# Designing New Ways for Selling Airline Tickets

Mladenka Vukmirović
Industry Development Department, Montenegro Airlines
Beogradska 10, 81000 Podgorica, Montenegro

Michał Szymczak
Department of Mathematics and Computer Science
Adam Mickiewicz University
Umultowska 87, 61-614 Poznań, Poland

Maciej Gawinecki
Systems Research Institute, Polish Academy of Science
Newelska 6, 01-447 Warsaw, Poland

Maria Ganzha,
Elblag University of Humanities and Economics, Elbląg, Poland
ul. Lotnicza 2, 82-300 Elbląg, Poland
and
Systems Research Institute, Polish Academy of Science
Newelska 6, 01-447 Warsaw, Poland

Marcin Paprzycki
Computer Science Institute, SWPS
Chodakowska 19/31, 03-815 Warsaw, Poland
and
Systems Research Institute, Polish Academy of Science
Newelska 6, 01-447 Warsaw, Poland

**Keywords:** software agents, air-travel ontology, travel support system, e-commerce, e-auctions, OTA, IATA

**Received:** October 12, 2006

*Large body of recent work has been devoted to multi-agent systems utilized in e-commerce; in particular, autonomous software agents participating in auctions. In this context we modify a model agent-based e-commerce system so that it can serve as an airline ticket auctioning system. Such a system can be then combined with a Travel Support System that utilizes ontologically demarcated travel-content. To achieve this goal, air travel data has to be demarcated utilizing an air travel ontology that has to support existing domain-specific real-world standards. One of such standards that steadily gains popularity in the air travel industry (and other travel areas) is the Open Travel Alliance (OTA) messaging system that defines, among others, the way that entities should communicate about air travel related issues. The aim of this paper is to outline our efforts leading toward creating an agent-based system for selling airline tickets that utilizes an air-travel ontology that matches the OTA messaging specification as well as satisfies procedures described in IATA manuals.*

*Povzetek: Opisan je večagentni system za prodajo letalskih kart.*

## 1 Introduction

Broadly understood e-commerce is often closely associated with software agents, which are to facilitate higher quality information, personalized recommendations, decision support, knowledge discovery etc. [27]. When developed and implemented, agent systems are to be, among others, adaptive, proactive and accessible from a broad variety of devices [42]; and as such are to deal autonomously with information overload (e.g. large number of e-shops offering the same product under slightly different conditions—price, delivery conditions, warranty etc.). Moreover, recent advances in auction theory have produced a general methodology for describing price negotiations [8, 9]. Combination of these factors gave new impetus to research on automating e-commerce [24]. In this context, we have started working on two independent research projects. The first one is devoted to the development of a model agent-based e-commerce system [2–5, 12 and references to our work cited in these papers]. In this system, we model a distributed marketplace where buyer agents approach e-stores and engage in price negotiations with seller agents. What makes our work unique is, among others, an attempt at conceptualizing not only price negotiations, but also a complete process from the moment when User-Cuyer

"decides" to make a purchase of product *P* to the successful purchase (or to a decision that such a purchase is impossible – e.g. due to the market conditions). The second project is an agent-based Travel Support System (TSS) [13, 39, 40]. In the TSS, travelers are to find complete support of their needs including, among others, items like restaurant information, historical points of interest, local weather etc. The central part of the TSS is a Jena-based repository [20, 21] that contains travel-related data is represented as RDF demarcated instances of a travel ontology [13]. Specifically, we have developed a complete ontology of a hotel (understood as a travel-related entity) and a restaurant; and then merged them [35]. The overarching goal of the design of the TSS was delivery of personalized information to users [13].

More recently we have asked, what would happen if our model e-commerce system had to be used in a more realistic scenario, where instead of an unspecified product *P,* airline tickets were to be sold and the system would have to interact with an actual airline reservation system. As a result we have proposed an augmented system in which two additional agents: a *FlightOffer Agent* (FOA) and a *Reservation Agent* were created to interact with Global Distribution Systems (GDS), e.g. AMADEUS or SABRE [1, 33] and facilitate delivery of all necessary air-travel related information.

In the next step we have considered how this augmented system could be integrated with the TSS. Since in the TSS travel data is stored as instances of travel ontologies (currently hotel and restaurant data), air travel related data should be also stored in the same way. Furthermore, air travel ontology that is to be used within the system should be tightly integrated with ontologies already existing in the system. After a thorough analysis of existing air-travel ontologies we have decided to develop our own [40].

The aim of this paper is to summarize our research results up to date. In the next section we present the augmented ticket auctioning system. We follow (in Sections 3 and 4) with a list of existing travel-related ontologies and a summary of the Open Travel Alliance (OTA) messaging system. OTA messages are then used as a starting point to design an air travel ontology, which is outlined in the next section.

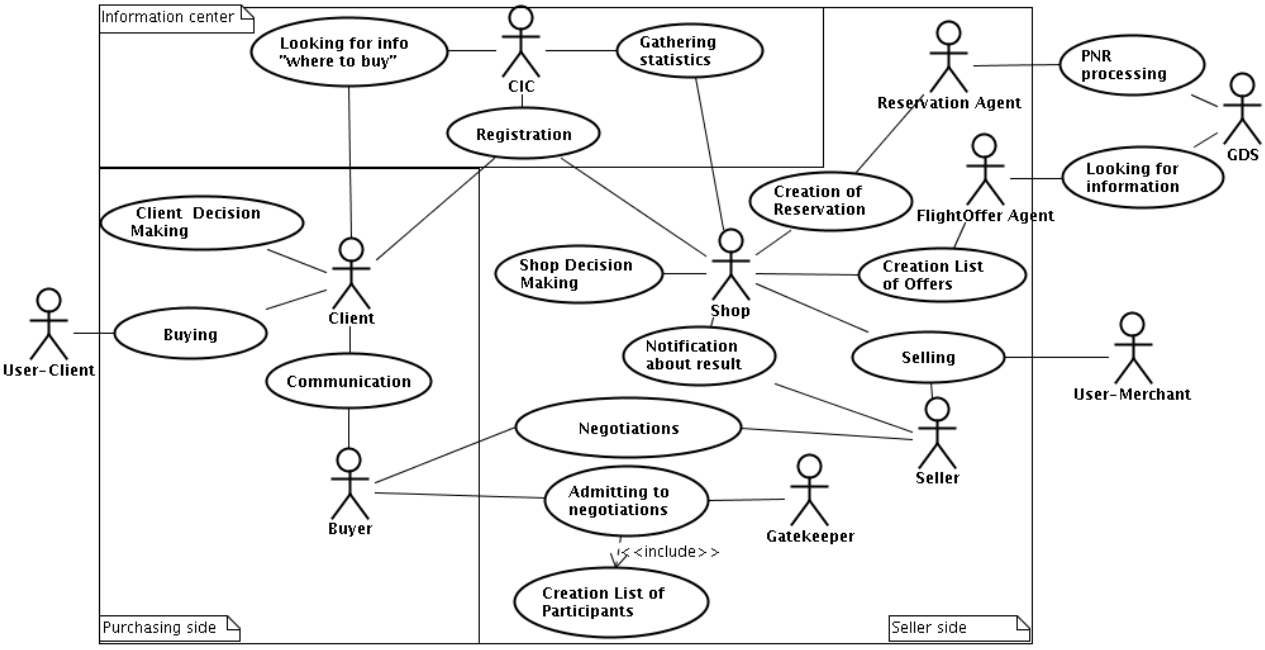


Figure 1: Airline ticket auctioning system – use case diagram.

## 2 Airline ticket auction system

Before proceeding with the description of the system, let us point some of the assumptions made in our work. (1) In our original agent-based e-commerce system e-stores were "drivers" within the marketplace. In other words, buyers could purchase only products that were available for sale through existing e-stores. We have decided, in the initial phase of our work on airline ticket selling system, to accept this approach (while planning to remove this limitation in the future). Therefore, in the augmented system, multiple "travel agencies" sell tickets to a variety of "popular destinations." They obey basic rules of airline ticket trading, but it is only "them" who decides which tickets to sell. Specifically, if the user of the system would like to fly from Tulsa, OK to San Diego, CA, she may not find such a connection. At the same time, connections between Amsterdam and Detroit, MI may be sold by every e-store. While this assumption may seem limiting, we would like to point out that success of priceline.com (and other auction places that sell airline tickets) makes our model scenario "realistic enough." (2) While we are utilizing the *CIC Agent* that stores "yellow-pages" (what?) and "white-pages" (who?) information as the approach to matchmaking [38], we see possible interesting extensions of its role in the system. It

could be possible to allow the *CIC Agent* to study market trends and sell this information to interested travel agencies. (3) In all situations where it was possible we utilize existing structures that have been described in [2–5, 10] and interested readers should consult these sources for further details.

Let us represent design of the system through its UML use case diagram in Figure 1 (detailed descriptions of the system can be found in [39]). We can distinguish three major parts of the system. (1) The *Information center area* where white-page and yellow-page information is stored and serviced by the *CIC Agent*. As specified above, currently, *User-Merchants* request that their e-stores sell tickets only for specific routes that they believe to be profitable. Each such route is advertised through the *CIC*. Every time the *Client Agent* is searching for an airline ticket for its *User-Client* it communicates with the *CIC* to find out which e-travel agencies sell it. (2) The *Purchasing side* where agents and activities representing the *User-Client* are represented. Here the *User-Client* informs its *Client Agent* which tickets she would like to purchase. While the *Client Agent* should be viewed as an incarnation of a *Personal Agent* [24] that knows preferences of its *User-Client* and autonomously acts on her behalf, their exact interrelations will be established in the future. *Client Agent* obtains from the *CIC* information which e-travel agencies sell requested tickets and sends a *Buyer Agent* to each one of them. *Buyer Agents* engage in price negotiations with *Seller Agents*. Successful price negotiations results in a reservation. *Client Agent* decides which agency to make a purchase from and, if the reservation did not expire and the tickets are still available in the GDS, they are purchased. (3) The *Seller side* involves *Shop Agent* acting on behalf of its *User-Merchant* and attempting at selling air tickets for routes defined by her. It interacts with the *FlightOffer Agent* in creating a list of specific offers that are registered with the *CIC*. Upon successful price negotiation the *Reservation Agent* creates and manages a reservation and, if this is to be the case, is responsible for completing the purchase. Observe that both the *FlightOffer Agent* and the *Reservation Agent* interact directly with the GDS. In this way they act as "wrapper agents" translating data between the outside world (the GDS) and the system. Let us now describe in more details the roles of these agents that have been added, or that act differently than in the original e-commerce system.

## 2.1 Shop Agent

*Shop Agent* (*SA*) acts as the representative of the *User-Merchant* and, at the same, time participates in the *Selling* function of the system. As specified above, in our current system design, it is the *User-Merchant* who specifies the input provided to the system. Specifically, for each route that is to be offered, she specifies: departure airport code, destination airport code, booking class, fare basis code, and the initial rule by which seats are to be offered for sale. For example, if *User-Merchant* wants to sell out seats that would have been offered for Advanced Purchase Excursion Fare—APEX [18, 19] but time limit for this fare has expired, *User-Merchant* would specify the number and the period for which she wants to offer seats on specific flights. This info would be used in availability check and price retrieval. The time-period would be needed to set bounds within which flights should be offered. Optionally *User-Merchant* can specify flight number as well. This narrows down the availability list and may be necessary in the case when there is more then one flight per day between two given destinations. Furthermore this can be used also in the case when, for instance, user-merchant wants to offer seats on morning flights, but not on evening flights. In this case she can specify which flight number(s) can be chosen from. In this way, all other possible flight numbers are excluded.

Obviously, it is possible to extend functionality of our system. For instance, while at present our system acts only as a "distributor" of a predefined set of tickets, it is possible to modify it in such a way that the *SA* could start distributing (acquire and put for auction) also tickets for routes that *User-Client*s are looking for. Since the *CIC* agent stores information about all unfulfilled *User-Client* queries, an *SA* could be enabled to obtain an access to this data (e.g. purchase it), analyze it and decide that, for instance, there is a growing need for tickets between Podgorica and Beijing and offer these for sale.

Statechart diagram of the *Shop agent* is depicted in Figure 2. At first the *SA* creates the *Gatekeeper Agent* (which plays here exactly the same role as described in [6]) and waits for a *User-Merchant* order. After receiving such an order the *SA* creates *FlightOffer Agent*, which communicates with the GDS and gathers needed information to create list of offers for the *Shop Agent* (one *FlightOffer Agent* is created for each route to be serviced and exists for as long as tickets for a given route are sold by the *SA*). List of offers includes information about every itinerary: data about both (inbound and outbound) flight numbers, number of seats and class of service for both flights etc. *Shop Agent* creates also *Seller Agent*(s), "introduces" them to the *Gatekeeper*, and enters a complex state called *Selling*. Note here that *Seller Agent*s play exactly the same role as that described in [6]; they are to interact with incoming *Buyer Agent*s and through some form of price negotiation mechanism (e.g. an auction) select the *Buyer* that may purchase the ticket. In the *Selling* state the *SA* is listening to its *Seller Agent*(s). After receiving a message from one of the *Seller Agent*s – informing about the result of price negotiations – the *Shop Agent* acts depending on content of that message.

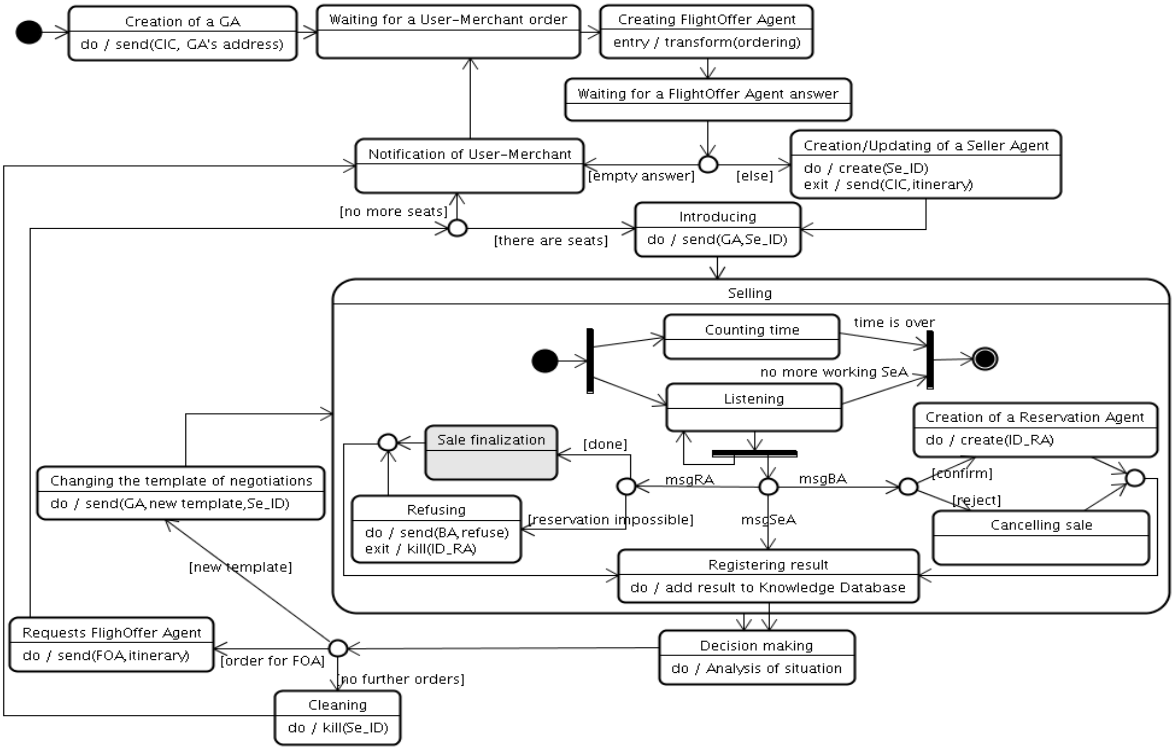


Figure 2: Shop Agent statechart diagram.

1. If the *Seller* informs the *SA* about a winner of price negotiations the *Shop Agent* waits for the corresponding *Buyer Agent* to confirm that it plans to actually buy the ticket (see also [10] for more details). Here, we have to stress, that in our general e-commerce model it is natural that multiple *Buyer Agent*s visit multiple e-stores [10]. Specifically, separate *Buyer* visits each e-travel agency that offers ticket(s) satisfying needed itinerary. The end of price negotiation means that the *Buyer* should consult with the *Client Agent*. Therefore, the *SA* does not know if the winner of price negotiations will actually attempt at making a purchase.
2. If the *Buyer Agent* confirms it wants to buy a ticket, the *Shop Agent* creates a *Reservation Agent* (*RA*), which communicates with the GDS to make a reservation. There are then the following possibilities:
   - If the *RA* was able to reserve tickets (it is possible that while the negotiations were taking place all tickets available in a given class of service etc. are already gone), it sends the reservation data to the *Shop Agent*. Upon reception of the data (all communication in the system is carried using ACL messages) the *Shop Agent* transfers it further to the *Buyer Agent* and carries out standard procedures involved in completing the sale (Figure 1, state "Sale finalization").
   - In the opposite case (the *RA* was not able to secure the reservation) the *Shop Agent* notifies the *Buyer Agent* that reservation is impossible and kills the *Reservation Agent*.
3. If the Buyer Agent sends message that it does not want to make a purchase, this fact is registered in a local Knowledge Database. More precisely, all information about processes that take place within the shop when it is attempting to sell tickets is recorded in the Knowledge Database. In the future, this information will be used by the *SA* to adapt its behavior. Currently we denote this fact by introducing the *Decision Making* box, which denotes multi-criterial decision making. For instance, one of important factors that influences the way that the *SA* interacts with incoming *BA*s is trust (see for instance [7, 28]). It should also be mentioned that in our system we utilize a modified negotiation framework [3, 4, 6] introduced originally by Bartollini, Jennings and Preist [8, 9]. In this framework, the negotiation process was divided into a generic *negotiation protocol* and a *negotiation template* that contains parameters of a given negotiation. These parameters specify, among others, the price negotiation mechanism itself. Observe, in Figure 2, that one of possible results of *Decision Making* is change of the negotiation template. In other words, the *SA* may decide that since only very few tickets are left but there is also only very short time to sell them, it will deeply discount them and sell them with a fixed price, or through a very short time lasting English auction with a low threshold value and a relatively large increment.
4. If there is no winner, the *Shop Agent* writes information into the *Knowledge Database* and starts to analyze the current situation (the *Decision Making* box in Figure 2). As a result it may change the negotiation template, or request another itinerary from the *FlightOffer Agent*. Finally, it may establish that for that given route (*User-Merchant* order) either there is nothing more to do (all tickets have been sold) or that nothing can be done (the remaining tickets cannot be sold in the current condition of the market). Then it will remove all

"servant" agents servicing that route and inform its *User-Merchant* about the situation. It is important to note that we assume that in all price negotiation mechanisms the *Seller* institutes a time limit for negotiations. This moment is presented within the *Shop Agent* diagram as a sub-state "Counting time" (within the *Selling* state). If the *Seller* does not sell any tickets within that time the *Shop Agent*, again, registers this information in the *Knowledge Database*, kills this *Seller* and notifies its user-merchant accordingly. Following, the *SA* enters the *Multi-criterial Decision Making* state. As described above, here it can decide, among others, to sell more seats on some specific itinerary or to change the template of negotiations or to conclude that nothing more can be sold and its existence should be completed.

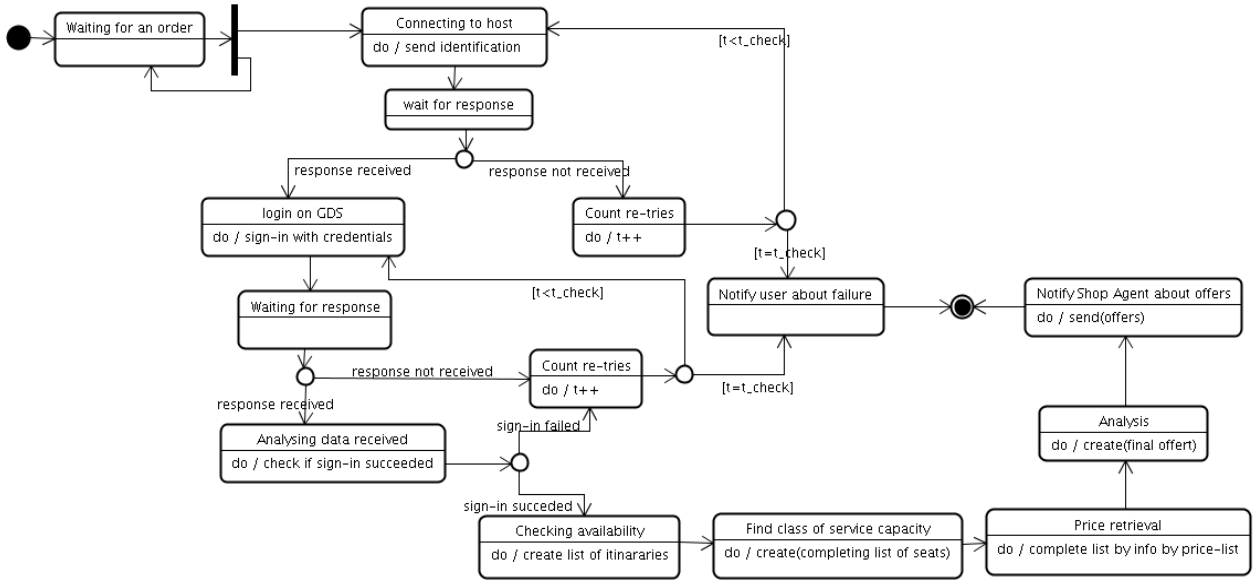


Figure 3: *FlightOffer Agent* statechart diagram.

## 2.2 *FlightOffer* and *Reservation Agents*

These two agents have been added to the system and their role is to communicate with the GDS. The statechart diagram of the *FlightOffer Agent* is presented in Figure 3.

This agent communicates with the GDS to find information about flights that satisfy conditions specified by the *User-Merchant*. If such flights are available the *FlightOffer Agent* prepares (process represented by actions that are enclosed within multi-state boxes *Checking availability*, *Find Class of service capacity*, *Price retrieval* and *Analyzing module*) a *List of Offers* for the *Shop Agent*. All the multi-state states—*Checking availability*, *Find Class of service capacity*, *Price retrieva*l and *Analyzing module*—involve communication with the GDS. In Figure 4 we present the statechart of the *Price retrieval* sub-state to illustrate the nature of proposed communications between the *FlightOffer Agent* and the GDS. Upon obtaining all the necessary information form the GDS it sends the information to the *Shop Agent*. Note that the role of the *Analyzing module* is to check the request of the *User-Merchant* against the data retrieved from the GDS to assure consistency of the final offer (e.g. if the *User-Merchant* requested 10 seats, but only 5 are available then only 5 can be in the offer). The second agent that communicates with the GDS is the *Reservation Agent*. It is created by the *Shop Agent* after receiving, from the *Buyer Agent*, confirmation of willingness to make a purchase. Its function is to make an actual reservation within the *GDS* server. In case of successful completion of its task the *Reservation Agent* transfers all reservation's data to the *Shop Agent*. If the reservation is impossible it informs about it the *Shop Agent*. Both cases mean that its job is complete and it then self-destructs.

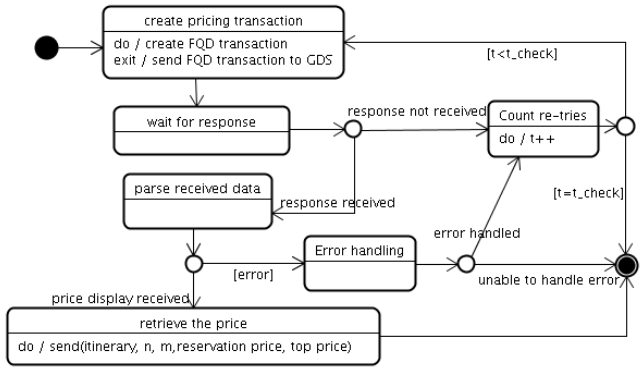


Figure 4: *FlightOffer Agent*'s *Price retrieval* sub-state statechart diagram

Let us now consider the question of integrating this system with the Travel Support System (TSS). While there is a number of interesting questions that would have to be addressed, the one that we are concerned with in this paper is as follows. In the TSS all data is stored in the system in a semantically demarcated

fashion. Furthermore, we envision the augmented e-commerce system as comprising a number of e-travel agencies that utilize methods developed there to sell airline tickets for selected routes. In this case we have to deal with the following situation. Data stored in and provided by the *GDS* is not ontologically demarcated. Hotel and restaurant information stored in the travel agency is ontologically demarcated. Therefore, to be able to combine these two systems one has to provide travel agencies in the e-commerce system with: (1) air-travel ontology, that should be integrated with the two already developed ontologies, and (2) way of translating data provided by the GDS into an appropriate form for the travel agency and the GDS to "understand" each-other. In the remaining parts of this paper we address the first issue, while in the concluding remarks we sketch our proposed solution of the second one.

# 3 General and travel ontologies

As the first step in the direction of being able to utilize an air travel ontology, we have researched the existing available ontologies.

While the largest general ontology building projects, such as (1) the Cyc project [31], (2) WordNet [41], (3) Suggested Upper Merged Ontology (SUMO) [36], and (4) SENSUS [34] do not provide us with an "ontology of travel," there exist a number of smaller scale attempts at defining such an ontology. (1) Mondeca´s [30] tourism ontology defines tourism concepts based on the WTO thesaurus. (2) The Travel Agent Game in Agentcities (TAGA) is an agent framework for simulating the global travel market on the Web. Its purpose is to demonstrate Agentcities and Semantic Web technologies [37]. In addition to the FIPA content language ontology, TAGA defines (a) basic travel concepts such as itineraries, customers, travel services, and service reservations, and (b) different types of auctions, roles participants play in them, and protocols used. (3) Harmonize is an attempt at ontology-mediated integration of tourism systems following different standards [15]. Its goal is to allow organizations to exchange information without changing data structures. The Harmonize project also involves sub-domains that are only partially related to the world of travel: geographical and geo-spatial concepts, means of transportation, political, temporal, activity/interest, gastronomy etc. These sub-domain concepts can be used within the travel system (directly, as needed) or incorporated into the ontology constructed for the system. It is claimed that the next generation of "eTourism" will be powered by the Semantic Web technology (resulting in an eTourism Semantic Web portal which will connect the customers and virtual travel agents from anywhere at anytime). Goes with out saying that this is a very interesting project, however, airline ticket sales are not included in the current version of Harmonize ontology. (4) Finally, a number of "minimalist" travel ontologies can be found within the DAML language portal [11]. For instance, the Itinerary-ont is an ontology for representing travel itineraries. It reuses the airport codes ontology and involves definitions of terms like Aircraft, Class, Flight etc. Another example is the Trip Report Ontology that defines Airfare, Amount, Date, etc., and models on-line sale.

The complete list of pros and cons for ontologies listed above may be found in [40]. There, we report results of our in-depth analysis of the possibility to utilize any of them in airline ticket sales. Overall, none of them had a fully developed air travel part and that could also interface with an actual GDS, and therefore we had to develop our own, based on the Open Travel Alliance messaging system.

# 4 OTA and OTA Air Messages

The Open Travel Alliance (OTA) is a non-profit organization working to establish a common electronic vocabulary for exchange of travel information. Such an exchange is to take form of standardized eXtensible Markup Language (XML) messages. OTA specifications have been designed to serve: (a) as a common language for travel-related terminology, and (b) as a mechanism for exchange of information between travel industry members [14]. The OTA Air Messages standard, which is of particular interest in our work, specifies structure and elements of different scenarios involved in selling air travel tickets. Let us note that since this is a specification of messaging, it does not cover any other operations involved in selling air-tickets (e.g. airfare calculations). These operations have to be treated separately. OTA messages have been proposed as pairs of request and response messages (RQ / RS below). Let us summarize their main features (their complete description can be found in [32]).

***OTA_AirAvailRQ/RS*** – establishes airline flight availability for a city pair, specific date, specific number and type of passengers. The request can also be narrowed to a specific airline, flight or booking class. Optional requested information can include: time / time window, connecting cities, client preferences (airlines, cabin, flight types etc.). The response message (*RS*) contains flight availability. Furthermore, a set of origin and destination options is returned, each of which contains one or more (connecting) flights that serve that city pair. For each flight information about: origin and destination airports, departure and arrival date/times, booking class availability, equipment, meal information and code-share information is returned.

***OTA_AirBookRQ/RS*** – requests to book a specific itinerary for one or more identified passengers. The message contains optional pricing information, allowing the booking class availability and pricing to be rechecked as part of the booking process. Optional requested information can include: seat and meal requests, Special Service Requests (SSR), Other Service Information (OSI), remarks, fulfillment

information – payment, delivery details, type of ticket desired. If booking is successful, the RS message contains the itinerary (including the directional indicator, status of booking, and number of passengers), passenger and pricing information sent in the request, along with a booking reference number (PNR Locator) and the ticketing information. The RS echoes back received information with additional information – booking reference from the GDS through which reservation was created.

***OTA_AirFareDisplayRQ/RS*** – allows a client to request information on fares, which exist between a city pair for a particular date or date range. No inventory check for available seats on flights is performed by the server before the RS is send back. The request can optionally contain information indicating that a more specific response (e.g. passenger information, specific flight information and information on the types of fares that the client is interested in) is required. The RS message repeats *FareDisplayInfo* elements, each of which contains information on a specific fare contract including airline, travel dates, restrictions and pricing. It can also return information on other types of fares that exist, but have not been included in the response.

***OTA_AirFlifoRQ/RS*** – requests updated information on the operation of a specific flight (it requires the airline, flight number and departure date; the departure and arrival airport locations can be also be included). The RS includes real-time flight departure and arrival information. It also includes: departure airport, arrival airport, marketing and operating airline names; when applicable, flight number, type of equipment, status of current operation, reason for delay or cancellation, airport location for diversion of flight, current departure and arrival date and time, scheduled departure and arrival date and time, duration of flight, flight mileage, baggage claim location.

***OTA_AirLowFareSearchRQ/RS*** – requests priced itinerary options for flights between specific city pairs on certain dates for a specific number and types of passengers. Optional requested information can include: time / time window, connection points, client preferences (airlines, cabin, flight types etc.), flight type (nonstop or direct), number of itinerary options desired. The RS contains a number of *Priced Itinerary* options. Each includes: a set of available flights matching the client's request, pricing information including taxes and full fare breakdown for each passenger type, ticketing information – ticket advisory information and ticketing time limits, fare basis codes and the information necessary to make a rules entry.

***OTA_AirPriceRQ/RS*** – requests pricing information for specific flights on certain dates for a specific number and type of passengers. The message allows for optional information such as fare restriction preferences and negotiated fare contract codes to be included. The pricing request contains information necessary to perform an availability / sell from availability / price series of entries for an airline CRS or GDS. The RS contains a *Priced Itinerary* that includes: set of flights, pricing information including taxes and full fare breakdown for each passenger type, ticketing information, fare basis codes and the information necessary to make a fare rules entry.

***OTA_AirRulesRQ/RS*** – requests text rules for a specific fare basis code for an airline and a city pair for a specific date. Negotiated fare contract codes can be included in the request. The RS contains a set of, human readable, rules, identified by their codes.

***OTA_AirSchedulesRQ/RS*** – provides customer, or a third party, with ability to view flight schedules. It requires specification of the departure and arrival cities and a specific date. It offers flight information on airlines that provide service between requested cities and could be used when customer: (1) wants to determine what airlines offer service to/from specific destinations, (2) is looking for a specific flight number – by entering the arrival and departure cities, and the approximate arrival or departure time, specific flight number can be found, (3) needs to determine the days of the week that service is scheduled to and from requested destinations, (4) wants to determine aircraft type used to fly that route. Message may request other information that customers are interested in: meal service, duration of flight, on-time statistics and if smoking is allowed. In addition, these messages provide foundation for electronic timetables.

***OTA_AirSeatMapRQ/RS*** – displays seats available on a given flight, as well as their location within the aircraft. It is used o make seat assignments as it identifies all information necessary to request and return an available seat map for a particular flight. Types of information for the seat map request include: airline, flight number, date of travel, class of service and frequent flier status. The RS includes: flight, aircraft and seat description information.

***OTA_AirBookModifyRQ/OTA_AirBookRS*** – requests to modify an existing booking file. It contains all elements of the *OTA_AirBookRQ* plus a general type of modification, i.e. name change, split, cancel or other; as indicated with the attribute *ModificationType*. The modification operation on different elements is either indicated with the existing attribute *Status* (for air segments, SSR's and seat requests) or with attribute *Operation* of type *ActionType* for other elements (i.e. other service information, remarks or *AirTraveler* elements). In the *AirBookModifyRQ*, all data to be changed is submitted and in the *AirReservation* element all existing data may be submitted. This allows the receiving system to perform a consistency check before updating the booking file (but to keep the message small, this part can be omitted). Changes to a booking (1) may result in required updates of the ticket (e.g. revalidation), (2) may imply charges for the change, (3) the pricing may change, and/or (4) some fees may need to be collected. Pricing and fulfillment details required to achieve results of

*AirBookModify* ticketing, are out of scope and are omitted. The RS confirms changes in the itinerary.

# 5 Proposed ontology

As indicated above, in Section 3 and in our research [33, 34] we have established that existing air-travel ontologies have been designed mostly as "academic" demonstrator systems – rather than with the goal of actually working within the context of real-life airline reservation systems – and this explains lack of important features when it comes to dealing with genuine air travel data. According to our best knowledge, the only project that actually involves airline industry is the OTA specification (which, as stated above, is only a messaging specification). Therefore, we decided to create new ontology that would: (1) utilize International Air Transport Association (IATA) [14-19] mandated data descriptions and recommended practices; (2) utilize as much as possible from existing travel ontologies – as long as they follow IATA practices, (3) match features included in the OTA specification, and (4) be synchronized with our existing travel ontology. To achieve this goal we have applied a bottom-up approach and our initial goal was to model reservations as defined in the AMADEUS global distribution system.

In the proposed ontology we have divided main classes into following groups: *AirTravelCodes, AirTravel, AirInfrastructureCodes and AirInfrastructure. AirInfrastructure* group encloses most basic terms related to air travel industry such as *Airline*, *Airplane* and *Airport*. While all three are defined in line with specifications presented in [14, 15, 19], the latter one (*Airport*) is a subclass of our *OutdoorLocation* class that was designed for the TSS [11]. In this way it is possible for the traveler to obtain more data regarding the airport than the city name, which usually is the only information that can be obtained from other airline travel related ontologies. Specifically, the *TSS* offers *OutdoorLocation* class that includes, among others, such details as geographical, urban location, address details, nearby attractions etc. To illustrate the results, let us present here the n-triples for this class:

```
base:OutdoorLocation a rdfs:Class;
  rdfs:subClassOf geo:SpatialThing;
  rdfs:comment "Outdoor location.
Geographical and urban references.".

base:address a rdf:Property;
  rdfs:comment "Address details.";
  rdfs:domain base:OutdoorLocation;
  rdfs:range adrec:AddressRecord.

base:attractionCategory a rdf:Property;
  rdfs:comment "Nearby attractions.";
  rdfs:domain base:OutdoorLocation;
  rdfs:range base:AttractionCategoryCode.

base:indexPoint a rdf:Property;
  rdfs:comment "Reference map point.";
  rdfs:domain base:OutdoorLocation;
  rdfs:range base:IndexPointCode.

base:indexPointDist a rdf:Property;
  rdfs:comment "Distance from the reference
map point.";
  rdfs:domain base:OutdoorLocation;
  rdfs:range base:IndexPointCode.

base:locationCategory a rdf:Property;
  rdfs:comment "Location category.";
  rdfs:domain base:OutdoorLocation;
  rdfs:range base:LocationCategoryCode.

base:neighbourhood a rdf:Property;
  rdfs:label "Neighbourhood";
  rdfs:comment "The neighborhood of the
Outdoor
               location.";
  rdfs:range xsd:string;
  rdfs:domain base:OutdoorLocation.

base:crossStreet a rdf:Property;
  rdfs:label "Cross street";
 rdfs:comment "The nearest street that
crosses the street that the restaurant is
on.";
  rdfs:range xsd:string;
       rdfs:domain base:OutdoorLocation.

base:AttractionCategoryCode a rdfs:Class;
  rdfs:comment "Possible categories of places
which might be of interest for
visitors/guests and can be found in the
neighborhood." .

base:IndexPointCode a rdfs:Class;
  rdfs:comment "Possible reference map
points.".

base:LocationCategoryCode a rdfs:Class;
  rdfs:comment "Possible location
categories.".
```

As our system needs recognition of IATA codes to fulfill its aim, we have added three-letter IATA airport code as a property of our class. These codes are represented with a separate class *AirportCode* that was based upon the DAML *AirportCodes* class from the Itinerary-ont ontology, shortly described in Section 3. In this way we were able to offer more complete information about airport and to include information that other ontologies also provide. Following is the N3 notation based depiction of the *Airport* class:

```
base:Airport a rdfs:Class;
 rdfs:subClassOf loc:OutdoorLocation;
 rdfs:comment "Used for airport's city and
geographical location description".

base:airportCode a rdf:Property;
 rdfs:domain base:Airport;
 rdfs:range apc:AirportCode.
```

For the sake of clarity, let us provide the definition and some instances of our *AirportCode* class.

```
base:AirportCode a rdfs:Class;
rdfs:comment "Represents three letter code of
an airport".

#instances of AirportCode class
base:TGD a base:AirportCode.
```

```
base:WAW        a       base:AirportCode.
base:LIS        a       base:AirportCode.
base:MOW        a       base:AirportCode.
```

*AirInfrastructureCodes* group contains, used in other classes, codes for airports and countries. Included classes are *ISOCountryCode* and *AirportCode*. *AirTravelCodes* group comprises industry codes used in GDSs and CRSs for the itinerary reservation and the ticket issuance: *IATATicketIndicator*, *IATAStatusCode*, *CabinClass*, *BookingClass*, *IATAFareBasis*, *MealCode*, *SSRCode*, *SSRMealCode*, *TicketDestignator* (details can be found in [16-21]). Finally, the *AirTravel* group takes care of upper-level terms that define more complex objects used in the air travel systems. Following classes are included in this group: *OfficeID*, *TerminalID*, *AgentCredentials* – that define credentials of the GDS/CRS user, *AvailabilityDisplay* – that defines available flight options for a certain route, *Flight* – with usual properties together with status statistics, *IATAItinerary* – that defines itinerary for the passenger, *PNR* – Passenger Name Record or, simply described, a reservation with all details of the passenger, the itinerary, special requests and the GDS/CRS locator code, *Pricing* – that describes available prices for a certain route with or without taxes included, *SeatMapPlan* – for a certain flight, *Tariff* - with *Category* properties that are coded as in the *ATPCO*'s (Airline Tariff Publishing Company) recommendation, and *TimetableDisplay* – with timetable of different airlines for a certain route.

As stated above some classes were inherited or used as upper level classes from the TSS. These classes were: *OutdoorLocation*, *IATADiscountCodes**, *MeanOfPayment*, *FareTax*, *Discounts*, *DiscountCodes*, *IATATaxCodes**, *NameRecord*, and *PersonTitle*. Marked with * are classes that were sub classed from classes inherited from the TSS.

One additional, very important, concept in traveling is currency. At first we designed a very simple class that contained only the currency code. Promptly this showed to be insufficient as air travel currency application involved some complicated restrictions. As in the case of air travel ontology, we made an effort to find an already existing ontology of currency, and inject it into our project. We studied several currency ontologies (more details can be found in [35]) and found out that ontology used in Cambia web-service [10] was the most appropriate one. Unfortunately, it was rather broad, and furthermore we had to modify it so that it could be used for currency conversion guided by the IATA conversion rules [21].

| OTA message | OTA message element | Related properties of Tariff class from our ontology |
|---|---|---|
| OTA_AirFareDisplayRS | FareDisplayInfo ***attributes*:** | Tariff **class properties:** |
| | • *FareApplicationType* | • *FareAplicationType* |
| | • *ResBookDesigCode* | • *bk* **range** BookingClass |
| | • *MilageIndicator* | • *milageInd* |
| | • *FareStatus* | • *fareStatus* |
| | FareReference | *farebasis* **range** IATAFareBasis **class** |
| | RuleInfo ***subelements*** | Tariff **class properties** (rules)**:** |
| | • *MinimumStay* | • *_06* **range** StayLength **class** |
| | • *MaximumStay* | • *_07* **range** StayLength **class** |
| | FilingAirline | *carrier* **range** Airline **class** |
| | DepartureLocation ***attribute*** *LocationCode* | *origin* **range** Airport **class** |
| | ArrivalLocation ***attribute*** *LocationCode* | *destination* **range** Airport **class** |
| | Restriction ***attributes*** | Tariff **class properties** |
| | • *GlobalIndicatorCode* | • *globaldirection* |
| | • *MaximumPermittedMilage* | • *mpm* |
| | PricingInfo ***attributes*** | Tariff **class properties** |
| | • *NegotiatedFare* | • *_35* |
| | • *PassengerTypeCode* | • *paxtype* |
| | • *TicketingDestignatorCode* | • *bk* |
| | BaseFare ***attributes*** | Tariff **class properties** |
| | • *Amount* | • *ow*, *rt* |
| | • *CurrencyCode* | • *currency* **range** Currency **class** |
| | • *DecimalPlaces* | ***Defined under Currency class*** |

Table 1: *Matching the OTA message with the air-travel ontology.*

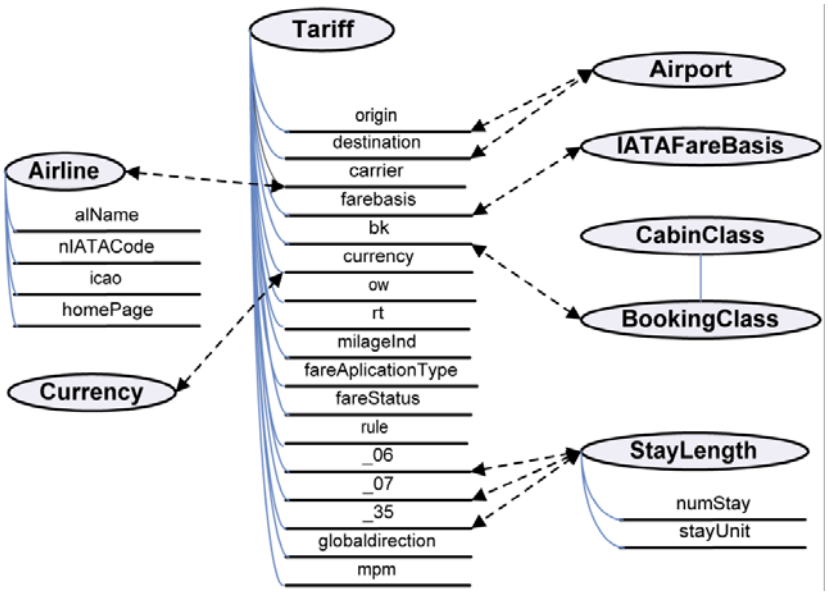


Figure 5: Protégé display of *Tariff* class.

Let us stress that since the OTA was defined as a messaging system used for information exchange, while the proposed ontology was created with intention to describe persistent data in our system, therefore quite often more then one class from our ontology has to be used in association with a single OTA message. As request (RQ) messages contains only data used to make a query, let us illustrate how the RS message matches with the proposed ontology in the case of the *OTA_FareDisplaylRS*. In our ontology an equivalent class is *Tariff*. In Table 1 we depict how elements of the message match elements of our ontology. Furthermore, Figure 5 shows relations of the *Tariff* class with other classes (*Airline, Airport, IATAFareBasis, StayLength, BookingClass*) from our ontology.

Finally, one of the major advantages of utilizing the ontology technologies to demarcate electronic data is that it provides us with a highly readable, customizable and scalable knowledge (data) model. This allows us, among others, to swiftly browse the travel related content, based on the ontology concept references. Figure 6 presents such references between Hotel, Airport, Restaurant, OutdoorLocation, Currency and Person concepts. Obviously, the TSS ontology and its air-ticketing-dedicated extension contain far larger number of inter-concept references; however, presenting them within a single figure would greatly limit its readability.

# 6 Concluding remarks

In this paper we have summarized results obtained thus far in our attempt in developing an agent-based airline ticket selling system. We have started from presenting an augmented version of a model agent-based e-commerce system and followed with a suggestion that such a system should be merged with an agent-based Travel Support System that we are also developing. To achieve this goal it was necessary to develop ontology of air travel. Based on our analysis of existing travel ontologies we have decided to develop our own ontology that is based on IATA manuals and OTA messaging system. As a result, in this paper we have illustrated how an ontology can be extracted from OTA messages. Overall, when completed (currently, the proposed merged travel ontology it is available for comments at: http://agentlab.swps.edu.pl) our (air) travel ontology should be capable of being used to interface our Travel Support System with an actual GDS (which is one of important goals of our project).

Let us note that there exist already GDS's that allow communication using OTA messaging. Leading this development, AMADEUS in its newly created platform called 'Results CMS' aimed at lowering cost of operations and offered OTA messaging as a way to distribute airline inventory to external travel sites and dynamic package providers. Therefore, as the next step of our research, we plan to develop two parsers. Let us assume that a query that is related to air-travel has been formulated in our system. Obviously, this will be a SPARQL query (as SPARQL is our language of choice to query ontologically demarcated content stored in Jena repository). This query will then be translated into an OTA message and submitted to the GDS. Such a translation will be based on our air-travel ontology. As a response, the GDS will send an OTA response message, containing requested information. This message will be then parsed and information translated into instances of our air-travel ontology. We will then use our display system [25] to present them to the user. We will report on our progress in subsequent publications.

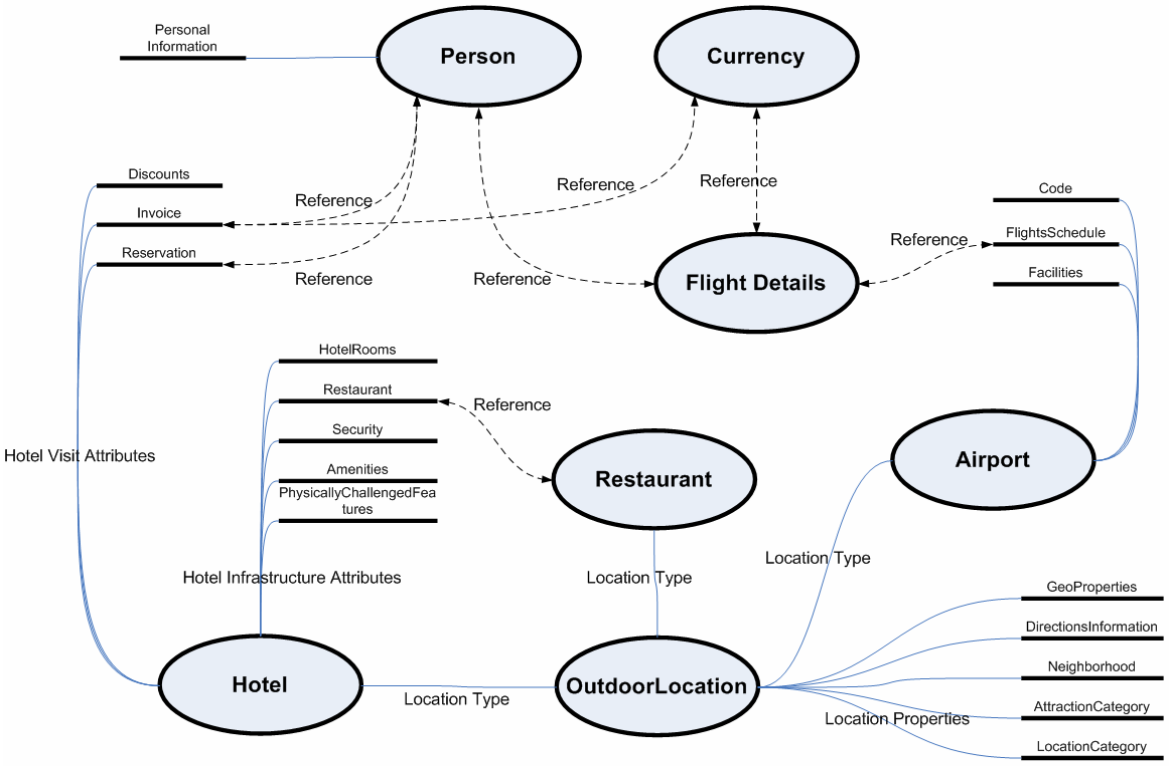


Figure 6: Ontology concept references

## Acknowledgement

We want to thank Mr Zoran Djurišić, the President of Board of Directors of Montenegro Airlines for his support of this research. Work of Maria Ganzha, Maciej Gawinecki and Marcin Paprzycki was partially sponsored by the EU IRG grant – project E-CAP.

## References

[1] AMADEUS, http://www.amadeus.com/

[2] Bădică, C., Badita, A., Ganzha, M., Iordache, A., Paprzycki M.: Implementing Rule-based Mechanisms for Agent-based Price Negotiations. In: Proceedings of the SAC'2005 Conference (in press)

[3] Bădică, C., Ganzha, M., Paprzycki, M., Pîrv˘anescu, A.: Combining Rule-Based and Plug-in Components in Agents for Flexible Dynamic Negotiations. In: M. P˘echou˘cek, P. Petta, and L.Z. Varga (Eds.): Proceedings of CEEMAS'05, Budapest, Hungary. LNAI 3690, Springer-Verlag, pp.555-558, 2005.

[4] Bădică, C., Ganzha, M., Paprzycki, M., Pîrvănescu, A.: Experimenting With a Multi-Agent E-Commerce Environment. In: V. Malyshkin (Ed.): Proceedings of PaCT'2005, Krasnoyarsk, Russia. LNCS 3606, Springer-Verlag, pp.393-402, 2005.

[5] Bădică, C., Bădită, A., Ganzha, M., Paprzycki, M., Developing a Model Agent-based E-commerce System, in: Jie Lu et. al. (eds.) E-Service Intelligence - Methodologies, Technologies and Applications (to appear)

[6] Bădică C., Ganzha M., Paprzycki M., UML Models of Agents in a Multi-Agent Ecommerce System. In: Proceedings of the ICEBE 2005 Conference, IEEE Press, Los Alamitos, CA, 56-61

[7] Costin Bădică, Maria Ganzha, Maciej Gawinecki, Pawel Kobzdej, Marcin Paprzycki (2006) Towards Trust Management in an Agent-based E-commerce System - Initial Considerations. In: A. Zgrzywa (ed.) Proceedings of the MISSI 2006 Conference, Wroclaw University of Technlogy Press, Wroclaw, Poland, 225-236

[8] Bartolini, C., Preist, C., Jennings, N.R.: Architecting for Reuse: A Software Framework forAutomated Negotiation. In: Proceedings of AOSE'2002: Int. Workshop on Agent-Oriented Software Engineering, Bologna, Italy, LNCS 2585, Springer Verlag, pp.88-100, 2002.

[9] Bartolini, C., Preist, C., Jennings, N.R.: A Software Framework for Automated Negotiation.In: Proceedings of SELMAS'2004. LNCS 3390, Springer-Verlag, pp.213-235, 2005.

[10] Cambia Service, http://zurich.agentcities.whitestein.ch/Services/Cambia.html

[11] DAML Ontologies, http://www.daml.org

[12] Ganzha, M., Paprzycki, M., Pîrvănescu, A., Bădică, C, Abraham, A.: JADE-based Multi-Agent E-commerce Environment: Initial Implementation, In: Analele Universităţii dinTimişoara, Seria Matematică-Informatică, 2005 (to appear).

[13] Gordon M., Paprzycki M., Designing Agent Based Travel Support System. In: Proceedings of the ISPDC 2005 conference, IEEE Computer Society Press, Los Alamitos, CA, 2005, 207-214

[14] http://www.opentravel.org/about.cfm

[15] Harmonize, http://deri.at/research/projects/e-tourism

[16] IATA Airline Coding Directory – Airline Designators, 70th Edition
[17] IATA City Code Directory, 43rd Edition, Effective 9 December 2005 – 31 December 2006
[18] IATA Passenger Services Conference Resolutions Manual, 24th Edition, Effective 1 June 2005 – 31 May 2006
[19] IATA Passenger Tariff Coordination Conferences Manual, Composite, Dec 9, 2005 until Dec 31, 2006
[20] IATA Reservation Service Manual, 23rd Edition
[21] IATA Standard Schedules Information Manual, Mar 1, 2006 until Sep 30, 2006
[22] Jena 2 Ontology API – General concepts, http://jena.sourceforge.net/ontology/index.html#generalConcepts
[23] Jena Documentation, http://jena.sourceforge.net/documentation.html
[24] Kowalczyk, R., Ulieru, M., Unland, R.: Integrating Mobile and Intelligent Agents in Advanced E-commerce: A Survey. In: Agent Technologies, Infrastructures, Tools, and Applications for E-Services, Proceedings NODe'2002 Agent-Related Workshops, Erfurt, Germany. LNAI 2592, Springer-Verlag, pp.295-313, 2002.
[25] Maciej Gawinecki, Minor Gordon, Paweł Kaczmarek, Marcin Paprzycki (2003) The Problem of Agent-Client Communication on the Internet. Scalable Computing: Practice and Experience, 6(1), 2005, 111-123
[26] Maes P., Agents that Reduce Work and Information Overload. Communications of the ACM, 37, 7, 1994, 31-40
[27] Maes, P., Guttman, R.H.,Moukas, A.G.: Agents that Buy and Sell: Transforming Commerce as we Know It. In Communications of the ACM, Vol.42, No.3, pp.81-91, 1999.
[28] Maria Ganzha, Maciej Gawinecki, Pawel Kobzdej, Marcin Paprzycki, Costin Bădică (2006) Functionalizing trust in a model agent based e-commerce system. In: M. Bohanec et. al. (eds.), Proceedings of the 2006 Information Society Multiconference, Josef Stefan Institute Press, 22-26]
[29] Mladenka Vukmirović, Marcin Paprzycki, Michał Szymczak (2006) Designing ontology for the Open Travel Alliance Airline Messaging Specification, In: M. Bohanec et. al. (eds.), Proceedings of the 2006 Information Society Multiconference, Josef Stefan Institute Press, 101-105]
[30] Mondeca, http://www.mondeca.com
[31] OpenCyc, http://www.opencyc.org
[32] OpenTravelTM Alliance, Message Users Guide. 2005B Version 1.0, 2 December 2005
[33] SABRE, http://www.sabre.com/
[34] SENSUS, http://www.isi.edu/natural-language/projects/ONTOLOGIES.html
[35] Szymczak M., Gawinecki M., Vukmirović M., Paprzycki M., Ontological reusability in state-of-the-art semantic languages, Proceedings of the XVIII Summer School of PIPS (to appear)
[36] SUMO, http://www.ontologyportal.org
[37] TAGA, http://www.agentcities.org
[38] Trastour, D., Bartolini, C., Preist, C.: Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In: Proceedings of the WWW'02: International World Wide Web Conference, Hawaii, USA. ACM Press, New York, USA, pp.89-98, 2002.
[39] Vukmirović M., Ganzha M., Paprzycki M.: Developing a Model Agent-based Airline Ticket Auctioning System. In: Proceedings for the IIPWM Conference, LNAI
[40] Vukmirović M., Szymczak M., Ganzha M., Paprzycki M.: Utilizing Ontologies in an Agent-based Airline Ticket Auctioning System. In: Proceedings of the 28th ITI Conference, IEEE Computer Society Press, Cavtat, Dubrovnik, Croatia, 385-390
[41] WordNet, http://www.daml.org/ontologies/196
[42] Wooldridge, M.: *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002.