified that this restaurant has to serve Italian cuisine and has to allow smoking. Additionally, we know, from Karol's profile, that he does not like *coffee* (weight 0.1) and *outdoor dining* (weight 0.05). Thus for the restaurant X:

```
:RestaurantX a res:Restaurant;
      res:cuisine res:ItalianCuisine;
      res:cuisine res:PizzaCuisine;
      res:cuisine res:CafeCoffeeShopCuisine;
      res:feature res:Outdoor.
```

the overall score will be decreased due to the influence of *Outdoor* and *CafeCoffeeShopCuisine* features, but will receive a ″temperature boost" because of the *ItalianCuisine* feature (explicitly specified feature). However, the restaurant X it won't be rated as high as the restaurant Y:

```
:RestaurantY a res:Restaurant;
      res:cuisine res:ItalianCuisine;
      res:smoking res:PermittedSmoking.
```

which serves *ItalianCuisine*, where smoking is also permitted. To be more specific, let us consider these two restaurants and the third one described by the following features:

```
:RestaurantZ a res:Restaurant;
      res:cuisine res:WineBeer;
      res:smoking res:PermittedSmoking.
```

Then the following table represents the way that temperatures of each restaurant will be computed.

Restaurant N3 descriptions (bold – requested, underlined – in the user profile; could be conjunctive)	Calculations
:RestaurantX a res:Restaurant; res:cuisine **res:ItalianCuisine**; res:cuisine res:PizzaCuisine; res:cuisine res:CafeCoffeeShopCuisine; res:feature res:Outdoor.	+0.5 (=1-0.5) requested; B +0 -0.49 (=0.01-0.5) profile -0.45 (=0.05-0.5) profile **= -0.44**
:RestaurantY a res:Restaurant; res:cuisine **res:ItalianCuisine**; res:smoking **res:PermittedSmoking**.	+0.5 (=1-0.5) requested; B +0.5 (=1-0.5) requested; B **= 1**
:RestaurantZ a res:Restaurant; res:cuisine res:WineBeer; res:smoking res:PermittedSmoking.	+0.3 (=0.8-0.5) not requested; profile; C +0.5 (=1-0.5) not requested; profile; C **= 0.8**

As a result, restaurants X and Y belong to the first bucket (to be displayed to the user as they both have features that belong to area B). However, while restaurant Y has high temperature (1) and definitely should be displayed, restaurant X has very low temperature (-0.44) and thus likely will not be displayed at all. Interestingly, restaurant Z, which

belongs to the second bucket (belongs to area C), has an overall score of 0.8 and is likely to be displayed. This example shows also the potential adverse effect of lack of information (e.g. in the ChefMoz repository; but more generally, within the Web) on the quality of content-based filtering (at least done in a way similar to that proposed above). Simply said, what we do not know cannot decrease the score, and thus a restaurant for which we know only address and cuisine may be displayed as we do not know that it allows smoking on premises (which would make it totally unacceptable to a given user).

RDF data Utilization – Content Delivery

Let us now present in more detail how the delivery of content to the user is implemented as an agent system. To be able to do this we need to briefly introduce additional agents (beyond these presented in Figure 2) and their roles (using Prometheus methodology (Prometheus, 2005)) – as represented in Figure 13.

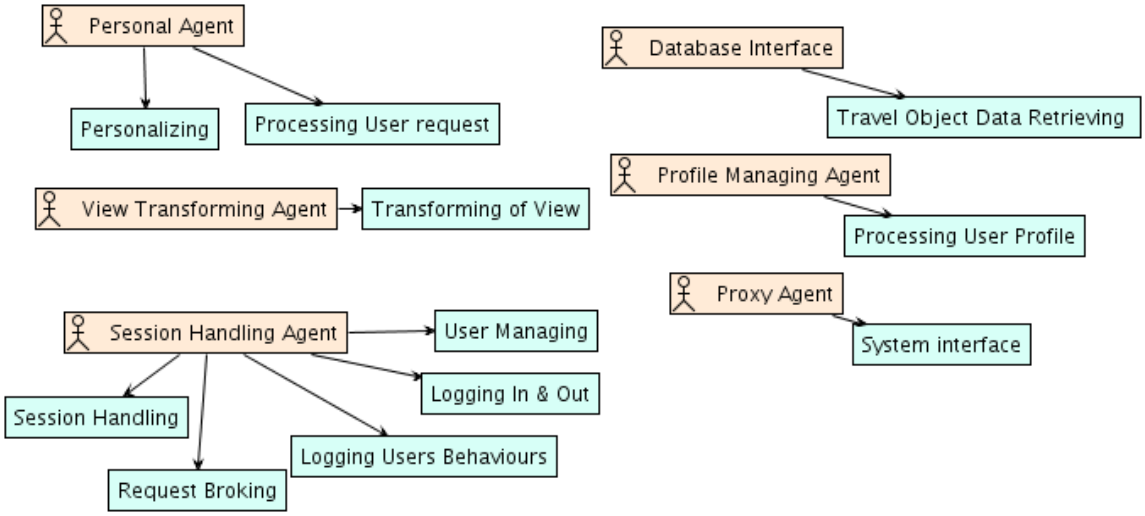

Figure 13. Content delivery agents and their roles.

In addition to the *PA* (described in details in Figure 7) and the *DBA*, we have also: (1) *View Transforming Agent* (*VTA*) responsible for delivering response in the form that matches the user I/O device, (2) *Proxy Agent* (*PrA*) that is responsible for facilitating interactions between the agent system and the outside world (need for these agents as well as a detailed description of their implementation can be found in (Kaczmarek, 2005)), (3) *Session Handling Agent* (*SHA*), which is responsible for complete management and monitoring of functional aspects of user interactions with the system, (4) *Profile Managing Agent* (*PMA*) which is responsible for (a) creating profiles for new users, (b) retrieving profiles of returning users and (c) updating user-profiles, based on implicit and explicit relevance feedback. Let us now summarize processes involved in content delivery through an UML action diagram. While rather complex, description contained in Figure 14 represent a complete conceptualization of actions involved in servicing user request from the moment that user logs into the system, to the moment when she obtains response to her query.

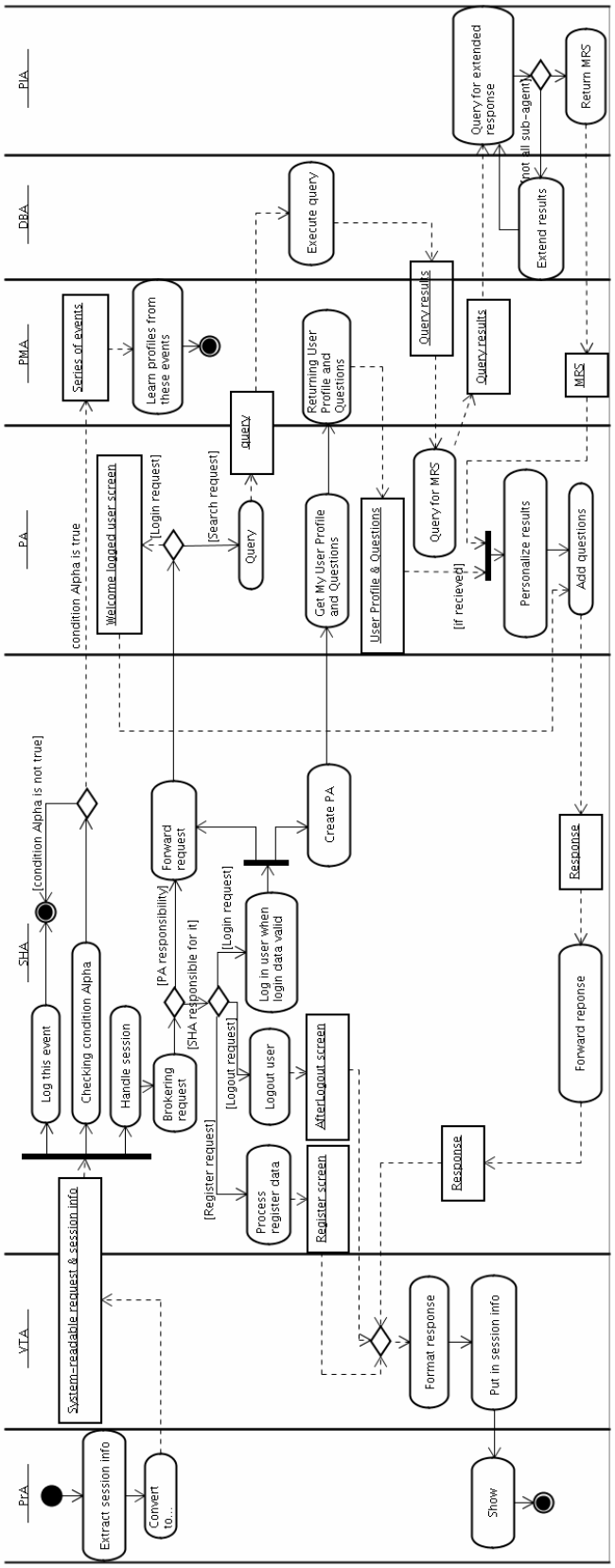

Figure 14. Content delivery action diagram.

State of the System

As indicated earlier in this chapter, we have concentrated on these features of our system that are currently being implemented and close to being ready, while omitting these features that we would like to see developed in the future. While the interface to the system is still under construction, it is possible to connect to it from a browser. Furthermore, we have emulated WAP-based connectivity. As of the day this chapter being written, we have implemented a function-complete content collection subsystem consisting of: (1) a number of hotel wrappers (*WA*) that allow us to feed hotel data into the system, (2) *CA* and *IA* agents that collaborate with the *WA*s to insert data into JENA-based repository, (3) initial version of the *DMA* and the *PIA*. For the CCS we have semi-automatically cleaned-up subsets of ChefMoz data, describing selected restaurants. We have also a relatively complete content delivery subsystem. In particular, (1) the *PrA*, the *SHA* and the *VTA* that facilitate user-system interactions have been implemented and tested, (2) the *PA* is working as described in this chapter (with the *PIA* working in the case of restaurants only), (3) the *PMA* has only limited capacity, it is capable of creating and managing a single user profile, (4) while the existing set of stereotypes involves only restaurants. Let us briefly illustrate the work of the system, by screen-shots of the query (Figure 15) and the response (Figure 16). The query was a general query about restaurants in Greensboro, NC; note the box that attempts at asking a question about *Bistro Sophia* that was suggested to the user in the previous session (Figure 16).

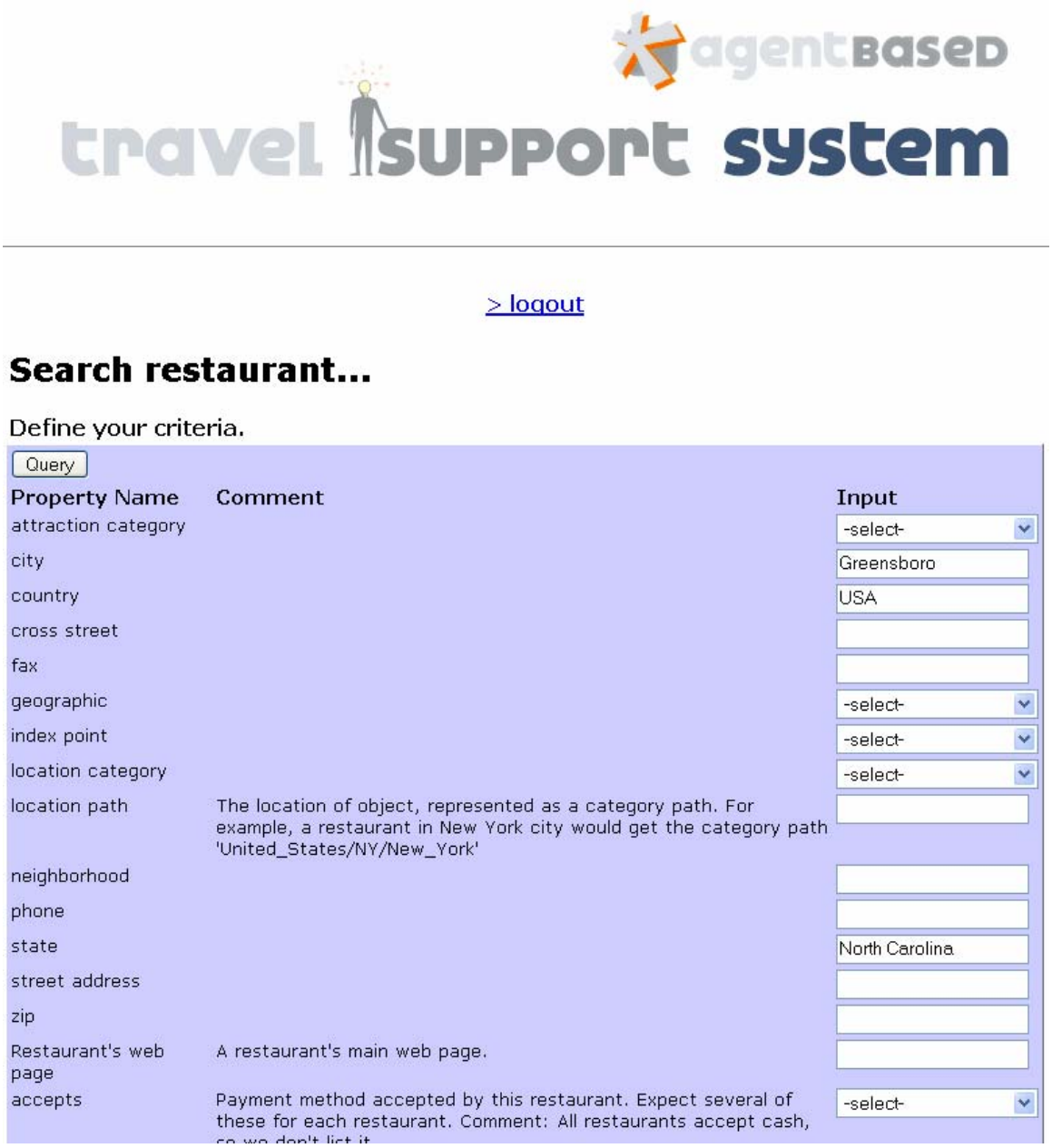

Figure 15. System query screenshot

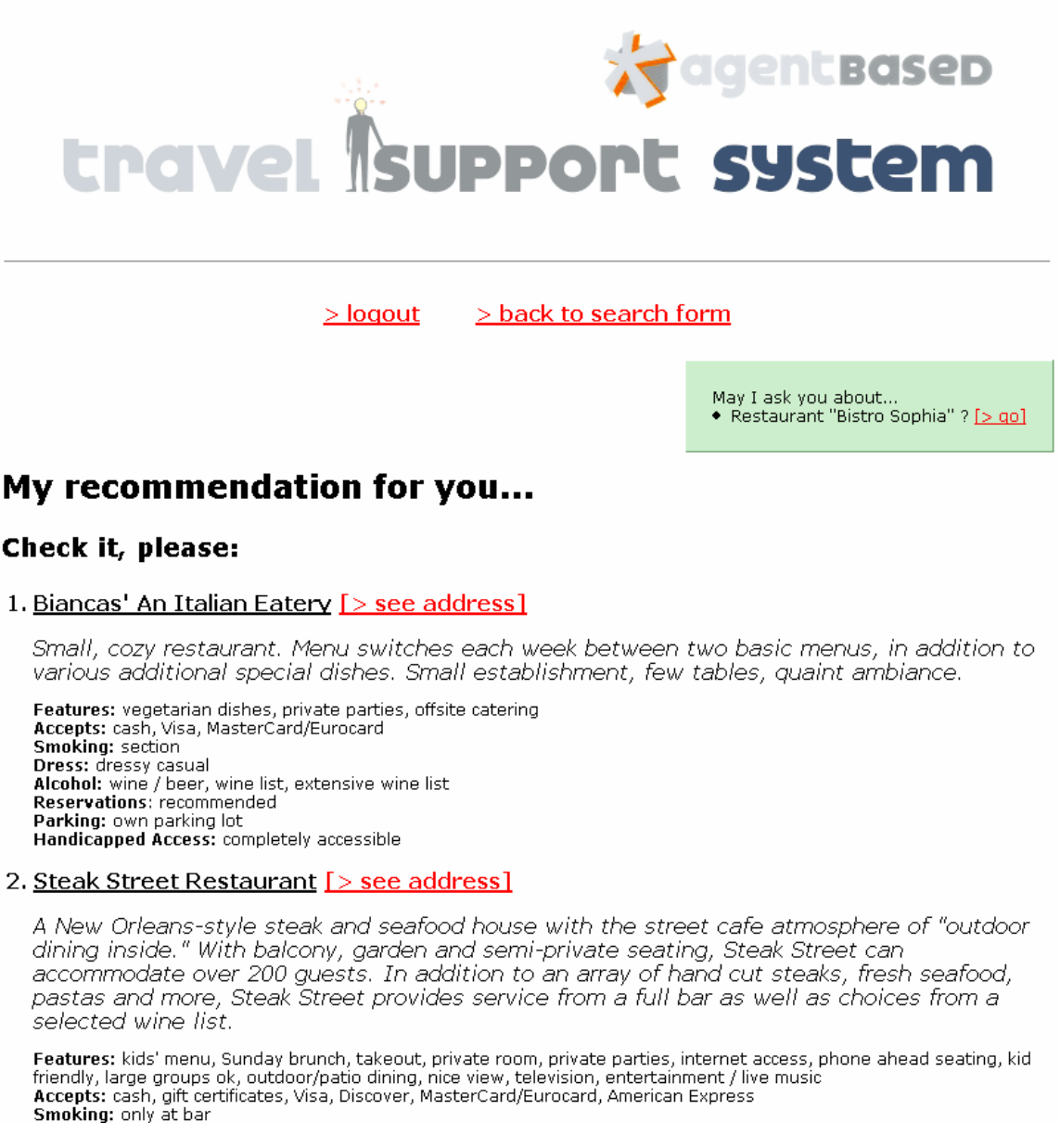

Figure 16. System response screenshot

One of the biggest problems related to testing our system is the fact that, being realistic, no user would be interested in a system that only provides a few hotel chains and restaurants (e.g. in Poland). This being the case we can ourselves test features of the system like: (1) is the user query handled correctly, i.e. do the returned results represent the correct answer taking into account the current state of the system, (2) do the *WA*s correctly deliver and the *CA* and *IAs* accurately insert tokens into the system, (3) are agent communication and interactions proceed without deadlocks and does the system scale. Unfortunately, it is practically impossible to truly test adaptive features of the system. Without actual users utilizing the system to satisfy their real travel needs, all the work that we have done implementing the system cannot be practice-verified.

This is a more general problem of the chicken-and-egg type that is facing most of Semantic Web research (regardless of its application area). Without real systems doing real work and utilizing actual ontologically demarcated data on a large scale (to deliver what users need) it is practically impossible to assess if the Semantic Web, the way was conceptualized, is the way that we will be able to deal with information overload, or is it just another pipe-dream like so many in the past of computer science.

Future developments

As described above, it seems to be clear what the future of the development of Semantic Web technologies applied in context of e-business (or in any other context) has to be. It has to follow the positive program put forward by Nwana and Ndumu. The same way as agent systems, large number of systems utilizing Semantic Web technologies has to be implemented and experimented with. Furthermore, it is necessary to develop tools that are going to speed up ontological demarcation of Web content. Here, both the content that is about to be put on the Web as well as tools supporting demarcation of legacy content need to be improved and popularized. Only then, we will be able to truly assess the value proposition of the Semantic Web. Furthermore, since software agents and the Semantic Web are truly intertwined, the development of the Semantic web should stimulate development of agent systems, while development of agent systems is likely to stimulate development of the Semantic Web.

To facilitate these processes we plan to continue development of our agent-based travel support system. The first step will be complete integration and testing of the above described system skeleton. We will proceed further by, among others: (1) developing ontologies of other important travel objects e.g. movie theaters, museums, operas, etc., (2) fully developing and implementing the *PIA* and the *DMA* infrastructures – according to the above presented description, (3) continuing implementing *WA*s to increase the total volume of data available in the system, (4) adding a GIS component to the system, to allow answering queries like: which restaurant is the closest one to that hotel?, (5) developing and implementing an agent-based collaborative filtering infrastructure, (6) investigating potential of utilizing text processing technologies for developing new generation of adaptive *WA*s.

References

Angryk R., Galant V., Gordon M., Paprzycki M. (2002): Travel Support System - an Agent Based Framework. In: H. R. Arabnia, Y. Mun (eds.), Proceedings of the International Conference on Internet Computing (IC'02), CSREA Press, Las Vegas, NV 719-725, 2002

T. Berners-Lee, J. Hendler, O. Lassila, (2001): The Semantic Web, Scientific American, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

Burke R. (2002): Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction, 12, 4, 331 - 370.

ChefMoz (2005) ChefMoz Dining Guide, http://chefdmoz.org

DAML (2005) Language Overview, http://www.daml.org/

Fensel D., (2001) Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce, Springer, Berlin

Fink J., Kobsa A. (2002): User Modeling for Personalized City Tours. Artificial Intelligence Review, 18 (2002) 33-74

Galant V. and Paprzycki M. (2002a): Information Personalization in an Internet Based Travel Support System. Proceedings of the BIS'2002 Conference, Poznań, Poland, April, 2002, pp. 191-202

Galant V., Gordon M., Paprzycki M. (2002b): Agent-Client Interaction in a Web-based E-commerce System. D. Grigoras (ed.), Proceedings of the International Symposium on Parallel and Distributed Computing, University of Iaşi Press, Iaşi, Romania, 2002, 1-10

Galant V., Gordon M., Paprzycki M. (2002c): Knowledge Management in an Internet Travel Support System. B. Wiszniewski (ed.), Proceedings of ECON2002, ACTEN, Wejcherowo, 97-104

Galant V., Jakubczyc J., Paprzycki M. (2002d): Infrastructure for E-Commerce. in: M. Nycz, M. L. Owoc (eds.), Proceedings of the 10th Conference Extracting Knowledge from Databases, Wrocław University of Economics Press, 32-47

Gawinecki M., Gordon M., Paprzycki M., Szymczak M., Vetulani Z., Wright J. (2005a): Enabling Semantic Referencing of Selected Travel Related Resources. In: W. Abramowicz, Proceedings of the BIS'2005 Conference, Poznań University of Economics Press, Poznań, Poland, 271-290

Gawinecki M., Gordon M., Nguyen N., Paprzycki M., Szymczak M. (2005b): RDF Demarcated Resources in an Agent Based Travel Support System. In: Proceedings of the 17th Mountain Conference of the Polish Information Society, (to appear)

Gawinecki M., Vetulani Z., Gordon M., Paprzycki M. (2005c): Representing Users in a Travel Support System, in: Kwaśnicka, H. et. al. (ed.) Proceedings of the ISDA 2005 Conference, IEEE Press, Los Alamitos, CA, 2005, 393-398

Gawinecki M., Kruszyk M., Paprzycki M. (2005d) Ontology-based Stereotyping in a Travel Support System, in: *Proceedings of the XXI Fall Meeting of Polish Information Processing Society*, PTI Press, 73-85

Gilbert A., Gordon M., Nauli A., Paprzycki M., Williams S., Wright J., (2004): Indexing Agent for Data Gathering in an e-Travel System. Informatica, Vol. 28, No. 1, 69-78

Gordon M., Paprzycki M. (2005a): Designing Agent Based Travel Support System, in: *Proceedings of the ISPDC 2005 Conferenc*e, IEEE Computer Society Press, Los Alamitos, CA, 207-214

Gordon M., Kowalski A., Paprzycki N., Pełech T., Szymczak M., Wasowicz T. (2005b) Ontologies in a Travel Support System, in: D. J. Bem et. al. (eds.) *Internet 2005*, Technical University of Wrocław Press, Wrocław, Poland, 285-300

Greer, J., McCalla, G. (1994) Student modeling: the key to individualized knowledge based instruction, Springer-Verlag, NATO ASI Series, 3-35

Harrington P., Gordon M., Nauli A., Paprzycki M., Williams S., Wright J. (2003): Using Software Agents to Index Data in an E-Travel System. N. Callaos (ed.), Electronic Proceedings of the 7th SCI Conference, Orlando, 2003, CD, file: 001428.pdf, 6 pages

J. Hendler (1999) Is There an Intelligent Agent in Your Future?, Nature, 11 March, 1999, http://www.nature.com/nature/webmatters/agents/agents.html

Hendler J, (2001) Agents and Semantic Web, IEEE Intelligent Systems Journal, 16, 2, 30-37

JADE (2005) http://jade.tilab.com/

JENA (2005) Jena 2 - A Semantic Web Framework, Hewlett Packard, http://www.hpl.hp.com/semweb/jena2.htm

Jennings N. R. (2001): An agent-based approach for building complex software systems. CACM 44, 4, 2001, 35-41

Kaczmarek P., Gordon M., Paprzycki M., Gawinecki M. (2005): The Problem of Agent-Client Communication on the Internet. Scalable Computing: Practice and Experience, 6(1), 2005, 111-123

Kobsa A., Koenemann J., Pohl W. (2001): Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships. The Knowledge Engineering Review, 16:2, 2001, 111-155

Maes P. (1994): Agents that Reduce Work and Information Overload. Communications of the ACM, 37, 7, 1994, 31-40

Montaner M., López B., De La Rosa J. L., (2003) A Taxonomy of Recommender Agents on the Internet, Artificial Intelligence Review, 19, 285–330

Nistor C. E., Oprea R., Paprzycki M., Parakh G. (2002): The Role of a Psychologist in E-commerce Personalization. Proceedings of the 3rd European E-COMM-LINE 2002 Conference, Bucharest, Romania, 2002, 227-231

Nwana H., Ndumu D. (1999): A perspective on software agents research. The Knowledge Engineering Review, 14, 2, (1999) 1-18

Orłowska, M. (2005) personal communication

OWL (2005) OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/

Paprzycki M., Angryk R., Kołodziej K., Fiedorowicz I., Cobb M., Ali D. and Rahimi S. (2001a) "Development of a Travel Support System Based on Intelligent Agent Technology," in: S. Niwiński (ed.), Proceedings of the PIONIER 2001 Conference, Technical University of Poznań Press, Poznań, Poland, pp. 243-255

Paprzycki M., Kalczyński P. J., Fiedorowicz I., Abramowicz W. and Cobb M. (2001b) "Personalized Traveler Information System," in: Kubiak B. F. and Korowicki A. (eds.), Proceedings of the 5th International Conference Human-Computer Interaction, Akwila Press, Gdańsk, Poland, pp. 445-456

Raccoon (2005) http://rx4rdf.liminalzone.org/Raccoon

RDF (2005) RDF Primer, http://www.w3.org/TR/rdf-primer

Rich E. (1979): User Modeling via Stereotypes. Cognitive Science 3, 329-354

Prometheus (2005) Prometheus Methodology, http://www.cs.rmit.edu.au/agents/prometheus/

Sołtysiak S., Crabtree B. (1998): Automatic learning of user profiles - towards the personalization of agent service. BT Technological Journal, 16 (3), 110-117

Wooldridge, M. (2002) An Introduction to MultiAgent Systems, John Wiley & Sons

Wright J., Gordon M., Paprzycki, M., Williams S., Harrington P. (2003):Using the ebXML Registry Repository to Manage Information in an Internet Travel Support System. W. Abramowicz and G. Klein (eds.), Proceedings of the BIS'2003 Conference, Poznań University of Economics Press, Poznań, Poland, 2003, 81-89

Zaslavsky, A. (2004) personal communication