DOI 10.12694/scpe.v14i2.844
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 2, pp. 111–129. http://www.scpe.org

ESTABLISHING SEMANTIC CLOSENESS IN AN AGENT-BASED TRAVEL SUPPORT SYSTEM

MARIUSZ MESJASZ*, MARCIN PAPRZYCKI†, AND MARIA GANZHA‡

Abstract.
Agent system that supports needs of travelers is a classic example of an application of personal agents. This paper re-evaluates assumptions behind the design and implementation of an agent-semantic travel support system (TSS) that was proposed in 2006. The goal is to modify these assumptions to match them with the current user-demands and adjust to the technological changes. On the basis of this analysis, the design of a new TSS is introduced. The key element of the proposed system is semantic matchmaking, which is used to establish, which travel services are to be suggested to the user (an in which order). Therefore, we describe in detail the proposed matchmaking algorithm and its implementation, and illustrate its work through selected examples.

Key words: Software agents, JADE, ontological matchmaking, recommender system, resource management, resource matching, semantic closeness

1. Introduction. One of the areas where modern technology is making its mark is the "world of travel." Nowadays, tourist can use his smart-phone to locate restaurants in the nearby area. Then, based on their ranking, published on a website (available through a smart-phone application), she can choose the one that matches her food preferences, while being highly recommended by others. Similarly, she can use web-based services to find means of transportation to reach the selected restaurant.

Unfortunately, this scenario sounds easier / nicer than it is in reality. First, as a matter of fact, tourist may be "forced" to visit several websites to gather the necessary information. Second, she has to spend time comparing the results to make an educated choice. To avoid the hassle, she may make the *simplest* selection (possibly, "contradictory" to his preferences), instead of looking for the *best* one. For example, a hungry tourist may choose an easily-accessible fast food place (even though she does not like fast-food), over restaurants that serve her favorite cuisine. The problem remains even when a tourist is in a hotel with an Internet access and spare time to check "everything." Here, she will spend time searching and comparing restaurants, rather than doing "something else." This illustrates the need for a travel support system that will be able to "intelligently" work for the user. For example, a tourist should be able to make a selection from a short list of candidates, based on cross-referenced sources, and containing only results that match her preferences.

2. State of the art in related areas. Let us start by considering state of the art in three areas related to travel support. First, we will consider general web-based travel services. Second, we will discuss travel services related to mobile technologies. Finally, we will comment on the state of ontology-based personal assistants.

2.1. Examples of travel-related systems. There are several services and applications, which facilitate travel-related information. Here, we discuss some of them, including two no longer developed systems - the "original" Travel Support System (predecessor of the one described in this paper), and the Chefmoz (ontology-based collection of restaurants). Despite the fact that they are no longer active, they provide a useful source of information and reflection.

Travel Support System (TSS; 2000–2008) The preliminary idea of the Travel Support System was introduced in the MS Thesis from 2000 [29]. The TSS was an agent-based system responsible for collecting information available on the Internet, storing it in an RDF database, and creating personalized answers to users' queries. The initial idea was explored in a series of publications that appeared between 2001 and 2008 (available at [18]). The working system prototype was released in 2006, and is still available at the SourceForge repository [19].

Interestingly, the need to redesign the system became evident immediately after its initial implementation. Most important lessons learned in the process of its development, implementation and evaluation

*Warsaw University of Technology, Department of Mathematics and Information Sciences
†System Research Institute Polish Academy of Sciences and Warsaw Management Academy (marcin.paprzycki@ibspan.waw.pl)
‡System Research Institute Polish Academy of Sciences and University of Gdask

were summarized in [28]. The publication pointed out to several design flaws concerning utilization of agents in the TSS system.

However, none of these problems have been resolved, as the development of the TSS stopped. Furthermore, due to broken software dependencies, the system is no longer operational. Therefore, the new implementation of the system proposed in this work started from requirements analysis, taking into account the previously acquired knowledge and the development of new technologies.

Chefmoz (2000 2011) Chefmoz was a branch of the Open Dictionary Project [9] and contained an online directory of restaurants (and their reviews). The collection covered more than 300,000 restaurants from 142 countries. In 2009, ChefMoz had become the largest (available on the Internet) global directory of restaurants. Similarly to the Open Dictionary Project, the Chefmoz provided its data as a hierarchical ontology scheme (represented as "low quality" RDF triples; where, low quality means that they did not actually adhere to ontological standards, contained large number of non-RDF tags, and could not have been imported into ontology development platforms, e.g. Protege, without considerable amount of extra manual work). Due to the persistent technical difficulties and lack of funding, the Chefmoz project was announced "dead" in 2011.

Booking.com Booking.com [3] is a hotel reservation portal. It allows its users to browse hotels, book rooms, and post their opinions. Website has a number of important features such as a large database of hotels (over 250,000 entries), opinions posted by real hotel guests, and worldwide accessibility (consumer support in more than 40 languages). Here, Booking.com represents a broad class of hotel-focused services (information aggregators) such as: hotels.com, venere.com, etc.

Despite the fact that Booking.com is a high quality service, we have to keep in mind that it is limited to hotel information. Therefore, it cannot answer complex queries (e.g. find hotels located near museums or art galleries). Moreover, Booking.com does not provide personalized recommendations. For example, it can be conjectured that a many users could find value in offers adjusted to their preferences, instead of a regular newsletter containing "statistically good deals" (for example, the cheapest prices).

Social Networks Many social networks could be a source of travel-related information and affect decisions where to spend time. The two most popular social networks of today are Facebook [5] and Twitter [21]. Users of both, can post comments, upload photos and share information that can influence travel decisions. For example, if our friends frequently visit a place, publish cool pictures, and localize the place on the map, it is likely that we will consider going there in the future. Observe that the TripAdvisor [20], a web page similar to Booking.com, uses this idea and integrates its services with Facebook. Customers may connect to their Facebook accounts and see offers from places visited by their friends.

Although this is an interesting use of social networks, this approach has important drawbacks (from the point of view of the development of the TSS). First, these services allow users to share all kinds of information (not only related to travel). For example, the user can post on Facebook not only information about a restaurant which, in her opinion, serves the best pepperoni pizza (travel information about a certain place), but also a comment about the current political situation in Andorra (information barely related to the TSS). Secondly, in the social media, the term "recommendation" is not appropriate (for example, while messages may encourage a user to go to a specific location or participate in an event, none of them was an actual recommendation). Finally, systems based on current social networks, would have to be substantially augmented to take into account the user profile. In their natural form they would be most likely based on opinions from "friends" (leading to questions as to the actual nature of social network acquaintances). Therefore, it might not be easy to provide recommendations that would match user preferences.

Yelp! Yelp! [25] is a web catalog of local businesses. It operates as a social network service – its content is filled by advertisers (local businesses) and ordinary users (reviewers and potential customers). The drawback is that the system does not provide personalized, or context-driven, recommendations. Therefore, users receive a newsletter, which is loosely connected to their profile (for example, it is based on the city where they live). As the result, the user receives a limited number of local offers that may or may not match his preferences. Yelp! is also a medium for friend-based recommendations. As a result, offers depend on the "external" recommendations, not on users' preferences. Finally, user recommendations

have to be completed manually, which takes time that the user may not be willing to spend.

Google Now Google Now [6] is an intelligent personal assistant provided by Google. The most important feature of Google Now is the ability to display information obtained in the interaction with the Google services, e.g. weather, traffic information, or dishes recommended in a restaurant. Information provided by the application is selected on the basis of Google collected data, such as Web History and location services (for instance, the GPS). Although Google Now seems to be similar to the Travel Support System, there are some differences. First of all, the TSS is focused on supporting needs of "travelers." Google Now, on the other hand, promotes use of Google services. Second, Google Now forces users to store personal information on Google's servers. Despite the fact that this is a useful feature (for example, if a mobile device is stolen), users have no other choice. Finally, as seen above, there is more to the world of travel than the Google services, and user should not be forced to use only one set of services, regardless how good it is.

2.2. Travel-related mobile applications. Let us now describe few smartphone-based applications that have been developed to support needs of travelers. First of all, it is worth to notice that most webased services, mentioned in the previous section, provide their own mobile applications. Unfortunately, such mobile applications often do not have additional features beyond the ability to quickly browse content of the web page of a given service on the mobile device. Thus, they suffer the same limitations as tehir web-based versions. For example, the mobile application [4] provided by Booking.com helps to quickly navigate through the list of available hotels. However, no one can use this application to cross-reference hotels with a list of museums. Therefore, similarly to Booking.com, the application provides only data related to hotels.

Secondly, due to the openness of application stores, for popular mobile operating systems like Google Android (Play Store) and iOS (iTunes App Store), a large number of travel-related applications has been developed. For example, OpenRice Hong Kong [11] (version 2.7.0) developed by Openrice Group Inc. [10] is a dining guide, which provides information about more than 40,000 OpenRice restaurants across Hong Kong. Unfortunately, according to its user reviews, it has some major bugs. Users have pointed out frequently crashes, poor quality on HD displays and persistent error messages. Yet another example, iPic Theaters [8] (version 1.2), developed by the iPic Entertainment [7], is a mobile application which helps its users buy tickets. Similarly to the OpenRice, its users complain about poor application performance. Specifically, the application can crash after the credit card information is provided. As a result, its users cannot be sure if the reservation has been processed. These two examples illustrate how hard it is to find a reliable travel-related applications.

Finally, with the development of the Internet, the idea of Semantic Web [16] is becoming ever more important. Here, the final outcome should be meta-level interoperability and understanding of all Internet-stored data. However, there are no well-known mobile applications that support this idea. All applications operate on their own data (for example, Booking.com, Yelp!) or connect to free data sources (sometimes characterized by low-quality content). Therefore, there is no unified way to obtain complex information (for example, cross-referencing monuments with restaurants). The user is still forced to use different travel-related applications in order to find some information separately.

Summarizing, there exists vast number of travel-related mobile applications, but very often they suffer from important limitations. These limitations either come from the source web-based services (the application is simply a front-end) or are caused by utilization of inaccurate data sources. Moreover, many developers failed to deliver a user-friendly, high-quality software. Overall, it can be stated that at the time of writing of this paper, application stores are full of low-quality mobile applications. Finally, despite the fact that Semantic Web could provide many benefits (unified access to the information, and cross-referencing), this idea is not utilized.

2.3. Semantic personal assistants. Let us now look into ontology-driven travel assistants. Ideas in this direction have been formulated already in 2003 (or, possibly, earlier). For instance, in [1] authors discuss the general idea of a semantic personal agent. Here, one of the use case application areas is travel support. Paper refers multiple technologies of the time and mentions DAML-based ontologies as the way of representing knowledge. However, it seems that the project did not progress beyond a very limited prototype.

In 2008, authors of [17] discuss a Smart Travel Service Adviser (STSA). The STSA is an agent-based solution for searching travel services, and it is focused on the User Satisfaction Level (USL). Here, RDF triples were to be used to represent knowledge. They were to be combined with Jena middleware for semantic data

storage and management, and with SPARQL for semantic queries. In this way the STSA project resembled the original TSS. Here, again, it does not seem that the project progressed beyond the initial prototype.

The lack of *actual* progress in the area of semantic personal agents for travel support is illustrated by the short article from May, 2011 [2]. Here, Larry Smith, repeats well known mantra that the semantic web will produce great results. However, he is not pointing to any existing / being developed applications.

One could assume that the problem is with development of semantic agents to be used for travel support. However, this is not the case. Let us mention just two examples. The EU funded project "A Platform for Organisationally Mobile Public Employees" (Pellucid; [12, 13, 14]) aimed at development of semantic personal agents to support worker mobility. Currently, all that remains are the web pages (some of them already returning the 404 error) and published articles. No actual application of Pellucid Semantic Assistants can be located.

Second, let us mention a Polish national Project WKUP (Virtual consultant for public services; [22, 23]). The aim of the project was to create an agent-semantic assistant (avatar) to support users of public services. Apparently, in 2009, in the Radlin county there was a trial run of the system. However, currently the www site of the trial is a broken link, and no other application of software developed within the WKUP project can be found.

In summary, the idea of semantic (travel oriented) personal assistants is very popular since at least early 2000. However, regardless of the public funding devoted to research, and large number of publications devoted to the subject, it is practically impossible to locate working prototypes (not to mention real-world applications).

3. Proposed System – Preliminary Considerations. Let us recall that systems outlined above have certain shortcomings. The information that they provide is either very narrow (e.g. Booking.com), or very robust but obscure (e.g. social networks and Google Now). Based on analysis of their functions, we can formulate the main features of our Travel Support System. However, before presenting the list of requested features, let us present two use case scenarios.

Scenario 1 Jack is on a business trip in a city that he has not visited before. Spending his free time between business meetings, he decided to visit the city center. After some time of sightseeing, he became hungry. Thankfully, he has installed (on his tablet with build-in 3G modem) the Travel Support System (TSS). The system is aware that Jack is a "committed vegetarian" (this information is stored in his TSS *user profile*). Therefore, the TSS rejects a very popular restaurant serving steaks (that happens to be near-by) and recommends a little known Thai/Indian place that serves vegetarian/vegan meals.

Scenario 2 Jack recently met Jill, and began to SMS-chat with her. After Jack sends a message to Jill, asking for a date, (assuming that she agrees) his instance of the Travel Support System starts negotiations with the one running on Jill's smart phone. Since Jack is already in her contacts, her Travel Support System shares selected information about her preferences (stored in her TSS profile). As a result, Jack's TSS sends to Jill's TSS a suggestion to meet in an Indian restaurant. In response, Jack is informed that Jill will likely be satisfied with this choice since she likes Indian food (information confirmed by her TSS). Note that we do recognize privacy issues present in this scenario, but we believe that they can be appropriately addressed.

From the use case scenarios, one can derive some observations. First, the system should produce personalized recommendations, based on user preferences (stored in a TSS user profile). For example, the restaurant proposed to Jack serves the appropriate type of food – vegetarian meals. Similarly, in the second scenario, the Indian restaurant recommended to Jack and Jill serves food that they both like. Second, the system should not dependent on location (restaurant can be anywhere) or the timing (now, or in a few days). In addition, there should be no difference between a spontaneous decision (find a restaurant now) and a deliberate action (planning dinner ahead of time). Depending on the situation, the system should either produce immediate results, or take time to prepare an in-depth list of potentially interesting places.

These observations allow us to present our vision of the TSS; depicted as a use case diagram, in figure 3.1. There we represent three entities located outside of the system: the *GPS*, the *User* and the *Data Source*. They interact with the system by asking queries (*User*), providing information (*GPS*, *Data Source*) and negotiating. The main actor of the system is the *User*. The TSS is involved in five functions: *Interacting* with the *User*, *Context monitoring*, *Learning*, *Negotiating* and *Cross Referencing*.

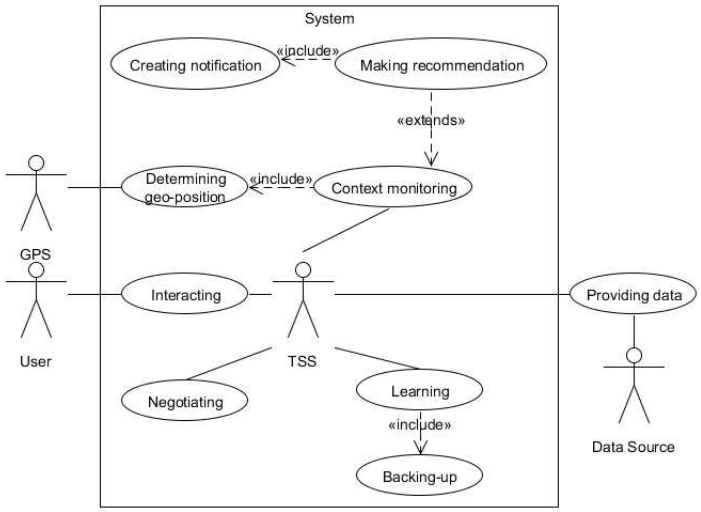

FIG. 3.1. *Use case of the Travel Support System*

The *Interacting* function involves preparing a list of results that match user preferences or correspond to the user input. It is also connected to the *Cross-referencing* and the *Learning* functions. The *Cross-referencing* function is used to obtain the results from one or more *Data Source*s. By the nature of the system, each action taken by the *User* triggers the *Learning* function. During the learning phase, the system adjusts user preferences stored in the hierarchical structure called the *user profile*. Due to this, the system is able to obtain useful information from the *Data Sources* (the information that match the user preferences). The *Context monitoring* function involves dealing with all contextual information available to the system. The most important contextual information is the geopositioning, provided by the *GPS*. However, the *Context monitoring* may also involve other types of information (for example, text messages, calendar entries, and the remaining battery power of a mobile device). The *Context monitoring* can trigger the *Making recommendation* function. The *Making recommendation* function involves proactive searching for information, based on the current user context. It is also connected to the *Cross-referencing* and the *Creating Notification* functions. The *Creating notification* function involves displaying a notification about a new set of recommendations made by the system. Here, it is assumed that this function will adjust the display format to the actual user device (used to interact with the TSS). The *Negotiating* function that connects two (or more) instances of the TSS, involves exchanging messages (in order to achieve an agreement). In the TSS, the agreement may, for instance, concern selection of a meeting place that will be satisfactory to two or more users. The *Negotiating* function is also connected to the *Checking permissions* function. The *Checking permissions* function ensures the confidentiality of shared information (for example, the local TSS will not start negotiations with a remote TSS if its owner is not in the contact list). Finally, the *Learning* function is connected to the *Backing-up* function that stores the current user profile on a remote server. This is crucial to prevent data loss when something happens to the mobile device (e.g. it is lost or stolen).

On the basis of what has been said thus far, we propose the following requirements for the TSS:

Use of mobile technology With the increasing popularity of mobile devices, and an almost ubiquitous access to high speed Internet connection, the Travel Support System should be able to take advantage of these capabilities. Thus, it should be able to run on smart phones, tablets, notebooks, as well as on desktop PCs. Furthermore, it should use services available in such devices (e.g. geo-positioning in smart phones and tablets). Finally, the system should be robust enough to allow users to select the right device for given situation (in the hotel room, the most preferable device could be a laptop, while on a street it may be a tablet or a smart-phone). However, the system should take limitations of each device into account and provide a suitable display mode for each of them.

Context-awareness In order to provide the best possible response to the user queries (and to be proactive), the system has to monitor user's contextual information. In the travel-related system, the most useful information is the geo-position (the GPS feed), which can be used most of the time (except, when the user decides otherwise). In the second scenario, an additional data source was considered: SMS messages. These messages may be used by the system to be proactive. They do not directly alter user queries, but may trigger autonomous actions within the application. For instance, a content of text messages is not included in the user data (the system is not likely to understand what it means), but it could try to involve them to produce a useful recommendation (it can query certain keywords – for example, "restaurant," "French," "dinner" – and check the results against the user profile).

Personalized Searching The result of a search has to be personalized to individual user's requirements. The requirements may be specified directly (user explicitly adds them to the query), or indirectly (the system is aware of user's previous choices and tendencies – instantiated in the profile). The system has to filter-out data contrary to the direct, or indirect, requirements and favor those which match user preferences, and are more likely to fulfill her needs. Therefore, the personalized results should be presented as a response to a query, while other results may be added to introduce variation and/or fill the space (e.g. in the case when only few services matching the query can be found).

Here, let us address the fact that the system should take into account potential changes of user preferences (and desires for something new). Thus, from time to time, it should return a result which is neutral or even contradictory to the stored preferences. The average time between such proposals may be measured by analyzing changes in user queries and responses to unusual suggestions. The contradictory result should be somehow "separated" from others (and marked appropriately). However, the system should be aware that some preferences may be very stable or practically unchangeable. For example, in the first scenario, Jack is a *committed* vegetarian, hence the system should not recommend him a steak.

Collecting important user information In order to allow personalized searching and to manage the user profile, the system must be able to collect information about the user. Observe that the user profile should be dynamically adjusted over time, based on the user interaction with the system (e.g. queries and responses, and decisions based on them).

Negotiations Objectives of negotiations may differ, depending on the functionality of the system. For example, in the original Travel Support System ([29]) negotiations between multiple instances of the system - representing travel service sellers and tourists – were proposed. There, special offers and promotions, based on user preferences, were to be negotiated. For instance, a restaurant's recommendation might also include information about availability of a free appetizer.

The new version of the system focuses on negotiations with a different objective. For instance, in the second scenario, Jack and Jill want to go out for a dinner, but none of them knows the best *common* place to go to. Therefore, the two instances of the TSS can negotiate possible place to meet.

Cross-referencing Many information services provide data pertinent only to a single topic (e.g. Booking.com provides information about hotels) and their users have to cross-reference multiple sources of information. Therefore, the system should be able to combine different sources of information, and respond to more sophisticated queries.

Online data storage and accessibility of the user profile from multiple devices Due to the fact that the TSS is targeted mainly for mobile devices (see, scenario 1 and 2), the following issues have to be taken into consideration: (a) serious risk of data loss caused by accidents while traveling (for example, losing or damaging the device), and (b) need of copying the user profile from one device to another. In the first case, we should recognize time that the system spends learning user preferences (creating a user profile). Total loss of such profile would mean that the system would have to start from scratch, which is highly undesirable. The second problem can be easily solved by connecting the device with an old user profile to one with the newest version and simply copying the profile. However, if the user forgets to copy his profile from one device to another, the system may not find the best results, with respect to her current preferences. The solution to these two problems is in online storage and synchronization. The user should be able to enable backing-up of her current profile to a remote server (e.g. to a cloud)

and uploading it whenever the current device has an obsolete profile.

Obviously, due to the limited space, we cannot address all issues described above. Therefore we have decided to focus on crucial two. First, we will briefly discuss the general agent structure of the new TSS. Second, we will describe in detail how we represent data in the system and how this data is used to make recommendations.

4. Software agents and the Travel Support System. As stated above, the new TSS expands upon the ideas presented in [19]. One of the key features introduced in the original Travel Support System was a comprehensive application of software agents [24]. However, when the resulting system was thoroughly analyzed (in [28]), several design flaws concerning usage of agents were elaborated. Let us summarize them:

- Software agents should not be used as a middleware solution, unless it is beneficial.
- Software agents should not replace currently existing technologies, unless there is a specific need for this.
- Designing an agent-based system requires finding balance between the autonomy of an agent and its need to communicate.

Unfortunately, these problems were not fixed, while the original TSS became obsolete due to software evolution (including collapse of some projects, like the Racoon), leading to unresolvable software dependencies. Therefore, a fresh start was needed for use of agents in the TSS.

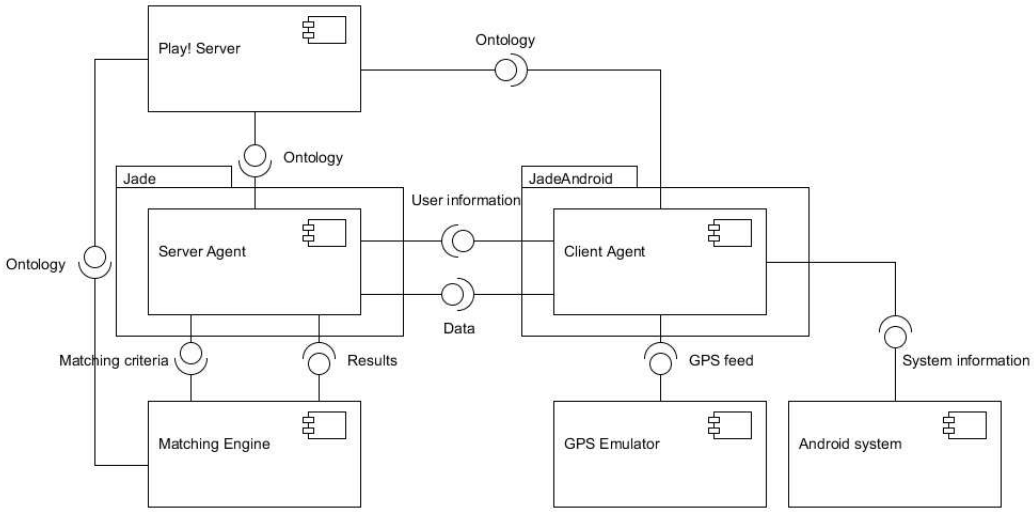

FIG. 4.1. *The component diagram of the new Travel Support System*

Figure 4.1 presents the component diagram of the proposed Travel Support System. Since one of our main assumptions was that the system will involve mobile technology, we have designed the TSS accordingly. As we can see, it is composed of two types of software agents: 1) *Client Agents*, and 2) *Server Agents*. These two types of agents perform different (and disjoint) tasks, while cooperating in order to support users.

The Client Agent runs on the mobile device (in our case on an Android-based system). First of all, the Client Agent manages the user profile on the (mobile) device. During the start-up of the system, the user profile is loaded from the server side of the system into the memory of the mobile device. Here, the agent receives an I/O interface, through which it can read and appropriately modify local user preferences.

The Client Agent is also responsible for backing-up the user profile and synchronizing it with the on-line server (the one, which runs the Server Agent). Obviously, the Client Agent monitors the user context. Since it is located on the mobile device it has access to the user data stored there. Here, let us stress, that the user is informed about this fact during the installation of the application; similarly to other applications running on the Android-based systems. The private user data includes text messages, entries in the calendar, contacts, and so on. Furthermore, the Client Agent has access to the GPS data, and the GIS software that provide the geo-spatial context.

To provide user with recommendations, the Client Agent communicates with the Server Agent. Here, we have decided that, even though the mobile devices have ever increasing power, their battery longevity and the complexity of the recommendation algorithm require that the actual data searching, cross-referencing and filtering should be implemented as a server-based solution. During communication, the Client Agent provides the Server Agent with information required to construct the search criteria, e.g. fragment of the user profile corresponding to the search query and the user contextual information (GPS coordinates and the current time). The user profile fragment contains all information related to a specific topic (for example, restaurants).

The actual search criteria are constructed by the Server Agent. Then, the Server Agent uses them in the local instance of the recommendation algorithm. The algorithm processes the search criteria and matches them against the available data sources. As a result, a set of most suitable recommendations is sent to the Server Agent. Then, the Server Agent communicates with the Client Agent who will presents the recommendation to the user.

5. Data representation in the system. Knowing the use case scenarios, the general idea behind the system, and the role of software agents, we have to consider how the information about travel services and user profiles is going to be represented. In the original TSS ([19]), ontologies represented in the RDF format [15] were used. Therefore, we have decided, for the time being, to re-use the already existing ontologies; in particular, the restaurant ontology. The assumption was that, as the system develops, a more comprehensive ontology of the world of travel will be incorporated (and/or developed if needed).

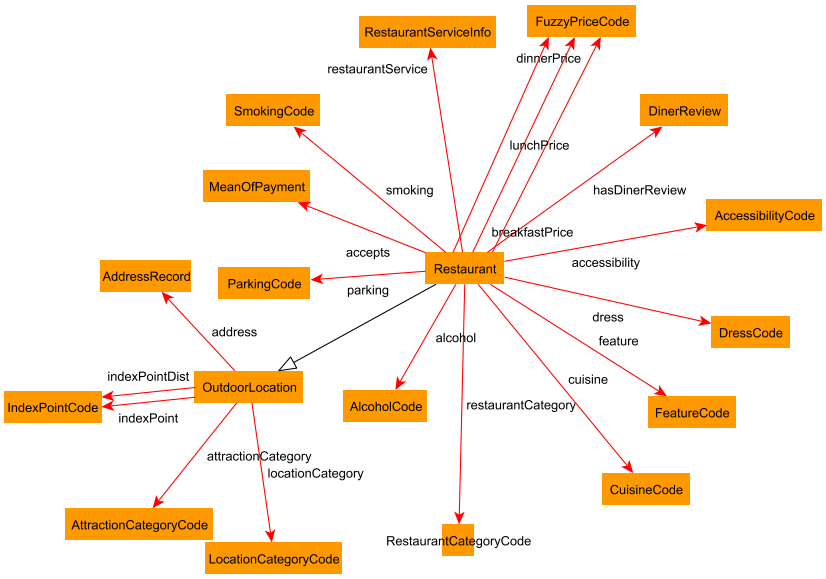

FIG. 5.1. *The structure of the Restaurant ontology*

The *Restaurant Ontology*, presented in Figure 5.1, contains the *Restaurant* class, which captures relations between the concept of the restaurant and its features. Restaurants and restaurant features are represented in the ontology as individuals (see, figure 5.1 for more details).

However, when deciding how to store the user profile, we have decided to use a simplified representation. Instead of adding additional elements to the ontology, we store pairs $(keyword, weights)$. Here, the keyword is an information object related to the *Restaurant* class, while the weight represents user preferences. Proceeding in accordance with the original TSS, we have decided to represent user preferences as numbers from the interval $[0, 1]$. This approach is based on the observation that, not only the Restaurant Ontology has few concepts, but also that not all of them have to have actual weight associated with them. For instance, if the system does not have any notion about users' opinion about Chinese cuisine, no user-specific weight can be associated with it

(and a neutral value of 0.5 has to be used). As a result, keywords for which no opinion is present do not need to be stored in the user profile.

Observe that this representation simplifies somewhat the process of preparing the recommendation, as it is relatively easy to access weights (for example, by using a hash mapping) and modify them if some adjustments are needed. Taking this into account, since Jack (from use case scenario 1) likes vegetarian food, his user profile could consist of the following pairs:

1. (*cur:Vegetarian*, 0.91)
2. (*cur:Indian*, 0.65)
3. (*cur:SteakHouses*, 0.04)
4. (*asc:SmokingArea*, 0.3)
5. (*drs:Casual*, 0.66)
6. (*rcc:FastFoodRestaurant*, 0.1)
7. (*rcc:CasualRestaurant*, 0.7)

Note that the profile reflects Jack's preferences by assigning appropriate weights to the "Vegetarian" (0.91; the largest weight) and the "SteakHouse" (0.04) cuisines. Thus the TSS system, working on Jack's mobile device will strongly favor vegetarian food over the other types of food. In addition, the steak house has the lowest priority in comparison with other cuisines (including those not listed in the Jack profile that will have the default weight – 0.5).

Finding the most suitable recommendation for Jack is going to be preformed by matching his user profile with profiles of potential restaurants. This operation is very similar to the item-to-user recommendation performed by many modern recommender systems [32]. However, this task is more complex, due to the data representation utilized in the TSS. Specifically, ontologies cover more in-depth relations between objects than the popular recommendation algorithms usually exploit. Hence, in this paper we describe, in-depth, the approach selected to make the ontology-based recommendations.

6. The semantic matchmaking algorithm. As discussed above, the central functionality of the Travel Support System is its ability to make recommendations. Since, in the TSS, all travel information is ontologically represented, recommendations can be seen as the realization of the problem of establishing semantic closeness between entities. Specifically, assuming that users' restaurant preferences are represented in his user profile, and all restaurants are instances of an ontology, the question that has to be answered is: which of the available restaurants matches the user profile the closest. Of course, since the TSS is also going to use geopositioning, an important filter may be the location. Thus the system should also be able to answer the question: which of the restaurants at a distance not larger than 1 km is matching a given user profile the closest.

To be able to respond to such queries (to establish "semantic" distance between entities), we have selected the Rhee-Ganzha algorithm [31]. This algorithm was originally proposed in 2009, but it has never been fully tested for a realistic ontology. Therefore, its implementation and application in the TSS will also allow us to establish its usability.

6.1. General description of the Rhee-Ganzha algorithm. Let us start from the general description of the algorithm. It is based on two observations. First, ontologies can be seen as a directed graph. Second, in the knowledge space, information resources do not exist in isolation from each other, but are connected through multiple relations. Such relations can be explicit or inferred. Thus, the existence, the strength, and the number of relations between two resources determine a degree of their semantic similarity (closeness). This, in turn, can be translated into a problem of establishing existence, and finding number and strength, of connections in a directed graph (on the way from the source – user, to the sink – travel entity). However, as it will be seen, this is not a simple network-flow problem, where the total throughput between the source and the sink is sought.

Obviously, the semantic closeness between the user (profile) and travel services can be used to deliver recommendations to the user. Namely, if the value of the semantic closeness between the user and the restaurant A is 0.75, while for the restaurant B it is 0.3, then the restaurant A is much more likely to match her desires. Moreover, semantic closeness can be also seen as a "degree of relatedness" between two information objects, or "how relevant one object is to another." Specifically, if two individuals are in many explicit or inferred relations, we can say that they are relevant to each other. Thus the proposed algorithm can help determine the level of semantic closeness (or relevance) to entities, which are quite different. For example, if the user likes restaurants

with smoking areas and free WiFi, then the system could use this information to: (a) create an alternative recommendation, which matches only some user preferences, or (b) recommend hotel on the basis of its having features (smoking area and free WiFI) that are liked by the user in restaurants. Taking this into account, let us summarize main assumptions behind the algorithm for calculating semantic closeness.

6.2. Core assumptions of the semantic matchmaking algorithm. The relevance (semantic closeness) calculation is based on the following three assumptions:

1. Having more relations from one object to another means that they are closer (more relevant to each other).
2. Each relation may have different importance depending on the type of the relation.
3. Even if the relations are of the same type, the weight of the connection can vary between individual objects (instances).

Since these assumptions are often referenced in the remaining parts of the paper, let us elaborate them. The first assumption is very intuitive. For example, let us consider three restaurants A, B and C. While, restaurants A and B allow smoking, restaurant C does not provide a designated smoking area. Therefore, it is intuitive that (in this context) restaurants A and B are semantically closer to each other than to the restaurant C.

The second assumption means that relations are weighted according to their semantic importance. Let us now consider two aspects of a restaurant: availability of *live music* and type of *cuisine*. In general, we can assume that the cuisine is more important than the presence of live music. Henceforth, assuming that a tourist likes live music, we still can assume that he will probably choose a restaurant serving his favorite food, while without live music; rather than a restaurant, which serves the food that he does not like, but where he can listen to live music. Of course, the tourist can do otherwise (e.g. when the live music is his favorite blues band), but this still illustrates difference in importance of specific features of travel entities. To express such differences within the actual system, each relation is described by a weight representing its importance. Such weight can be also referred as a relevance value because it states how relevant to each other are the two objects connected by a given relation.

The third assumption states that connections (of the same type) between two individual objects can have different weights. This situation can be illustrated by the following observation: both, an "upper class" Chinese restaurant and a "low-quality" Chinese fast-food, serve meals demarcated as *Chinese cuisine*. However, the quality of meals in these restaurants differs significantly. Note that the *cuisine* relation has the same importance for both restaurants (assumption two). Thus, an additional weight has to be introduced, which describes user preferences with respect to an individual information object (e.g. one tourist may not care what is the type of the restaurant, while another may not want to eat in fast-food places).

6.3. Relevance calculation process. Based on these assumptions, and following the description found in [31], we can now present the the core component of the algorithm finding semantic closeness between two objects. Let us start from defining the *matching criterion*, which is an ordered quadruple $< x, q, a, g >$, where:

x is a source object,

q is a query which defines a subset of objects that are considered potentially relevant; these objects will be matched against the source object x,

$a \in (0; \infty)$ specifies the threshold of closeness between objects to be judged actually relevant to each other,

g is a sub-query which is used to optimize the matching process by reducing the number of considered nodes.

Calculation of the semantic closeness proceeds in two stages (1) Graph Generation, and (2) Relevance Calculation. In the first stage, an ontology is used to create a graph representing connections between the user (profile) and the travel entities. This graph is used, in the second phase, to find all direct and indirect relations between the source and the sink individuals. Therefore, the second stage has as its input the graph corresponding to the ontology, the source object and a set of target entities. Here, the algorithm calculates semantic closeness between the source and each target object separately. Let us briefly discus each stage (for the detailed description, see [31]).

6.3.1. Graph Generation. A graph generated in the first stage is called a *Relevance Graph*. The Relevance Graph is a directed label graph $G = (V, E)$, where:

- V is a set of nodes (a set of individual information objects),
- E is a set of edges (a set of relations).

Observe that the structure of the graph does not restrict the number of edges between two adjacent nodes. Therefore, if two information objects are connected by multiple relations, the corresponding two nodes (in the relevance graph) will also be connected by the same number of edges. Moreover, the relevance graph can contain cycles.

To optimize the node generation method, in the Travel Support System, a relevance graph is to be built only from the information objects, which are relevant with respect to the contextual information. This means that the algorithm will process only a small part of the ontology. Specifically, in the current implementation, the algorithm does not need to include information objects exceeding a certain distance from the user (which are filtered-out first). For example, a maximum distance can be set to 2 km for walking and 50 km for a car drive.

Edges in the relevance graph are generated based on the relations between objects in the ontology. Namely, each edge is defined as $e \in E = (x, y, distance, weight)$, where

- $x \in V$ is the tail node of the edge e,
- $y \in V$ is the head node of the edge e,
- $distance \in N$ is the conceptual distance value (see, the second assumption),
- $weight \in (0; 1) \in R$ is the individual weight value (see, the third assumption).

Finally, the distance between the two adjacent nodes is expressed by the formula

$$Distance = \frac{1}{Relevance}, \tag{6.1}$$

where the definition of the *Relevance* is based on how relevant are the two objects connected by a relation instantiated through the given edge. Note that, these values can be determined by a domain expert. Namely, a restaurant reviewer (the domain expert in the subject of restaurants) can adjust relevance values in such a way that the "cuisine" relation is more important than the "parking" relation, by assigning them the relevance values equal to 0.7 and 0.2 respectively. In practice, when the expert is not involved, all distance values should be initialized to a single value (e.g. distance = 1). In such case, all calculations will be performed based solely on the user profile (and the structure of the ontology). For the non-adjacent nodes, the distance value is going to be calculated in a way presented in the next section.

6.3.2. Relevance Calculation. The relevance calculation starts from "simplifying" structure of edges in the relevance graph. Here, we distinguish two operations: edge scaling and edge merging. Edge scaling individualizes each edge's distance by multiplying its relevance value by its weight. This operation is a consequence of the third assumption. Equation 6.2 presents a formula used to calculate a scaled relevance value for an edge $e = (x, y, Distance, weight) \in E$.

$$\begin{aligned} NewRelevance &= oldRelevance \times weight \\ NewDistance &= \tfrac{1}{NewRelevance} \end{aligned} \tag{6.2}$$

Since the structure of the relevance graph is restricted only by the ontology that it represents, there can be multiple edges between two adjacent nodes. The original publication suggested applying the following edge merging formula:

$$NewDistance = (\sum_{i=1}^{n} \frac{1}{Distance_i})^{-1} \tag{6.3}$$

After these operations, nodes in the Relevance Graph are connected by a single weighted edge. This edge represents semantic closeness of the two adjacent nodes (with respect to a given property).

Let us now consider the semantic closeness of non-adjacent nodes. First of all, the algorithm finds all valid paths between two non-adjacent nodes (the source node and one of the target nodes). A path is valid if and only if it is a simple path (namely, it does not contains repeating nodes). If there is no valid path between two non-adjacent nodes, it means that they are not related, hence their relevance is equal to 0 (their distance is equal to infinity). If there exists one or more valid paths, first, the algorithm calculates length of each of them by applying the following formula:

$$Distance_P = \sum_{i=1}^{n-1} i \times Distance_{a_i a_{i+1}} \tag{6.4}$$

Next, the algorithm calculates the cumulative relevance of all paths between the source node and the target node. Specifically, for n paths $P_1, P_2, \ldots, P_n$ from $x \in V$ to $y \in V$ the relevance value $Relevance_{xy}$ is defined as follows

$$Relevance_x y = \sum_{i=1}^{k} \frac{1}{Distance_{P_k}} \tag{6.5}$$

The $Relevance_x y$ represents the semantic closeness of two objects. This value is then to be used to select and rank objects to be recommended to the user.

7. Implementing the Rhee-Ganzha algorithm. The critical part of the Rhee-Ganzha algorithm is to determine all simple paths between two nodes (process of scaling and merging multiple path can be achieved by applying equations 6.2 and 6.3). In the original work [31] no details concerning possible / actual implementation were provided. Therefore, after some investigations, we have decided use the Yens algorithm [33] for finding k-shortest simple paths as the centerpiece of our implementation.

7.1. Yens algorithm. Yen's algorithm [33] is a deviation algorithm that finds k single-source shortest loopless paths in a graph with non-negative edge weights, by deviating the initial one found by another algorithm (for example, the Dijkstra's algorithm). Let us consider the k-th shortest path $p_k = \{{v_1}^k, {v_2}^k, \ldots, {v_n}^k\}$. In order to find the $(k+1)$-th shortest path, the algorithm analyzes every node ${v_i}^k$ in p_k and computes the shortest loopless path p, which deviates from the p_k at this node. The loopless path p_k is said to be a parent of p, and ${v_i}^k$ is its deviation node. To avoid loops during the calculation of a new path, the algorithm should remove all sub-paths between the source node and a deviation node ${v_i}^k$. Therefore, all such nodes are temporarily removed and the algorithm calculates a new path in the resulting graph.

Our implementation of the Yen's algorithm utilizes its new implementation, described in [30]. This implementation was proved to be generally more efficient than the other implementations (the publication covers the straightforward implementation and the implementation proposed by Perko). Specifically, the new implementation replaces the node deletion with its reinsertion. This allows to solve the problem faster, by labeling and correcting labels of some nodes. In general, the number of corrected nodes is assumed to be smaller than the total number of nodes in a graph. The tests performed by the authors of the article seem to prove this hypothesis. However, let us make it clear that, in the worst case scenario, the modified algorithm performs its computation in $O(Kn(m + n \log n))$, where K is the number of paths, n is the number of nodes and m is the number of edges. This time complexity is also the worst-case time complexity of the straightforward implementation.

Let us now describe the implementation of the Yen's algorithm that was used in our system. A shortest path tree rooted at a vertex x is a spanning tree T_x of a graph G. If x is the terminal node, then T_x represents the tree of the shortest paths from every node to x. Otherwise, T_x represents the tree of the shortest paths from x to every node. A loopless path from $v \in V$ to x in T_x is denoted by $T_x(v)$. The cost of $T_x(v)$ is denoted by π_v and is also used as a label of v. Taking this into account, the Yen's algorithm can be summarized as follows.

Yens Algorithm

Input:

1. G - a graph with the source and the target nodes
2. K - the number of shortest path to be found

Output: K the shortest paths

initialize an array *deviation*, which stores derivation nodes for all shortest paths
p = shortest path from the source to the target nodes in the graph G found by another algorithm (e.g. the Dijkstra's algorithm, see below)
$deviation[p] = s$
$X = \{p\}$
$k = 0$
while $X \neq \emptyset$ and $k < K$ **do**
  $k = k + 1$
  p_k = shortest, loopless path in G
  $X = X \setminus \{p_k\}$
  $\pi_v = \infty$, for any $v \in V$
  remove loopless path p_k, except target node t, from the graph G
  remove edges $(deviation(p_k), v), v \in V$ of $p_1, p_2, ..., p_k$
  T_t = shortest tree rooted at t
  for $v_i^k \in \{v_{l_k}^k, ..., d(p_k)\}$ **do**
    restore node v_i^k in the graph
    calculate $\pi_{v_i^k}$ (label) using forward star form
    if $\pi_{v_i^k}$ is defined **then**
      correct labels of v_i^k successors using backward star form
      $p = sub(s, v_i^k) \cdot T_t(v_i^k)$
      $deviation(p) = v_i^k$
      $X = X \cup \{p\}$
    end if
    restore (v_i^k, v_{i+1}^k) in the graph
    if $\pi_{v_i^k} > \pi_{v_{i+1}^k} + cost_{v_i^k, v_{i+1}^k}$ **then**
      $\pi_{v_i^k} = \pi_{v_{i+1}^k} + cost_{v_i^k, v_{i+1}^k}$
      correct labels of v_i^k successors using backward star form
    end if
  end for
  Restore the graph G
end while

Forward star formation
Correct labels of x successors
Input: a graph G

for edge $e = (i, j) \in E$ **do**
  if $\pi_i > \pi_j + cost_{i,j}$ **then**
    $\pi_i = \pi_j + cost_{i,j}$
  end if
end for

Backward star formation
Correct labels of x successors
Input: a graph G and a vertex x

$list = \{x\}$
repeat
  i = element of $list$
  $list = list \setminus \{i\}$
  for edge $e = (i, j) \in E$ **do**

if $\pi_i > \pi_j + cost_{i,j}$ **then**
$\pi_i = \pi_j + cost_{i,j}$
$list = list \cap \{j\}$
end if
end for
until $list \neq \emptyset$

In each iteration, the Yen's algorithm deviates from the previously found path to generate the new one. During this process, starting from the source node as a deviation node, the Yen algorithm calls another algorithm, which finds the shortest loopless path from the deviation node to the target. Based on the suggestion found in [33], we have decided to use the Dijkstra's algorithm to implement this step.

7.2. Dijkstras algorithm. The Dijkstras algorithm [26] is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path weights. It is also used to produce a shortest path tree. For a given source node in the graph, the algorithm finds a path with the lowest cost (the shortest path). Such path is used by the Yen's algorithm to generate the remaining part from the deviation node to the target. The algorithm proceeds as follows:

Dijkstras algorithm
Constructs an array of nodes which can be used to reconstruct the shortest path
Input: a graph G, the *source* and *target* nodes
Output: an array of nodes which can be used to reconstruct the shortest path

initialize an array *distance*, which stores shortest distances between the *source* and other nodes in G
initialize an array *previous*, which stores previous nodes in optimal path
for each vertex v in the graph G **do**
distance from the source to v (denoted by $dist[v]$) is equal to infinity
the previous node in the optimal path from the source to v (denoted by $previous[v]$) is undefined
end for
$distance[source]$ is equal to 0
initialize a priority queue Q, which contains all the nodes
while Q is not empty **do**
let u be a vertex in Q with the smallest distance from the source
remove u from Q
if $distance[u]$ is equal to infinity **then**
break
end if
for each neighbour v of u **do**
calculate an alternative distance $alt = distance[u] + distance(u, v)$
if alt is smaller than $distance[v]$ **then**
$distance[v] = alt$
$previous[v] = u$
$reordervinQ$
end if
end for
end while
return *previous*

The shortest path can be reconstructed by executing the following algorithm:
Reconstruct
Reconstructs the shortest path from an array of nodes.
Input: the previous array of nodes constructed by Dijkstras algorithm, the target node
Output: a sequence containing the shortest path

let S be an empty sequence
u is equal to the target

while *previous*[u] is defined **do**
  insert u at the beginning of S
  $u = previous[u]$
end while
return S

This algorithm can yield different performance depending on the implementation of the priority queue. Keeping in mind that the algorithm may be used with graphs built on the basis of large ontologies, the Dijkstra algorithm should operate in the shortest possible time. Therefore, to achieve the lowest time complexity, the decision was made to utilize the Fibonacci heap. This improves the time complexity of the algorithm to $O(|E| + |V| \log |V|)$, where E is a set of edges, V is a set of vertices and $|.|$ denotes the size of a set.

7.3. Fibonacci heap. The Fibonacci heap, used the Dijkstra's algorithm, is implemented according to [27]. A Fibonacci heap is a collection of item-disjoint heap-ordered trees. A heap-ordered tree is a rooted tree containing a set of items arranged in the heap order. Namely, if x is a node, then its key is no less than the key of the parent (provided that x has a parent). Thus, the root contains an item with the smallest key. There is no explicit constraint on the number or structure of the trees. However, the number of children of a node represents its rank. The rank of a node with n descendants is $O(\log n)$. The heap-ordered trees can perform an operation called linking. The linking operation combines two item-disjoint trees into one.

The heap is accessed by a pointer to its root, which contains an item with the smallest key. We can also call this root the minimum node. If the minimum node is undefined, the heap is empty. Each node contains a pointer to its parent, another pointer to one of its children and its rank. The children of each node are doubly linked in a circular list. This helps the heap maintain the low cost of its operations. For example, the double linking between a root and its children makes removing elements possible in $O(1)$. Similarly, the circular linking between children makes concatenation of two such lists possible in $O(1)$.

Since the Fibonacci heap will be used as a priority queue, the most important operations are the *delete_min* operation and the *decrease_key* operation. The *decrease_key* operation starts by subtracting a given number from the *key* of an item i in a heap h. Secondly, the algorithm finds a node x containing i and cuts the edge joining x to its parent (it also decreases the rank of the parent). As a result, the algorithm creates a new sub-tree rooted at x. If the new key of i is smaller than the key of the minimum node then the algorithm redefines the minimum node to be x. The complexity of the *decrease_key* operation is $O(1)$ (this operation assumes that the position of i in h is known).

The *delete* operation is similar to the *decrease_key* operation. First, the algorithm finds the node x containing i and cuts the edge with its parent node. Second, it concatenates the list of children of x with the list of roots and destroys x. The complexity of the delete operation is $O(1)$ (this operation assumes, again, that the position of i in h is known).

The *delete_min* operation requires finding pairs of tree roots of the same rank to link. This is achieved by using an array indexed by ranks. After deleting the minimum node and forming a list of new tree roots, the algorithm inserts the roots, one by one, into the array. If the root is inserted into an array position which is already occupied, then the algorithm performs the linking operation on the roots in conflict. After successful linking, the resulting tree is inserted to the next higher position. The *delete_min* operation ends when all the roots are stored in the array and its amortized time is equal to $O(\log n)$.

8. Matching Process Examples. Let us now illustrate the matching process on the basis of the two use case scenarios presented in Section 3. In both cases, the algorithm utilizes the Restaurant ontology described in Section 5. In the first example, we will perform relevance calculations based on the following matching criteria from the first use case scenario – "finding a vegetarian restaurant."

1. $x = User\_Profile\_Restaurant$
2. $q =$
   $PREFIX onto : \langle http://localhost:9000/assets/Ontologies/Restaurant/Restaurant\rangle$
   $SELECT ?restaurant$
   $WHERE ?restaurant is a onto : Restaurant.$
3. $a = \frac{3}{5}$

4. $g = [lat = 22.2, long = 44, dist = 2000]$:
   lat = user's latitude
   $long$ = user's longitude
   $dist$ = the maximum distance (in meters) between the source (the user's current position) and a target object

Note that the source object x, *User_ Profile_ Restaurant*, does not exist in the ontology. It is an information object created on the basis of the user profile. Recall, the currently we apply a simplified representation of user profiles. For example, the source object corresponding to the user profile presented in section 5 has the following relations:

1. *res:User_ Profile_ Restaurant res:cuisine cui:Vegetarian*
2. *res:User_ Profile_ Restaurant res:cuisine cui:Indian*
3. *res:User_ Profile_ Restaurant res:cuisine cui:SteakHouses*
4. *res:User_ Profile_ Restaurant res:smoking acs:SmokingArea*
5. *res:User_ Profile_ Restaurant res:dress drs:Casual*
6. *res:User_ Profile_ Restaurant res:restaurantCategory rcc:FastFoodRestaurant*
7. *res:User_Profile_Restaurant res:restaurantCategory rcc:CasualRestaurant*

Then, to determine a set of possibly relevant target objects, the matching criteria defines the q query and g sub-query (see, section 6.3). In our example, the user is interested in restaurants. Hence, the q query accepts all individuals, which are instances of the *Restaurant* class. However, the q query does not determine, which information objects are located within a certain distance from the user (here, a sample value $dist = 2000$). Therefore, the g query removes all places located further than a given distance. By combining these two queries, the algorithm is able to obtain all restaurants located not further than 2 km from the user. In what follows, relevance calculations will be performed for sample restaurants *JoesSteakHouse*, *ThaiIndiaRestaurant* and *LuckyChefRestaurant*. Therefore, the algorithm calculates the semantic closeness between the source (Jack's profile) and the three target objects found in the previous step. The matching process involves:

1. source instance URI = *User_Profile_Restaurant*
2. target objects URI's = [*Joe's Steak House*, *ThaiIndiaRestaurant*, *LuckyChef Restaurant*]
3. relevance threshold: $a = \frac{3}{5}$

The relevance calculation starts from creating a *Relevance Graph* (described in section 6.3), which corresponds to the Restaurant Ontology depicted in figure 5.1. Since the resulting graph does not have multiple relations between any two adjacent nodes, the system skips the edge merging procedure. Now, let us recall that different features described by the Restaurant ontology can have different individual weights stored in the user profile. For example, *Indian* and *SteakHouses* cuisines have their individual weights equal to 0.8 and 0.45 respectively. Therefore, the algorithm needs to scale the edges by applying equation 6.2. Recalling our discussion in section 6.3.1, currently all edges have their initial distances equal to $d = 1$. Therefore, the calculations proceed as follows:

$$distance_{Vegetarian} = (\frac{1}{d_{cuisine}} \times w_{Vegetarian})^{-1} = (1 \times 0.91)^{-}1 = 1.1$$

$$distance_{Indian} = (\frac{1}{d_{cuisine}} \times w_{Indian})^{-1} = (1 \times 0.65)^{-}1 = 1.54$$

$$distance_{SteakHouses} = (\frac{1}{d_{cuisine}} \times w_{SteakHouses})^{-1} = (1 \times 0.04)^{-}1 = 25$$

Recall that if the user profile does not specify the weight for an information object, then a default weight is applied (here, $w = 0.5$). After creating the appropriate *Relevance Graph*, the algorithm calculates relevance values establishing relations between the source (*User_Profile_Restaurant*) and the target objects (*ThaiIndiaRestaurant*, *Joe's Steak House* and *LuckyChefRestaurant*):

ThaiIndiaRestaurant

The system utilizes the Yen's algorithm (described in section 7.1) to find all simple paths. Based on the *Relevance Graph*, we can define four paths from the *User_Profile_Restaurant* to the *ThaiIndiaRestaurantRestaurant*:

Path $1: User\_Profile\_Restaurant \rightarrow Indian \rightarrow ThaiIndiaRestaurant$
Path $2: User\_Profile\_Restaurant \rightarrow SmokingArea \rightarrow ThaiIndiaRestaurant$
Path $3: User\_Profile\_Restaurant \rightarrow Casual \rightarrow ThaiIndiaRestaurant$
Path $4: User\_Profile\_Restaurant \rightarrow CasualRestaurant \rightarrow ThaiIndiaRestaurant$

By applying equation 6.4, the respective distance values are:

$D_{Path1} = 1.1 + (2 \times 1.1) = 3.75$
$D_{Path2} = 3.33 + (2 \times 3.33) = 10.0$
$D_{Path3} = 1.52 + (2 \times 1.52) = 4.56$
$D_{Path4} = 1.43 + (2 \times 1.43) = 4.29$

Finally, the algorithm obtains the total relevance value, from equation 6.5:

$$Rel_{ThaiIndiaRestaurant} = \frac{1}{3.75} + \frac{1}{10.0} + \frac{1}{4.56} + \frac{1}{4.29} = 0.8191$$

JoesSteakHouseRestaurant

All simple paths between the *User_Profile_Restaurant* and the *JoesSteakHouseRestaurant* are:

Path $1: User\_Profile\_Restaurant \rightarrow SteakHouse \rightarrow JoesSteakHouseRestaurant$
Path $2: User\_Profile\_Restaurant \rightarrow SmokingArea \rightarrow JoesSteakHouseRestaurant$
Path $3: User\_Profile\_Restaurant \rightarrow Casual \rightarrow JoesSteakHouseRestaurant$
Path $4: User\_Profile\_Restaurant \rightarrow CasualRestaurant \rightarrow JoesSteakHouseRestaurant$

By applying equation 6.4 we obtain:

$D_{Path1} = 25 + (2 \times 25) = 75.0$
$D_{Path2} = 3.33 + (2 \times 3.33) = 10.0$
$D_{Path3} = 1.52 + (2 \times 1.52) = 4.56$
$D_{Path4} = 1.43 + (2 \times 1.43) = 4.29$

Next, by applying equation 6.5 we calculate the total semantic closeness as:

$$Rel_{JoesSteakHouseRestaurant} = \frac{1}{75.0} + \frac{1}{10.0} + \frac{1}{4.56} + \frac{1}{4.29} = 0.5657$$

LuckyChefRestaurant

All simple paths between the *User_Profile_Restaurant* and the *LuckyChefRestaurant* are:

Path $1: User\_Profile\_Restaurant \rightarrow Vegetarian \rightarrow LuckyChefRestaurant$
Path $2: User\_Profile\_Restaurant \rightarrow Casual \rightarrow LuckyChefRestaurant$
Path $3: User\_Profile\_Restaurant \rightarrow CasualRestaurant \rightarrow LuckyChefRestaurant$

By applying equation 6.4 we obtain:

$D_{Path1} = 1.1 + (2 \times 1.1) = 6.69$
$D_{Path3} = 1.52 + (2 \times 1.52) = 4.56$
$D_{Path4} = 1.43 + (2 \times 1.43) = 4.29$

Next, by applying equation 6.5 we establish the semantic closeness to be equal to:

$$Rel_{LuckyChefRestaurant} = \frac{1}{3.3} + \frac{1}{4.56} + \frac{1}{4.29} = 0.7554$$

Based on these results, and the proposed threshold value $a \geq \frac{3}{5}$, the *LuckyChefRestaurant* (0.7554) and the *ThaiIndiaRestaurant* (0.8191) can be recommended to the user. Since the relevance value of the *JoesSteakHouseRestaurant* (0.5657) is below the threshold, the algorithm will not include this recommendation in the results. The ranking of the recommended restaurants is also possible and will place the *ThaiIndiaRestaurant* as the overall winner.

In the second example, we assume that Jack would like to invite Jill to a restaurant (second use case scenario). Here, the Travel Support System can prepare a list of the most suitable restaurants. To simplify the calculations, let us assume that Jack will invite Jill to one of the restaurants for which we have just completed calculation of their semantic closeness to his preferences. Therefore, we know that the two restaurants that are the suitable for him are: *LuckyChefRestaurant* and *ThaiIndiaRestaurant*. Thus, his system starts negotiations with Jill's instance of the Travel Support System by sending a message containing these two restaurants as an initial proposal. Based on the description of the problem, the relevance calculations seen from Jill's perspective

can be expressed as:

1. source instance URI = *JillsRestaurant*
2. target objects URI's = [*LuckyChefRestaurant*, *ThaiIndiaRestaurant*]
3. relevance threshold: $a = \frac{3}{5}$

Let us now assume that Jill's user profile is as follows:

1. (*cur:Vegetarian*, 0.91)
2. (*cur:Indian*, 0.65)
3. (*cur:SteakHouses*, 0.04)
4. (*asc:SmokingArea*, 0.3)
5. (*drs:Casual*, 0.66)
6. (*rcc:FastFoodRestaurant*, 0.1)
7. (*rcc:CasualRestaurant*, 0.7)

Then, the semantic closeness information object corresponding to her profile and the restaurants proposed by Jack is:

ThaiIndiaRestaurant

All simple paths between *JillsRestaurant* and the *ThaiIndiaRestaurant* are:

Path 1 : $JillsRestaurant \rightarrow Indian \rightarrow ThaiIndiaRestaurant$
Path 2 : $JillsRestaurant \rightarrow SmokingArea \rightarrow ThaiIndiaRestaurant$
Path 3 : $JillsRestaurant \rightarrow Casual \rightarrow ThaiIndiaRestaurant$
Path 4 : $JillsRestaurant \rightarrow CasualRestaurant \rightarrow ThaiIndiaRestaurant$

By applying equation 6.4, the respective distance values are:

$D_{Path1} = 1.18 + (2 \times 1.18) = 3.54$
$D_{Path2} = 1.67 + (2 \times 1.67) = 5.00$
$D_{Path3} = 3.34 + (2 \times 3.34) = 10.0$
$D_{Path4} = 1.43 + (2 \times 1.43) = 4.29$

Finally, the algorithm obtains the total semantic closeness value by utilizing equation 6.5:

$$Rel_{ThaiIndiaRestaurant} = \frac{1}{3.54} + \frac{1}{5.0} + \frac{1}{10.0} + \frac{1}{4.29} = 0.7156$$

LuckyChefRestaurant

All simple paths between *JillsRestaurant* and the *LuckyChefRestaurant* are:

Path 1 : $JillsRestaurant \rightarrow Vegetarian \rightarrow LuckyChefRestaurant$
Path 3 : $JillsRestaurant \rightarrow Casual \rightarrow LuckyChefRestaurant$
Path 4 : $JillsRestaurant \rightarrow CasualRestaurant \rightarrow LuckyChefRestaurant$

By applying equation 6.4 we obtain:

$D_{Path1} = 2 + (2 \times 2) = 6.0$
$D_{Path2} = 3.34 + (2 \times 3.34) = 10.0$
$D_{Path3} = 1.43 + (2 \times 1.43) = 4.29$

Finally, by applying equation 6.5 we can calculate the total semantic closeness:

$$Rel_{LuckyChefRestaurant} = \frac{1}{6.0} + \frac{1}{10.0} + \frac{1}{4.29} = 0.4998$$

Based on these results, and the proposed matching criteria ($a \geq \frac{3}{5}$), where a is the relevance threshold, only the *ThaiIndiaRestaurant* will be recommended to Jill. Because the relevance value of *LuckyChefRestaurant* is below the threshold, the algorithm will not include this recommendation in the results. Therefore, Jack will be informed that he should propose to Jill the *ThaiIndiaRestaurant* restaurant (since it matches their preferences at least at the minimal level).

9. Concluding remarks. The aim of this paper was twofold. First we have discussed the main assumptions leading to the new version of an agent-semantic travel support system. Second, we have shown how the Rhee-Ganhza algorithm can be used to establish semantic closeness between the user profile and the restaurant objects. The system has been implemented and works on Android-based mobile devices. As a matter of fact, the results illustrating its work, presented in section 8, have been prepared on the basis of the working system. The next step will be to extend the travel ontology to be able to test the proposed approach on more complex ontologies, while asking more sophisticated queries.

REFERENCES

[1] *A personal agent application for the semantic web*. http://www.csee.umbc.edu/~finin//papers/aaaiss03.pdf.
[2] *Applying semantic search and ontology to the travel industry*. http://www.tnooz.com/2011/05/20/news/applying-semantic-search-and-ontology-to-the-travel-industry/.
[3] *Booking.com*. http://www.booking.com/.
[4] *Booking.com mobile application*. http://www.booking.com/general.html?sid=1b58219367c2cba11085678296d6347e;dcid=1;tmpl=docs%2Fiphone_landing&lang=en-gb.
[5] *Facebook*. https://www.facebook.com/.
[6] *Google Now*. http://www.google.com/landing/now/.
[7] *iPic Entertainment*. https://www.ipictheaters.com/default.aspx.
[8] *iPic Theaters mobile application*. https://play.google.com/store/apps/details?id=nz.co.vista.android.movie.ipic&feature=search_result#?t=W251bGwsMSwxLDEsIm56LmNvLnZpc3RhLmFuZHJvaWQubW92aWUuaXBpYyJd.
[9] *Open Directory Project*. http://www.dmoz.org/.
[10] *Openrice*. http://www.openrice.com/.
[11] *OpenRice mobile application*. https://play.google.com/store/apps/details?id=com.openrice.android&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5vcGVucmljZS5hbmRyb2lkIl0.
[12] *Pellucid (A Platform for Organisationally Mobile Public Employees)*. http://www.softeco.it/en/pellucid1.html.
[13] *Pellucid Project*. http://cordis.europa.eu/search/index.cfm?fuseaction=proj.document&PJ_RCN=5519298.
[14] Krawczyk K., Majewska M., Dziewierz M., Sota R., Balogh Z., Kitowski J., Lambert S., *Reuse of Organisational Experience Harnessing Software Agents*, Computational Science - ICCS 2004, Springer, 2004, pp. 583–590.
[15] *Resource Description Framework*. http://www.w3.org/RDF/.
[16] *Semantic Web*. http://www.w3.org/standards/semanticweb/.
[17] *Smart travel service advisor using Semantic Web and Agent Technology*. http://eprints.soton.ac.uk/266943/1/Travel_Advisor_paper.pdf.
[18] *Travel Support System publications*. http://www.ibspan.waw.pl/~paprzyck/mp/cvr/research/agents_TSS.html.
[19] *Travel Support System Sourceforge*. http://e-travel.sourceforge.net/.
[20] *TripAdvisor*. http://www.tripadvisor.com/.
[21] *Twitter*. https://twitter.com/.
[22] *Virtual consultant for public services, presentation in Polish)*. http://archiwum.spnt.pl/konwent2009/files/pdf/abg.pdf.
[23] *Virtual consultant for public services)*. http://www.um.radlin.pl/943,aaa.html.
[24] Maes P., *Agents that reduce work and information overload*, in Communications of the ACM Volume 37 Issue 7, 1994, p. 30-40.
[25] *Yelp!* http://www.yelp.com/.
[26] R. R. L. S. C. Cormen Thomas H., Leiserson Charles E., 2011.
[27] T. R. E. Fredman M. L., *Fibonacci Heaps and Their Uses in Improved Network*, Journal of the ACM, 1987.
[28] P. M. G. M. Gawinecki M., Kruszyk M., *Pitfalls of agent system development on the basis of a travel support system*, in Proceedings of the BIS 2007 Conference, Springer, 2007, pp. 488–499.
[29] A. Nauli, *Using software agents to index data of an e-travel system*, Master thesis, Oklahoma State University, 2000.
[30] M. E. Pascoal M., *A new implementation of Yens ranking loopless paths algorithm*, 4QR - Quarterly Journal of the Belgian, French and Italian Operations Research Societies, 2003.
[31] P. M.-W. S. M. F. G. G. M. P. M. Rhee S. K., Lee J., *Measuring semantic closeness of ontologically demarcated resources*, in Fundamenta Informaticae, 2009, pp. 395–418.
[32] S. B. Ricci F., Rokach L., *Introduction to Recommender Systems Handbook*, Springer, 2011.
[33] Y. J. Y., *An algorithm for finding shortest routes from all source nodes to a given destination in general networks*, in Quarterly of Applied Mathematics 27, 1970, p. 526530.

Edited by: Viorel Negru and Daniela Zaharie
Received: June 1, 2013
Accepted: Jul 15, 2013