Fundamenta Informaticae XXI (2001) 1001–1020
IOS Press

A Logic-Based System for e-Tourism

Francesco Ricca [C]
Dipartimento di Matematica
Università della Calabria, via P. Bucci Cubo 30b, 87030 Rende, Italy
ricca@mat.unical.it

Antonella Dimasi
Exeura Srl
Via Pedro Alvares Cabrai - C.da Lecco 87036 Rende (CS), Italy
antonella.dimasi@exeura.com

Giovanni Grasso
Dipartimento di Matematica
Università della Calabria, via P. Bucci Cubo 30b, 87030 Rende, Italy
grasso@mat.unical.it

Salvatore Maria Ielpa
Dipartimento di Matematica
Università della Calabria, via P. Bucci Cubo 30b, 87030 Rende, Italy
s.ielpa@mat.unical.it

Salvatore Iiritano
Exeura Srl
Via Pedro Alvares Cabrai - C.da Lecco 87036 Rende (CS), Italy
salvatore.iiritano@exeura.com

Nicola Leone
Dipartimento di Matematica
Università della Calabria, via P. Bucci Cubo 30b, 87030 Rende, Italy
leone@mat.unical.it

Abstract. In this paper we present a successful application of logic programming for e-tourism: the IDUM system. The system exploits two technologies that are based on the state-of-the-art computational logic system DLV: (i) a system for ontology representation and reasoning, called OntoDLV; and, (ii) HıLεX a semantic information-extraction tool. The core of IDUM is an ontology which models the domain of touristic offers. The ontology is automatically populated by extracting the information contained in the touristic leaflets produced by tour operators. A set of specifically devised logic programs is used to reason on the information contained in the ontology for selecting the holiday packages that best fit the customer needs. An intuitive web-based user interface eases the task of interacting with the system for both the customers and the operators of a travel agency.

Keywords: Answer Set Programming, ASP, E-Tourism, Knowledge Representation and Reasoning, Information Extraction

Address for correspondence: Dipartimento di Matematica, Università della Calabria, via P. Bucci Cubo 30b, 87030 Rende, Italy

[C]Corresponding author

1. Introduction

In the last few years, the tourism industry has strongly modified marketing strategies with the diffusion of e-tourism portals in the Internet. Large tour operators are exploiting new technologies, such as web portals and e-mails, in order both to simplify their advertising strategies and reduce the selling costs. The efficacy of e-tourism solutions is also witnessed by the continuously growing community of e-buyers who prefer to surf the Internet for buying holiday packages. On the other hand, traditional travel agencies are undergoing a progressive loss of marketing competitiveness. This is partially due to the presence of web portals, which basically exploit a new market. Indeed, Internet surfers often like to be engaged in self-constructing their holiday by manually searching for flights, accommodation etc. Instead, the traditional selling process, whose strength lies in both direct contact with customer and knowledge about customer habits, is experiencing a reduced efficiency. This can be explained by the increased complexity of matching demand and offer. Indeed, travel agencies receive thousand of e-mails per day from tour operators containing new pre-packaged offers. Consequently, the employees of the agency cannot know all the available holiday packages (since they cannot analyze all of them). Moreover, customers are more demanding than in the past (e.g. the classic statement "I like the sea" might be enriched by "I like snorkeling", or "please find an hotel in Cortina" might be followed by "with beauty and fitness center facilities") and they often do not declare immediately all their preferences and/or constraints (like, e.g., budget limits, preferred transportation mean or accommodation etc.). The knowledge of customers preferences plays a central role in the traditional selling process. However, the task of matching this information with the large and unstructured e-mail database is both difficult to carry out in a precise way and is time consuming. Consequently, the seller is often unable to find the best possible solution to the customer needs in a reasonable time.

The goal of the IDUM project is to devise a system that addresses the above-mentioned causes of inefficiency by offering:

(i) an automatic extraction and interpretation of the touristic offers contained in the touristic leaflets which are received by email by the travel agents,

(i) a classification of the incoming touristic offers in a rich and easy-to-browse ontology (so that they are immediately available for the seller), and

(ii) an "intelligent" search that combines knowledge about users preferences with geographical information, and matches user needs with the available offers, automatically taking into account also the knowledge of the travel agent on recommended/unrecommended places and dates, along with general information on meteorological and political dangers taken by the internet (e.g., from the web site of the Italian Ministry for Foreigner Affairs).

We could achieve the goal by exploiting some tools based on Computational Logics and, particularly, on Answer Set Programming (ASP) [10]. ASP is a powerful logic programming language, which is very expressive in a precise mathematical sense: in its general form, allowing for disjunction in rule heads and nonmonotonic negation in rule bodies, ASP can represent *every* problem in the complexity class Σ_2^P and Π_2^P (under brave and cautious reasoning, respectively) [7] in a fully declarative way. The core functionalities of IDUM were based on two technologies[1] relying on the state-of-the-art ASP system

[1]Both systems are developed by Exeura srl, a technology company working on analytics, data mining, and knowledge management, that is investing on their industrialization finalized to commercial distribution.

DLV [13]:

- OntoDLV [21, 22] a powerful ASP-based system for ontology representation and reasoning; and,
- HιLεX [16, 24, 23], an advanced tool for semantic information-extraction from unstructured or semi-structured documents.

More in detail, in the IDUM system, behind the web-based user interface (which can be used by both employees of the agency and customers), there is an "intelligent" core that exploits a rich OntoDLV ontology that models the domain of discourse and the user preferences, and stores all the available data. Geographic information was obtained by including *GeoNames* [20], one of the largest publicly-available geographical databases, and enriched by modeling the knowledge of the travel agent regarding places and offered holidays. The ontology is automatically populated with touristic offers by extracting the information contained in the touristic leaflets produced by tour operators. It is worth noting that, offers are mostly received by the travel agency in a dedicated e-mail account. Moreover, the received e-mails are human-readable, and the details are often contained in email-attachments of different format (plain text, pdf, gif, or jpeg files) and structure that might contain a mix of text and images. The HιLεX system allows for automatically processing the received contents, and for populating the ontology with the data extracted from touristic leaflets. Once the information is loaded onto the ontology, the user can perform an "intelligent" search for selecting the holiday packages that best fit the customer needs. IDUM tries to mimic the behavior of the typical employee of a travel agency by exploiting a set of specifically devised logic programs that "reason" on the information contained in the ontology.

In the remainder of the paper, we first introduce the employed ASP-based technologies; then, in Section 3, we describe how the crucial tasks have been implemented; we show the architecture of the IDUM system in Section 4; in Section 5 we discuss related work; finally, we draw the conclusion in Section 6.

2. Underlying ASP-based technologies

The core functionalities of the e-tourism systems IDUM were based on two technologies relying on the DLV system [13]: OntoDLV [21, 22] a powerful ASP-based ontology representation and reasoning system; and, HιLεX [16, 24, 23], an advanced tool for semantic information-extraction from unstructured or semi-structured documents.

In the following we briefly describe both OntoDLV and HιLεX, the reader interested in a more detailed description is referred to [21, 22] and [16, 24, 23], respectively.

2.1. The OntoDLV System

Traditional ASP in not well-suited to ontology specifications, since it does not directly support features like classes, taxonomies, individuals, etc. Moreover, ASP systems are a long way from comfortably enabling the development of industry-level applications, mainly because they lack important tools for supporting programmers. All the above-mentioned issues were addressed in OntoDLV [21, 22] a system for ontologies specification and reasoning. Indeed, by using OntoDLV, domain experts can create, modify, store, navigate, and query ontologies; and, at the same time, application developers can easily develop their own knowledge-based applications on top of OntoDLV, by exploiting a complete Application Programming Interface [8]. OntoDLV implements a powerful logic-based ontology representation language, called OntoDLP, which is an extension of (disjunctive) ASP with all the main ontology constructs including classes, inheritance, relations, and axioms. In OntoDLP, a *class* can be thought of as a collection of individuals who belong together because they share some features. An individual, or *object*, is any identifiable entity in the universe of discourse. Objects, also called class instances, are unambiguously identified by their object-identifier (oid) and belong to a class. A class is defined by a name (which is unique) and an ordered list of attributes, identifying the properties of its instances. Each attribute has a name and a type, which is, in truth, a class. This allows for the specification of *complex objects* (objects made of other objects). Classes can be organized in a specialization hierarchy (or data-type taxonomy) using the built-in *is-a* relation (*multiple inheritance*). Relationships among objects are represented by means of *relations*, which, like classes, are defined by a (unique) name and an ordered list of attributes (with name and type). OntoDLP relations are strongly typed while in ASP relations are represented by means of simple flat predicates. Importantly, OntoDLP supports two kind of classes and relations: (base) classes and (base) relations, which correspond to basic facts (that can be stored in a database); and *collection* classes and *intensional* relations, which correspond to facts that can be inferred by logic programs; in particular, *collection classes* are mainly intended for object reclassification (i.e., for repeatedly classifying individuals of an ontology). For instance, the following statement declares a class modeling customers, which has six attributes, namely: *firstName*, *lastName*, and *status* of type string; *birthDate* of type Date; a positive integer *childNumber*, and *job* which contains an instance of another class called Job.

```
class Customer (firstName: string, lastName: string,  birthDate: Date,
                status: string,childNumber: positive integer, job: Job).
```

As in ASP, logic programs are sets of logic rules and constraints. However, OntoDLP extends the definition of logic atom by introducing class and relation predicates, and complex terms (allowing for a direct access to object properties). This way, the OntoDLP rules merge, in a simple and natural way, the declarative style of logic programming with the navigational style of the object-oriented systems. In addition, logic programs are organized in *reasoning modules*, to take advantage of the benefits of modular programming. For example, with the following program we single out the pairs of customers having the same birthdate:

```
module (CustomersWithTheSameBirthDate) {
  sameBirthDate(C1,C2,D) :- C1: Customer(birthDate:D),  C2: Customer(birthDate:D).
}
```

Classes and relations can be populated by either asserting logic facts or by exploiting the information contained in existing relational databases [2]. In the latter case, the instances are defined by means of mapping rules that "virtually import" the data from a given database. For instance the following specification populates *Continent* class by taking the information from table *continent_codes* in the database identified by *geo_db*:

```
virtual class Continent ( name:string )
{
  f(ID):Continent( name:Name ) :- continent_codes@geo_db( geoname_id:ID, name:Name ).
}
```

Here, the rule acts as mapping between the data contained in table *continent_codes* and the instances of class *continent*; whereas functional object identifiers are suitably built from database values for identifying ontology instances. This kind of classes and relations are called *virtual* because their instances come from (and might permanently reside in) an external source; but, as far as reasoning and querying are concerned they are like any other class directly specified in OntoDLP.

The core of OntoDLV is a rewriting procedure [22] that translates ontologies, and reasoning modules to an equivalent standard ASP program which, in the general case, runs on the state-of-the art ASP system DLV [13]. OntoDLV features an advanced persistency manager that allows one to store ontologies transparently both in text files and internal relational databases; while powerful type-checking routines are able to analyze ontology specifications and single out consistency problems. Importantly, if the rewritten program is stratified and non-disjunctive [10, 9, 19] (and the input ontology resides in relational databases) then the evaluation is carried out directly in mass memory by exploiting a specialized version of the same system, called DLV ^{DB} [27]. Note that, since class and relation specifications are rewritten into stratified and non-disjunctive programs, queries on ontologies can always be evaluated by exploiting a DBMS. This makes the evaluation process very efficient, and allows the knowledge engineer to formulate queries in a language more expressive than SQL.

2.2. The HιLεX System

HιLεX [16, 24, 23] is an advanced system for ontology-based information extraction from semi-structured and unstructured documents, which is already exploited in many relevant real-world applications. The HιLεX system implements a semantic approach to the information extraction problem based on a new-generation semantic conceptual model by exploiting: (i) ontologies as knowledge representation formalism; (ii) a general document representation model for unifying different input formats (html, pdf, doc, ...); and, (iii) the definition of a formal attribute grammar able to describe, by means of declarative rules, objects/classes w.r.t. a given ontology.

Most of the existing information extraction approaches do not work in a semantical way and they are not independent of the specific type of document they process. Contrariwise, the approach implemented in HιLεX confirms that it is possible recognize, extract and structure relevant information from heterogeneous sources. HιLεX is based on OntoDLP for describing ontologies, since this language perfectly fits the definition of semantic extraction rules. Regarding the unified document representation, the idea is that a document (unstructured or semi-structured) can be seen as a suitable arrangement of objects in a two-dimensional space. Each object has its own semantics, is characterized by some attributes and is located in a two-dimensional area of the document called *portion*. A portion is defined as a rectangular area univocally identified by four cartesian coordinates of two opposite vertices. Each portion "contains" one or more objects and an object can be recognized in different portions.

The language of HιLεX is founded on the concept of *ontology descriptor*. A "descriptor" looks like a production rule in a formal attribute grammar, where syntactic items are replaced by ontology elements, and where extensions for managing two-dimensional objects are added. Each descriptor allows us to describe: (i) an ontology object in order to recognize it in a document; or (ii) how to "generate" a new object that, in turn, may be added in the original ontology.

Note that an object may also have more than one descriptor, thus allowing one to recognize the same kind of information when it is presented in different ways.

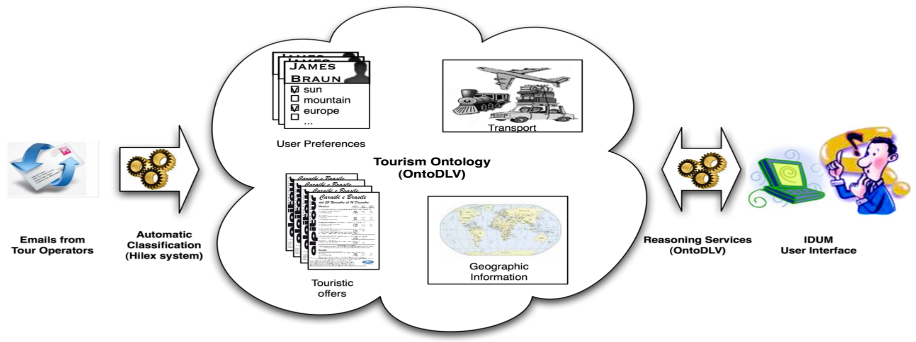

Emails from Tour Operators
Automatic Classification (Hilex system)
JAMES BRAUN
sun
mountain
europe
...
User Preferences
Transport
Tourism Ontology (OntoDLV)
Touristic offers
Geographic Information
Reasoning Services (OntoDLV)
IDUM User Interface

Figure 1. The IDUM System.

3. The IDUM System

In this section we describe the core of the IDUM system: its innovative features based on ASP. IDUM is an e-tourism system conceived for classifying and driving the search of pre-packaged touristic offers for both travel agencies operators and their customers. The system, like other existing portals, is equipped with a proper (web-based) user interface; but, behind the user interface there is an "intelligent" core that exploits knowledge representation and reasoning technologies based on ASP. In IDUM (see Figure 1), the information regarding the touristic offers provided by tour operators is mainly received by the system as a set of e-mails. Each e-mail might contain plain text and/or a set of leaflets, usually distributed as pdf or image files which store the details of the offer (e.g., place, accommodation, price etc.). Leaflets are devised to be human-readable, might mix text and images, and usually do not have the same layout. E-mails (and their content) are automatically processed by using the HiLεX system (i.e., e-mails are "unwrapped": attachments are separately processed, enclosed external links are followed and corresponding web pages analyzed); and, the extracted data about touristic offers is used to populate an OntoDLP ontology that models the domain of discourse: the "*tourism ontology*".

Note that, the automatic population of the ontology is one of the innovative features of the system; but IDUM can deal also with sources different from e-mails. Indeed, the internal ontology can be also manually-populated by travel agents by exploiting a proper form, and additional leaflets can be given in input to the extraction system.

The resulting ontology is, then, analyzed by exploiting a set of reasoning modules (ASP programs) combining the extracted data with knowledge about places (geographical information) and user preferences specified in the tourism ontology. The system mimics the typical deductions made by a travel agency employee for selecting the most appropriate answers to the user needs.

It is worth pointing out that the final goal of the system is to provide an effective interface for the employees of a travel agency, and, thus, IDUM is not equipped with a reservation system (travel agencies already own conventional systems for managing reservations). The same holds for the customer interface, which only allows for browsing the holiday packages. Indeed, the customer interface has been devised with the intent of welcoming customers and offering a comfortable access to the travel agency. The goal is also to reduce the "visiting-only" phenomenon, which consists in the repeated visit of customers that not necessarily end in a purchase.

In the following subsections, we describe the main components of the tourism ontology and the implementation of the above-mentioned ASP-based features. It is worthwhile noting that, for clarity of presentation, we show a simplified version of both ontology and reasoning modules (logic programs).

3.1. The Tourism Ontology

The "*tourism ontology*" was developed with the cooperation of the staff of a real touristic agency. In this way, we could model the key entities that describe the process of organizing and selling a holiday package. In particular, the tourism ontology models: geographic information, travel agent knowledge, user preferences, and touristic offer information.

Geographical Information. The most relevant ontology entities that model geographical information are reported in Figure 2. World locations are modeled by class *Place*, and partitioned into six (sub)classes, one for each touristically-relevant kind of place: *Continent*, *Nation*, *AdminSubArea*, *PopulatedPlace*, *Mountain*, and *Island*. In practice, places are hierarchically organized into four levels according to their natural containment relationship. In particular, the basic place-containment is modeled in the ontology by means of five relations, namely: *ContinentContainsNation*, *NationContainsAdminSubArea*, *AdminSubAreaContainsPopulatedPlace*, *AdminSubAreaContainsMountain*, and *AdminSubAreaContainsIsland*. The first one models the containment of nations in continents; whereas we considered only one level of subdivision of nations, which was determined in accordance with their administrative organization (e.g., Italy is divided into "regioni", England is divided into "countries", etc.); on the last level, there are "populated places" (i.e., cities and villages), mountains and islands, which, in turn, are contained in administrative areas. The complete *part-of* hierarchy of places, containing all direct and indirect inclusions, is obtained with the intensional relation *Contains*. Indeed, *Contains* is defined by means of a logic program made up of six logic rules (see Figure 2): one for each virtual relation modeling basic place-containment, and an additional one that basically encodes a transitive closure. Note that, ASP rules allowed us to define (and compute) the full place-inclusion hierarchy in a simple yet effective way. In the following it will be clear that the information modeled in the *Contains* relation is crucial for the effectiveness of both extraction process and package search.

The relation *AlternateNames* associates all the known toponyms with corresponding places (e.g. a different one for each language); this information is exploited during both the extraction process and pre-processing of user queries steps for correctly disambiguating the input.

The above-described ontology entities have been populated by exploiting one of the largest publicly-available geographical databases: *GeoNames* [20]. *GeoNames* contains thousand of places, alternative toponyms and related geographic information. In order to map the *GeoNames* database to our ontology we exploited both virtual classes and virtual relations [2].[2] In particular, some of the mappings between *tourism ontology* classes and *GeoNames* tables are reported in Figure 3. The first statement of Figure 3 links the *GeoNames* database to the ontology by defining the database identifier "geo_db"; whereas, the instances of class *Continent* (resp. *Nation*) are obtained from table *continent_codes* (resp. *country_info*) by means of a single mapping rule where database tables are referred by exploiting *sourced atoms*. Note that, object identifiers are univocally built by exploiting the database code *geoname_id*. Finally, *ContinentContainsNation* is given by the join of tables *continent_codes* and *country_info*. The remaining virtual classes and relations were defined in a similar way.[3]

[2]Note that, apart from class *Place* and relation *Contains*, all the remaining (basic) ontology entities in Figure 2 are *virtual*.
[3]A description of *GeoNames* is not reported here for space reasons. The interested reader can find it on the Web at [20].

```
% places and related information
class Place ( name:string, kind:PlaceKind ).

class PlaceKind ().
continent:PlaceKind().  nation:PlaceKind(). adminSubArea:PlaceKind().
populatedPlace:PlaceKind(). mountain:PlaceKind().  island:PlaceKind().

% virtual classes taking places from GeoNames
virtual class Continent( name:string, kind:PlaceKind ) isa Place.
virtual class Nation( name:string, kind:PlaceKind ) isa Place.
virtual class AdminSubArea( name:string, kind:PlaceKind ) isa Place.
virtual class PopulatedPlace( name:string, kind:PlaceKind ) isa Place.
virtual class Mountain( name:string, kind:PlaceKind ) isa Place.
virtual class Island( name:string, kind:PlaceKind ) isa Place.

% alternative place-names from GeoNames
virtual relation AlternativeName ( place:Place, alternateName:string ).

% place containment
intentional relation Contains ( ctr:Place, ctd:Place ).
{
 Contains( ctr:P1, ctd:P2 ) :- ContinentContainsNation( ctr:P1, ctd:P2 ).
 Contains( ctr:P1, ctd:P2 ) :- NationContainsAdminSubArea( ctr:P1, ctd:P2 ).
 Contains( ctr:P1, ctd:P2 ) :- AdminSubAreaContainsPopulatedPlace( ctr:P1, ctd:P2 ).
 Contains( ctr:P1, ctd:P2 ) :- AdminSubAreaContainsMountain( ctr:P1, ctd:P2 ).
 Contains( ctr:P1, ctd:P2 ) :- AdminSubAreaContainsIsland( ctr:P1, ctd:P2 ).

 Contains( ctr:P1, ctd:P2 ) :- Contains( ctr:P1, ctd:P3 ), Contains( ctr:P3, ctd:P2 ).
}

% virtual relations reconstructing basic place containment from GeoNames
virtual relation ContinentContainsNation ( ctr:Continent, contained:Nation ).
virtual relation NationContainsAdminSubArea ( ctr:Nation, contained:AdminSubArea ).
virtual relation AdminSubAreaContainsPopulatedPlace ( ctr:AdminSubArea,
                                                       ctd:PopulatedPlace ).
virtual relation AdminSubAreaContainsMountain ( ctr:AdminSubArea,
                                                       ctd:Mountain ).
virtual relation AdminSubAreaContainsIsland ( ctr:AdminSubArea,
                                                       ctd:Island ).
```

Figure 2. Tourism Ontology: Geographic Information.

Travel Agent Knowledge. In the ontology, the mere geographic information is enriched by the knowledge that is usually exploited by travel agency employees for selecting a travel destination (see Figure 4). For example, travel agents know which place offers a specific kind of trip (e.g. Kenya offers safari, Sicily offers both sea and sightseeing). To this end, the *tourism ontology* contains the class *TripKind* (e.g. sa-

```
% specify source database
geo_db:dbsource(jdbc:"jdbc:mysql://localhost/geonames",odbc:"gn",user:"geo",psw:"geo").

% Continent class specification
virtual class Continent ( name:string, kind:PlaceKind ) {
  f(ID):Continent( name:Name, kind:continent ) :-
                                continent_codes@geo_db( geoname_id:ID, name:Name ).}

% Nation class specification
virtual class Nation ( name:string, kind:PlaceKind ) {
  f(ID):nation( name:Name, kind:nation ) :-
                                country_info@geo_db( geoname_id:ID, country:Name ).}

% ContinentContainsNation relation specification
virtual relation ContinentContainsNation( ctr:Continent, ctd:Nation ) {
  ContinentContainsNation( ctr:f(ContId), ctd:f(NatId) ) :-
                             continent_codes@geo_db( geoname_id:ContId,code:Code ),
                             country_info@geo_db( geoname_id:NatId,continent:Code ).}
...
```

Figure 3. Mapping the Geonames database.

fari, cruise, etc. are instances of the *TripKind* class) and relation *PlaceOffer* that associates a kind of trip with places. Nevertheless, a place might offer some specific facilities (e.g. in Marsa Alam Egypt it is possible to go snorkeling). Thus, the ontology contains a class called *PlaceFacility* and a binary relation *PlaceOffersFacility* associating facilities with places. Moreover, travel agents might suggest avoiding sea holidays in winter, or going to India during the wet monsoon period; whereas, a visit to Sicily might be recommended during summer. This information is encoded in the ontology by means of the two relations: *RecommendedPeriod*, *BadPeriod*, respectively associating places with periods in which they should be visited or avoided. It is also the case that places engaged in a war or subjected to terrorism (like, eg. Afghanistan today) should not be recommended for a holiday; thus, relation *DangerousPlace* lists places that are currently considered risky.

It is worth noting that the instances of classes and relations modeling the knowledge of travel agents were manually-inserted (in accordance with the information provided by a real travel agent and/or publicly available on the Internet) for a large number of places, and can be updated in the system through the travel agent interface.

Touristic Offer Information. Holiday packages are modeled by the *TouristicOffer* class (see Figure 4). In detail, for each holiday package we store: both the place of departure and the final destination; the kind of trip; cost and duration (in days); the tour operator selling the package (which is an instance of the *TourOperator* class); an url pointing to the original leaflet, and the expiration date (which is exploited by the system for periodically removing expired offers). The instances of *TouristicOffer* are added either automatically, by exploiting the HιLεX system (see next Section), or manually (via the user interface) by the personnel of the agency. Optionally, touristic offers might have associated some transportation

```
% travel agent knowledge
relation PlaceOffer ( place: Place, kind: TripKind ).
relation SuggestedPeriod ( place:Place, period: HolidayPeriod ).
relation BadPeriod ( place:Place, period: HolidayPeriod ).
relation DangerousPlace ( place:Place ).
class PlaceFacility ().
relation PlaceOffersFacility( place:Place, fac:PlaceFacility ).

% touristic offer information
class TouristicOffer ( start: Place, destiNation: Place,
    kind: TripKind, cost: positive integer, fromDay: Date, toDay: Date,
    period: HolidayPeriod, maxDuration: positive integer,
    deadline: Date, uri: string, tourOperator: TourOperator ).

class TripKind ().
class HolidayPeriod ().
class TourOperator ( name:string, company:string, phone:string, fax:string,
                     email:string, contact:string ).
class TransportationMean ().
class AccommodationKind ().
class AccommodationFacility ().

relation OfferMean ( offer:TouristicOffer, means:TransportationMean ).
relation OfferAccommodation ( offer:TouristicOffer, acc:AccommodationKind,
                             descr:string ).
relation OfferAccommodationFacility( offer:TouristicOffer, fac:AccommodationFacility ).

% user profile
class Customer (firstName: string, lastName: string, birthDate: Date, status: string,
                childNumber: positive integer, job: Job ).

relation CustomerPreferredTrip( cust:Customer, kind:TripKind ).

relation CustomerPreferredMean( cust:Customer, mean: TransportationMean,
                              relevance:positive integer ).
relation CustomerPreferredAccommodation( cust:Customer, acc:Accommodation,
                              relevance:positive integer ).
relation CustomerPreferredAccommodationFacility( cust:Customer, fac:AccommodationFacility,
                              relevance:positive integer ).
relation CustomerPreferredPlaceFacility( cust:Customer, fac:PlaceFacility,
                              relevance:positive integer ).
```

Figure 4. Tourism Ontology: Travel Agent Knowledge, Offers Information and User Profile.

means,[4] an accommodation kind (e.g. hotel, apartment, etc.) and some accommodation facilities (e.g. swimming pool, spa service, etc.). This additional information is modeled in the ontology by means

[4]In case there is more than one alternative transportation mean, the ontology is intended to be filled with the main one.

of the following classes: *TransportationMean*, *AccommodationKind* and *AccommodationFacility*. The association of this information to touristic offers is modeled by the following relations: *OfferMean*, *OfferAccommodation* and *OfferAccommodationFacility*.

User Profile. In order to personalize the trip-search, user profiles are also modeled in the *tourism ontology*. The class *Customer* models the personal information of each customer. User preferences are modeled by exploiting a number of relations, namely: *CustomerPreferredTrip, CustomerPreferredMean*, *CustomerPreferredAccommodation*, *CustomerPreferredAccommodationFacility*, and *CustomerPreferredPlaceFacility*. The first relation associates a preferred trip kind to each customer; whereas, the remaining ones associate, for each customer, a preference score with each specific facility that might be associated to an offer, namely: transportation means, accommodation kind, accommodation facilities and place facilities.

It is worth pointing out that, at the moment, the user profile data is manually-inserted in the ontology and can be updated by either the travel agent or the customer himself by filling-in an appropriate form.

3.2. Automatic Extraction of Touristic Offers

Touristic offers are mainly available in digital format and they are received via e-mail. It is usual that more than a hundred emails per day crowd the mail folder of a travel agency, and often the personnel cannot even analyze the entire in-box. This causes a loss of efficiency in the selling process, because some interesting offers will be ignored. Note that, most of the information is contained in pdf, gif or jpeg files attached to the e-mail messages, and this strongly limits the efficacy of standard search tools like, e.g., the ones provided by e-mail clients.

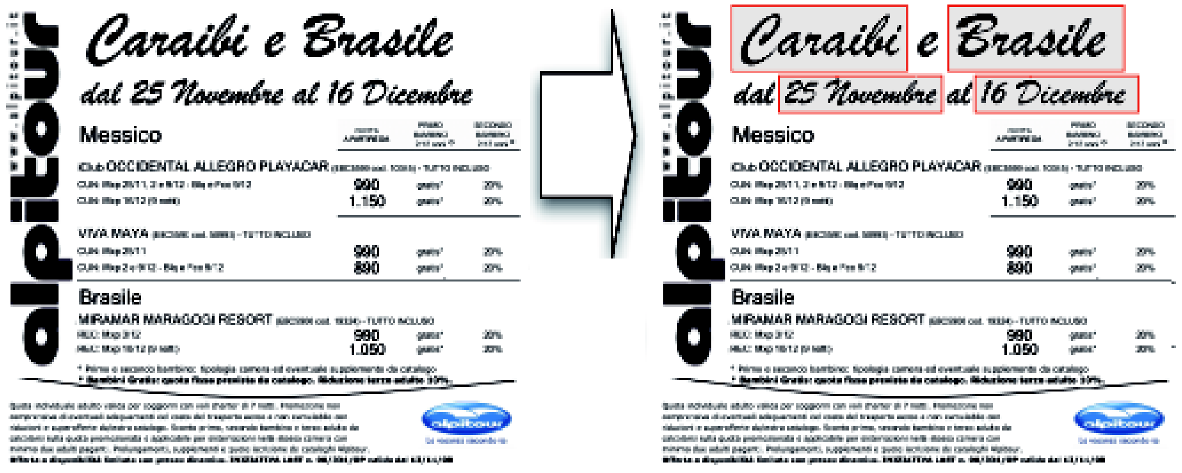
To deal with this problem, the IDUM system is equipped with an automatic classification system based on HιLεX. Basically, after some pre-processing steps, in which e-mails are automatically read from the inbox, and their attachments are properly handled (e.g. image files are analyzed by using OCR software), the input is sent to HιLεX. In turn, HιLεX is able to both extract the information contained in the e-mails and populate the *TouristicOffer* class. This was obtained by encoding several ontology descriptors (actually, we provided several descriptors for each kind of file received by the agency). For instance, the following descriptor:

```
<TouristicOffer (Destination, Period)> -> <X:place(XX)>{Destination:=X;}
    <X:date(XX)>{Period:=X;} SEPBY <X:separator()>.
```

extracts from the leaflet in Figure 5(a) that the proposed holiday package regards trips to both the Caribbean islands and Brazil. Moreover, it also extracts the period in which this trip is offered. The extracted portions are outlined in Figure 5(b). The result of the application of this descriptor are two new instances of the TouristicOffer class.

3.3. Personalized Trip Search

The second crucial task carried out in the IDUM system is the personalized trip search. This feature was conceived to make simpler the task of selecting the holiday packages that best fit the customer needs. We tried to “simulate” the deductions made by an employee of the travel agency in the selling process by using a set of reasoning modules, i.e. a set of specifically devised logic programs.



alpitour

Caraibi e Brasile

dal 25 Novembre al 16 Dicembre

Messico

Club OCCIDENTAL ALLEGRO PLAYACAR [illegible]

[illegible]	990	[illegible]	[illegible]
[illegible]	1.150	[illegible]	[illegible]

VIVA MAYA [illegible]

[illegible]	990	[illegible]	[illegible]
[illegible]	890	[illegible]	[illegible]

Brasile

MIRAMAR MARAGOGI RESORT [illegible]

[illegible]	990	[illegible]	[illegible]
[illegible]	1.050	[illegible]	[illegible]

[illegible]

Figure 5. Extracting offer information.

In a typical scenario, when a customer enters the travel agency, an employee tries to understand his current desires and preferences at first; then, the seller has to match the obtained information with a number of pre-packaged offers. Actually, a number of candidate offers are proposed to the customer and, then, the employee has to understand his needs by interpreting the preferences of the customer. Customer preferences depend on his personal information but also on his holiday habits. Actually, this information has to be elicited by the seller by interviewing the customer, but most of it might be already known by the employee if he is serving a regular customer. In this process, what has to be clearly understood (for properly selecting a holiday package fitting the customer needs) is summarized in the following four words: *where*, *when*, *how*, and *budget*. Indeed, the seller has to understand *where* the customer desires to go; *when* he can/wishes to leave; how much time he will dedicate to his holiday; which is the preferred transportation means (*how*); and, finally, the available budget. However, the customer does not directly specify all this information, for example, he can ask for a sea holiday in January but he does not specify a precise place, or he can ask for a kind of trip that is unfeasible in a given period (e.g. winter holiday in Sicily in August). In this case the seller has to exploit his knowledge of the world for selecting the right destination and a good offer in a huge range of proposals. This is exactly what the IDUM system does while searching for a holiday package. Current needs are specified by filling an appropriate search form, where some of the key information has to be provided (i.e. where and/or when and/or available money and/or how). Note that, the tourism ontology models both the knowledge of the seller and the profile of customers; moreover, the extraction process continuously populates the ontology with new touristic offers and periodically removes the expired ones. Thus, the system, by running some specifically devised reasoning modules, combines the specified information with the one available in the ontology, and shows the holiday packages that best fit the customer needs.

Some of the reasoning modules exploited by the IDUM system are reported (in a simplified form) in Figure 6. The first module, named *match profile*, allows for ranking the available offers w.r.t. the user preferences. In particular, the first four rules single out the scores obtained by offers w.r.t. the profile of customers. In detail, the first rule can be read as follows: “for each customer C associate to each offer O the score R if the transportation mean included in O has relevance R according to the preferences of C”.

In a similar way, the next three rules associate, for each customer, a specific score with available offers depending on: accommodation kind, accommodation facilities and destination facilities, respectively. Finally, the auxiliary predicate *preferenceSatisfaction* computes the overall satisfaction ranking obtained by available offers. Given a customer C, the overall satisfaction ranking Sat for an offer O is obtained by summing up all the scores obtained by the features included in O (in accordance with the preferences of C). The ranking defined by *preferenceSatisfaction* is, then, exploited by the system either for (a) proposing some possibly interesting offer to the user when he/she logs-in (see Figure 9); or (b) for ranking the results obtained during the search of touristic offers. Task (a) is implemented by evaluating module *welcome recommendations*; whereas, task (b), is encoded in the module called *personalized search* (see Figure 6). Basically, module *welcome recommendations* mimics the usual behavior of a travel agent that proposes to an incoming old customer some "tempting" offer. More in detail, predicate *loginOffers* is defined that lists the touristic offers that propose the trip kind preferred by C and maximize the satisfaction of user preferences. However, the customer might have in mind some specific holiday, and in this case, the system has to search a holiday package that matches the current customer requests. Suppose that the customer specifies destination and period, then the *personalized search* module creates a suitable selection of holiday packages as follows. The first two rules select *requested places* by exploiting *Contains* relation (i.e. we consider the specified destination together with all the places it contains). Note that, this additional information plays a fundamental role for improving the search; indeed, it is common that the user specifies a nation (e.g., "Italy") and the system has to consider also packages having a specific place (e.g., "Rome") as destination. Then, rule (iii) singles out each offer O having a requested place as destination. However, it might be the case that a matching offer is not recommendable (and we call it "*bad offer*") either because the chosen period is not the right one for the specified destination (rule iv) or because a trip to the specified destination is currently considered risky (rule v). If some "bad offer" matches the customer query, then the system prompts an alert message next to the search results (see Figure 10); moreover, the system might recommend some alternative for the specified period which is not bad (rule vi). Note that, predicates collecting both matching and alternative holiday packages associate to found offers a preference satisfaction score. In this way, results can be ordered by the user interface according to user preferences and offers that best fit the customer needs are showed first (see Figure 10).

Several other search criteria can be specified, including trip kind, available budget, and so on (see Figure 8). Actually, depending on the input parameters, the system selects a specific reasoning module obtained by suitably modifying the one described above. For instance, when the user specifies both trip kind and holiday period rule (i) is replaced by:

```
requestedPlace( P ) :- query( tripKind:K ), PlaceOffer( place:P, kind:K ).
```

where requested places are the ones offering the specified trip kind. Otherwise, when the budget is also specified both rule (iii) and rule (iv) are replaced by:

```
matchingOffer( O ) :- O:TouristicOffer( destination:P, cost:C ),
                           requestedPlace( P ), query( budget:B ), C <=B.
recommendedAlternative( Off, Sat ) :- query( customer:Cust, period:Per, budget:B ),
                           SuggestedPeriod( place:Dest, period:Per ),
                           not DangerousPlace( place:Dest ),
                           Off:TouristicOffer( destination:Dest, cost:C ),
                           preferenceSatisfaction( Cust, Off, Sat ), C <=B.
```

```
% ----------------------- Offer ranking based on customer preferences -----------------------
module(match profile) {
 satisfiesCustomerPreference( O, C, R ) :- O:TouristicOffer(), OfferMean( offer:O, means:M ).
                      CustomerPreferredMean( cust:C, mean:M, relevance:R ),
 satisfiesCustomerPreference( O, C, R ) :- O:TouristicOffer(),
                      CustomerPreferredAccomodation( cust:C, acc:A, relevance:R ),
                      OfferAccomodation( offer:O, acc:A ).
 satisfiesCustomerPreference( O, C, R ) :- O:TouristicOffer(),
                      CustomerPreferredAccomodationFacility( cust:C, fac:AF, relevance:R ),
                      OfferAccomodationFacility( offer:O, fac:AF ).
 satisfiesCustomerPreference( O, C, R ) :- O:TouristicOffer( destination:P ),
                      CustomerPreferredPlaceFacility( cust:C, fac:PF, relevance:R ),
                      PlaceOffersFacility( place:P, fac:PF ).

 preferenceSatisfaction( C, O, Sat ) :- C:Customer(), O:TouristicOffer(),
                      #sum{ R : satisfiesCustomerPreference( O, C, R )  } = Sat. }

% ---------------------- Recommend best offers when the user logs-in ----------------------
module(welcome recommendations) {
 loginOffer( C, O ) :- query( customer:C ), CustomerPreferredTrip( cust:C, kind:K ),
                      O:TouristicOffer( kind:K ), preferenceSatisfaction( C, O, MaxSat ),
                      MaxSat = #max{ Sat : preferenceSatisfaction( C, O, Sat )}. }

% ---------------------------- Search offers - basic method ----------------------------
module(personalized search) {
 % select available offers matching query
 (i)   requestedPlace( P )  :- query( place:P ).
 (ii)  requestedPlace( Pc ) :- requestedPlace( Pp ), Contains( cts:Pp, ctd:Pc ).
 (iii) matchingOffer( O, Sat ) :- O:TouristicOffer( destination:Dest, period:Per ),
                      requestedPlace( Dest ), query( customer:Cust, period:Per ),
                      preferenceSatisfaction( Cust, O, Sat ).

 % generate alert messages if matching offers are not recommendable
 (iv) badOffer( O ) :- matchingOffer( O ), O:TouristicOffer( period:P, destination:D ),
                      BadPeriod( place:D, period:P ).
 (v)  badOffer( O ) :- matchingOffer( O ), O:TouristicOffer( destination:P ),
                      DangerousPlace( place:P ).

 % select alternative/recommended offers
 (vi) recommendedAlternative( Off, Sat ) :- query( customer:Cust, period:Per ),
                      SuggestedPeriod( place:Dest, period:Per ),
                      not DangerousPlace( place:Dest ),
                      Off:TouristicOffer( destination:Dest, period:Per ),
                      preferenceSatisfaction( Cust, Off, Sat ). }
```

Figure 6. Personalized Trip Search: Welcome Suggestions and Touristic Offer Search.

where the additional condition is considered in the body and so on.

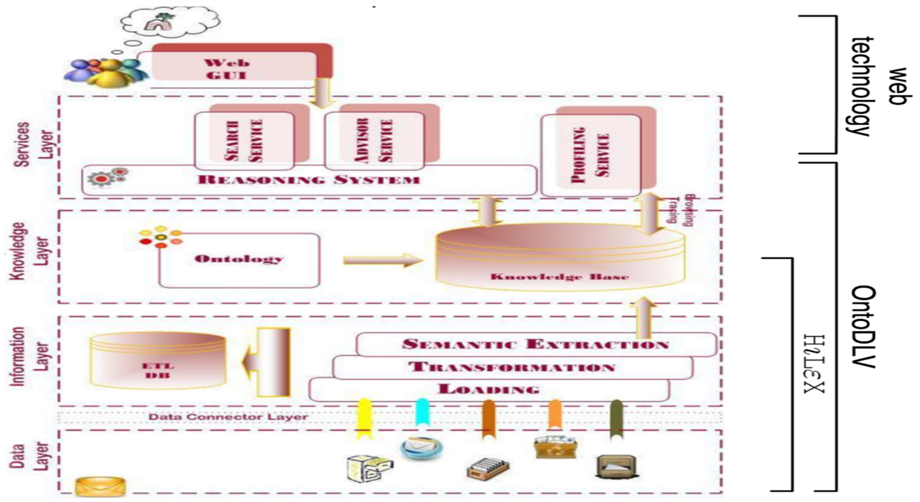


Figure 7. System Architecture.

4. System Architecture

The architecture of the IDUM system, which is depicted in Figure 7, is made up of four layers: *Data Layer, Information Layer, Knowledge Layer*, and *Service Layer*. In the Data Layer the input sources are dealt with. In particular, the system is able to store and handle the most common kind of sources: e-mails, plain text, pdf, gif, jpeg, and HTML files. The Information Layer provides ETL (Extraction, Transformation and Loading) functionalities, in particular: in the loading step the documents to be processed are stored in an auxiliary database (that also manages the information about the state of the extraction activities); whereas, in the Transformation step, semi-structured or non-structured documents are manipulated. First the document format is normalized; then, the "bi-dimensional logical representation" is generated (basically, the HīLεX portions are identified); finally, HīLεX descriptors are applied in the Semantic Extraction step and ontology instances are recognized within processed documents. The outcome of this process is a set of concept instances, which are recognized by matching semantic patterns, and stored in the core knowledge base of the system where the tourism ontology resides (Knowledge Layer). Domain ontology and extracted information are handled by exploiting the Persistency manager of the OntoDLV system (see Section 2.1). The Services Layer features the profiling service and the intelligent search (see Section 3.3) which implements the reasoning on the core ontology by evaluating in the OntoDLV system a set of logic programs. The Graphical User Interface (GUI) can access the system features by interacting with a set of web-services. The GUI offers both a dedicated access to travel agents (see Figure 8) and a user friendly environment for customers (see Figure 9-10).

5. Related Work

The usage of ontologies for developing e-tourism applications was already studied in the literature [15, 12, 3, 17, 18], and the potential of the application of semantic technology recognized [6, 11].

The architecture of an e-tourism system able to create a touristic package in a dynamic way is pre-

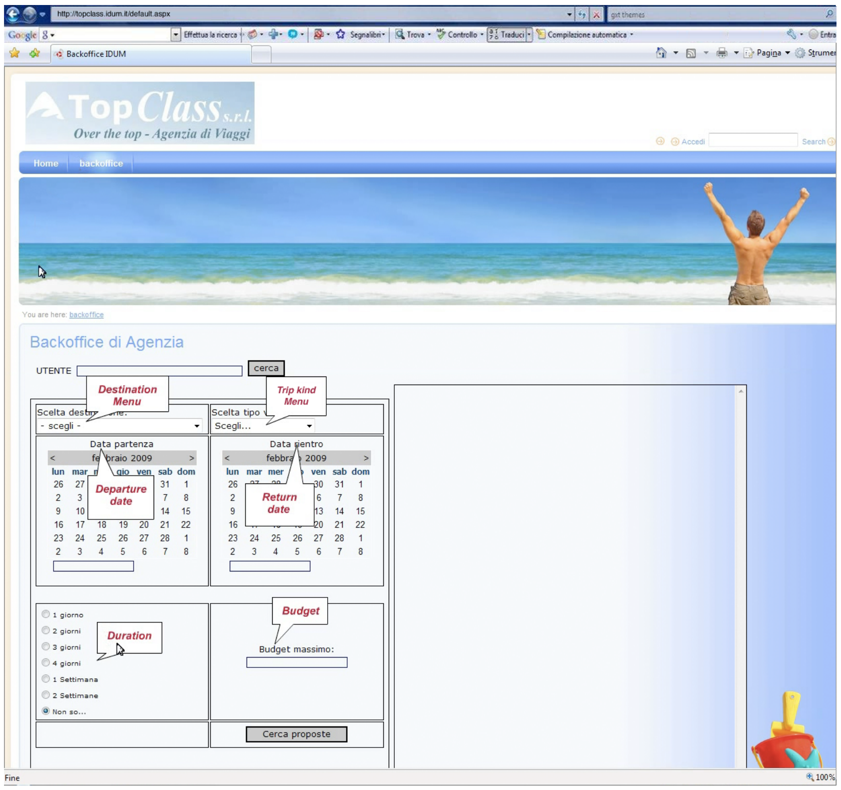


Figure 8. Agent Home Page

sented in [12]. This system permits the customer to specify a set of preferences for a vacation and dynamically access and query a set of information sources to find component such as accommodation, car rental, and leisure activities in real time. It is based on an ontology written in OWL-DL [26]. The ontology exploited in [3, 12] encodes the same key concepts as ours, but does not include information about user preferences. Another advantage of the our approach is the possibility of developing ASP programs that reason on the data contained in the ontology for developing complex searches, while there is no accepted solution for combining logic rules and OWL-DL ontologies.

The SPETA system [1], which is based on the ontology of [12], acts as an advisor for tourists. Fundamentally, SPETA follows people who need advising when visiting a new place, and who consequently do not know what is interesting to visit. Here the ontology is enriched with user profile information for determining the common characteristics of the previously visited places and the user behavior. In this way the system recommends attractions which are likely to fit the user expectations. It exploits GPS technology to know user position and it gets user data from previous users history and also from social networks. Both SPETA and IDUM exploit an ontology for building personalized solutions for the users, but the goal of SPETA is different from that of IDUM. Indeed, the former was conceived for offering assistance and information to the user when they already are on a place, while the goal of IDUM is to assist the users in the selection of a holiday.

The E-Tourism Working Group at DERI [4] is developing e-tourism solutions based on the Semantic

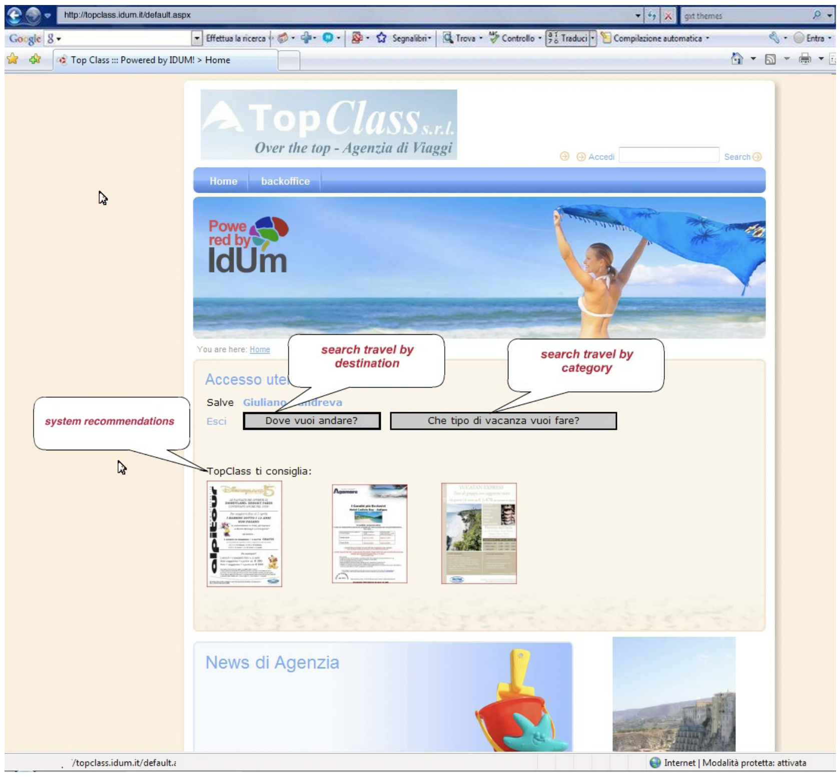

http://topclass.idum.it/default.aspx
Top Class ::: Powered by IDUM! > Home
TopClass s.r.l.
Over the top - Agenzia di Viaggi
Accedi
Search
Home
backoffice
Powered by IdUm
You are here: Home
search travel by destination
search travel by category
system recommendations
Accesso utente
Salve Giuliana Andreva
Esci
Dove vuoi andare?
Che tipo di vacanza vuoi fare?
TopClass ti consiglia:
News di Agenzia
Internet | Modalità protetta: attivata

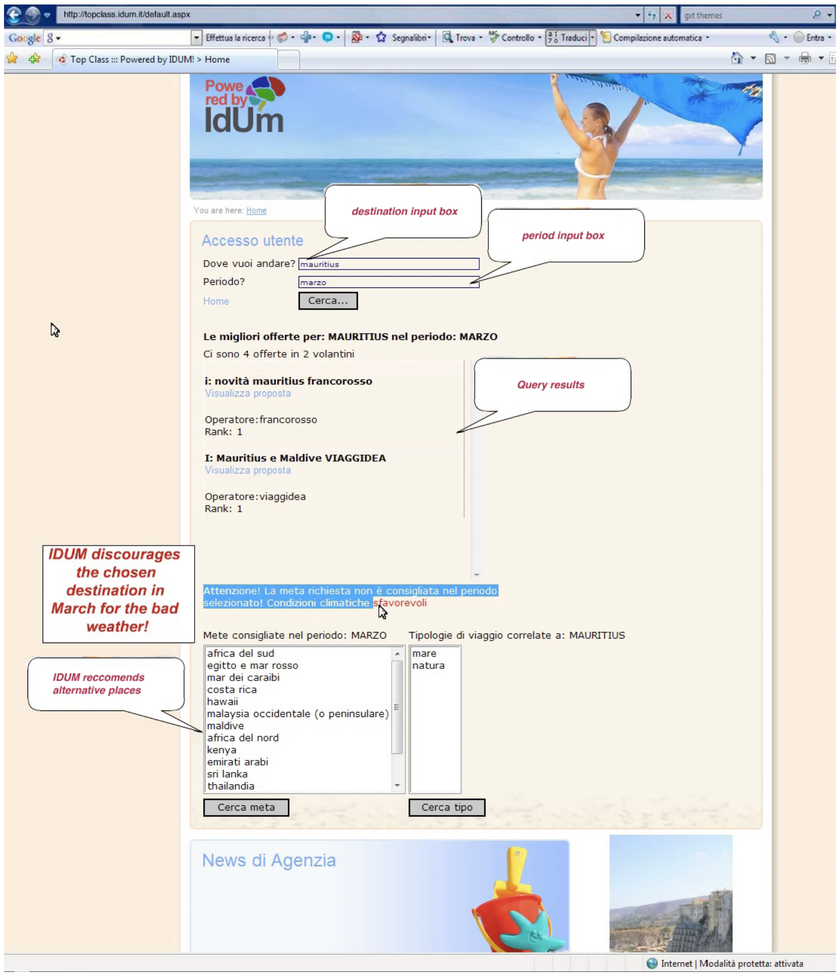
Figure 9. Customer Home Page

Web technology. Their goal is to develop an advanced eTourism Semantic Web portal which will connect the customers and virtual travel agents from anywhere at any time with any needs and requests. In [25], they present OnTour an information retrieval system that exploits an RDF engine for storing the data regarding accommodation facilities of different types. DERI also developed a content management system: OnTourism [5]. The solution is similar to IDUM, but it is based on the use of Lixto Software [14] which extracts information from web pages. Information about current events is crawled from several web sources and it is rendered in a machine-accessible semantic.

6. Conclusion and Market Perspective

In this paper we have described a successful example of commercial and practical use of logic programming: the e-tourism system IDUM.

The core of IDUM is an ontology modeling the domain of the touristic offers, which is automatically populated by extracting the information contained in the e-mails sent by tour operators; and an intelligent search tool based on answer set programming is able to search the holiday packages that best fits the customer needs. Actually, several logic programs have been devised for implementing the intelligent search. In the development process, we exploited many of the advanced features of the language like negation as failure and aggregates. The declarative nature of ASP allowed us to design effective solutions


destination input box
period input box
Query results
IDUM discourages the chosen destination in March for the bad weather!
IDUM reccomends alternative places
Powered by IdUm
News di Agenzia

You are here: Home

Accesso utente

Dove vuoi andare? mauritius
Periodo? marzo
Home Cerca...

Le migliori offerte per: MAURITIUS nel periodo: MARZO
Ci sono 4 offerte in 2 volantini

i: novità mauritius francorosso
Visualizza proposta

Operatore:francorosso
Rank: 1

I: Mauritius e Maldive VIAGGIDEA
Visualizza proposta

Operatore:viaggidea
Rank: 1

Attenzione! La meta richiesta non è consigliata nel periodo selezionato! Condizioni climatiche sfavorevoli

Mete consigliate nel periodo: MARZO
africa del sud
egitto e mar rosso
mar dei caraibi
costa rica
hawaii
malaysia occidentale (o peninsulare)
maldive
africa del nord
kenya
emirati arabi
sri lanka
thailandia
Cerca meta

Tipologie di viaggio correlate a: MAURITIUS
mare
natura
Cerca tipo

Figure 10. Query Result (search by place and period)

and to tune rapidly our modules by following the suggestions of the domain experts.

The system has been developed under project "IDUM: Internet Diventa Umana" (project n. 70 POR Calabria 2000/2006 Mis. 3.16 Azione D Ricerca e Sviluppo nella Imprese Regionali - Modulo B Voucher Tecnologici) funded by the Calabrian Region. The project team involved five organizations: the Department of Mathematics of the University of Calabria (which has ASP as one of the principal research area), the consortium Spin, Exeura srl (a company working on knowledge management), Top Class srl (a travel agency), and ASPIdea (a software farm specialized in the development of web applications). The members exploited their specific knowledge for developing the innovative features of the system. The strong synergy among partners made it possible to push the domain knowledge of the travel agency TopClass in both the ontology and in the reasoning modules. The result is a system that mimics the behavior of a seller of the agency and it is able to search in a huge database of automatically classified offers. IDUM combines the speed of computers with the knowledge of a travel agent for improving the efficiency of the selling process.

The IDUM system was initially conceived for solving the specific problems of a travel agency, and it is currently employed by one of the project partners: Top Class srl. We are working on an enterprise version of the system conceived for offering its advanced services to several travel agencies. The enhancements of IDUM will be developed under another technology-transfer PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project funded by the Calabrian region. The value of the system was confirmed by the good position obtained by the proposal in the project evaluation ranking (IDUM occupies the 2nd position ahead of more than 400 competing proposals). Moreover, we received very positive feedbacks from the market, indeed many travel agents are willing to use the system, and the potential of IDUM has been recognized also by the chair of the Italian touring club, which is the most important Italian association of tour operators.

As far as future work is concerned, we are investigating the application of data-mining techniques for automatically updating the user profile according to the customers' buying history. The actual system is targeted for finding closed packages (as they are presented in the leaflets), and it does not features an automatic holiday composition from more packages (clearly, this task can be carried out manually by both the travel agent and the final customer). An automatic package-composition feature is also the subject of future work.

Acknowlegments

The authors are thankful to the staff of Top Class srl and ASPIdea srl; a special thanks is addressed to Giuliano Candreva, Gianfranco De Franco, Dino De Santo and Carmine Donato.

References

[1] Angel, G.-C., Javier, C., Ismael, R., Myriam, M., Ricardo, C.-P., Miguel, G.-B. J.: SPETA: Social pervasive e-Tourism advisor, *Telemat. Inf.*, **26**(3), 2009, 306–315, ISSN 0736-5853.

[2] Bennardo, G., Grasso, G., Leone, N., Ricca, F.: Upgrading Databases to Ontologies, *Proceedings of the 3rd International Workshop on Applications of Logic Programming to the (Semantic) Web and Web Services (ALPSWS2008) 9-13 2008, Udine, Italy* (J. de Bruijn, S. Heymans, D. Pearce, A. Polleres, E. Ruckhaus, Eds.), 434, CEUR, 2008.

[3] Cardoso, J.: Developing An Owl Ontology For e-Tourism, in: *Semantic Web Services, Processes and Applications*, 2006, 247–282.

[4] DERI: Digital Enterprise Research Institute. Innsbruck, A.: `http://e-tourism.deri.at//`.

[5] Ding, Y., Prantner, K., Luger, M., Herzog, C.: OnTourism: Semantic eTourism Portal, *Proceedings of the 2nd International Scientific Conference of the e-Business Forum - E-Business in Travel, Tourism and Hospitality*, Athens, Greece, 3 2008.

[6] Dogac, A., Kabak, Y., Laleci, G., Sinir, S., Yildiz, A., Kirbas, S., Gurcan, Y.: Semantically enriched web services for the travel industry, *SIGMOD Rec.*, **33**(3), 2004, 21–27, ISSN 0163-5808.

[7] Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog, *ACM TODS*, **22**(3), 1997, 364–418.

[8] Gallucci, L., Ricca, F.: Visual Querying and Application Programming Interface for an ASP-based Ontology Language, *Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07)* (M. D. Vos, T. Schaub, Eds.), 2007.

[9] Gelfond, M., Leone, N.: Logic Programming and Knowledge Representation – the A-Prolog perspective , *AI*, **138**(1–2), 2002, 3–38.

[10] Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases, *NGC*, **9**, 1991, 365–385.

[11] Joe, B., Carole, G.: TourisT: the application of a description logic based semantic hypermedia system for tourism, *HYPERTEXT '98: Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems*, ACM, 1998, ISBN 0-89791-972-6.

[12] Jorge, C.: Combining the semantic web with dynamic packaging systems, *AIKED'06: Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2006, ISBN 111-2222-33-9.

[13] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning, *ACM TOCL*, **7**(3), 2006, 499–562.

[14] Lixto:: `http://www.lixto.at//`.

[15] Maedche, A., Staab, S.: Applying Semantic Web Technologies for Tourism Information Systems, *Proceedings of the 9th International Conference for Information and Communication Technologies in Tourism, ENTER 2002, Innsbruck, Austria, 23 - 25th 2002* (K. Wber, A. Frew, M. H. (eds.), Eds.), 2002.

[16] Manna, M.: *Semantic Information Extraction: Theory and Practice*, Ph.D. Thesis, Dipartimento di Matematica, Universitá della Calabria, Rende, Cosenza Italia, 2008.

[17] Martin, H., Katharina, S., Daniel, B.: Towards the semantic web in e-tourism: can Annotation do the trick?, *Proceedings of the 14th European Conference on Information System (ECIS 2006)*, 2006.

[18] Martin, H., Katharina, S., Daniel, B.: Towards the Semantic Web in e-Tourism: Lack of Semantics or Lack of Content?, *Poster Proceedings of the 3rd Annual European Semantic Web Conference (ESWC 2006)*, 2006.

[19] Minker, J.: Overview of Disjunctive Logic Programming, *AMAI*, **12**, 1994, 1–24.

[20] Names:, G.: `http://www.geonames.org//`.

[21] Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., Leone, N.: OntoDLV: an ASP-based System for Enterprise Ontologies, *Journal of Logic and Computation*, 2009.

[22] Ricca, F., Leone, N.: Disjunctive Logic Programming with types and objects: The DLV$^+$ System, *Journal of Applied Logics*, **5**(3), 2007, 545–573.

[23] Ruffolo, M., Leone, N., Manna, M., Saccà, D., Zavatto, A.: Exploiting ASP for Semantic Information Extraction, *Proceedings ASP05 - Answer Set Programming: Advances in Theory and Implementation* (M. de Vos, A. Provetti, Eds.), Bath, UK, 2005.

[24] Ruffolo, M., Manna, M.: HiLeX: A System for Semantic Information Extraction from Web Documents, *ICEIS (Selected Papers)* (Y. Manolopoulos, J. Filipe, P. Constantopoulos, J. Cordeiro, Eds.), 3, 2008, ISBN 978-3-540-77580-5.

[25] Siorpaes, K., Bachlechner, D.: OnTour: Tourism Information Retrieval based on YARS, *Proceedings of ESWC 2006*, 2006.

[26] Smith, M. K., Welty, C., McGuinness, D. L.: OWL web ontology language guide. W3C Candidate Recommendation, 2003, `http://www.w3.org/TR/owl-guide/`.

[27] Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with Recursive Queries in Database and Logic Programming Systems, *TPLP*, **8**, 2008, 129–165.