

THE PROBLEM OF AGENT-CLIENT COMMUNICATION ON THE INTERNET

MACIEJ GAWINECKI ¶, MINOR GORDON †‡, PAWEŁ KACZMAREK ¶, AND MARCIN PAPRZYCKI ‡§

Abstract.

In order for software agent technology to come to full fruition, it must be integrated in a realistic way with existing production technologies. In this paper we address one of the interesting problems of *real-world* agent integration: the interaction between agents and non-agents. The proposed solution is designed to provide non-agents (client software in particular) access to agent services, without restricting the capabilities of agents providing them.

1. Introduction. The future success of online content providers will depend on their ability to filter the mass of information available on the Web into the form that is truly useful to the individual user [26, 23, 44, 25, 2, 64]. Before filtering services can be automated and scaled, however, the World Wide Web must first be transformed to a machine-interpretable content base. One of the attempts at achieving this goal is the Semantic Web [53, 54, 55], which promises to allow computing processes to filter, separate and synthesize elements of information [14]. The major thrusts of the Semantic Web effort are the development of languages to describe content ([48, 13, 42, 62, 63]) and of software capable of interpreting this content.

Software agents are one of the key components in this latter category [54, 55]. Research in this area has long emphasized the potential of agents for intelligently interpreting semantic content, personalizing this content for the user, and acting autonomously and collaboratively to deliver it. Agents representing users are in a position to intelligently filter information from the Web in order to satisfy a user's request by consulting dynamic and static sources of semantically-described content as well as other intelligent agents. A large body of literature has developed around this role of software agents (see for example [27] and references quoted there).

As a *framework* for conceptualizing intelligent computing processes such as those required for working with the Semantic Web, software agents are close to ideal. As a *specific technology* for working with the Semantic Web data, however, agents are far from ready to assume the roles ascribed to them. Though the theoretical concepts that define the software agent paradigm have been thoroughly researched (and remain a focus of current research efforts: see [34, 15] as well as recent proceedings from multiple agent-focused conferences), technological advances to support agent-based models have usually not accompanied this research [37]. Many proponents and even standards bodies (in particular FIPA: [22]) have described detailed scenarios in which agents play critical roles, yet very few of these designs have progressed beyond the initial stage of development. Although the absence of supporting technology has certainly been one of the reasons for this gap between agent theory and practice, we believe that the lack of realistic implementations of flexible agent-based applications has also resulted in part from the attitudes of agent researchers. Most designers have either been too "realistic" and implemented systems that can hardly be called agent-based, or have made so many assumptions about the *future* of agent technology (i.e. practically every computer connected to the Internet is capable of receiving agents [22]) that their designs will almost certainly be outmoded before that state of technology exists and are impossible to implement beyond some initial working model employing *currently* existing software. Serious efforts toward finding an intermediate solution have only recently gained strength, and many of the conceptual problems pointed out by critics such as Nwana and Ndumu [37] have begun to be addressed. In addition to the basic problems of communications management and ontology agreement, Nwana and Ndumu cited the development of realistic agent-based systems as both the problem and the solution to the evolution of agent technology. This paper represents our attempt to address that problem.

With this in mind, we must first recognize that developers seeking to apply software agent technology to a specific application currently have two choices: they can either wait for the ideal technology to come along and facilitate the implementation process, or make do with existing agent

¶Department of Mathematics and Computer Science, Adam Mickiewicz University, Poznan, Poland

†Computer Science Department, Technical University of Berlin, Berlin, Germany

‡Computer Science Department, Oklahoma State University, Tulsa, OK 74106, USA

§Computer Science, SWPS, Warszawa, Poland

technology, knowing that they will have to work around it as much as work with it. While the former approach certainly has some currency for those whose goal is to develop industrial-strength applications and therefore prefer to use well-established technology, we focus on the latter in our research. By choosing this approach we acknowledge that any application we develop with existing, rudimentary technology will almost inevitably be outmoded in the future by more elegant solutions. Our hope is that in the process of developing such an application we may help shape the “ideals” of software agent technology itself, so that future agent software developers will find it easier to work in realistic contexts.

The application we are in the process of developing is an agent-based system supporting the needs of travelers. In this paper we address one of the interesting problems that has materialized during the implementation phase: the problem of software agents communicating with non-agents, the latter being the clients of the system. The remaining parts of the paper are organized as follows. In the next section we briefly describe the system under development and follow with a discussion of the role and requirements for agents communicating with a heterogeneous, functionally-limited client base. In the subsequent section we analyze in detail possible approaches to facilitating agent-client interactions. In the next two sections we introduce and discuss the proposed solution.

2. Internet-based Travel Support Systems—General Considerations. When considering existing travel support systems such as Expedia, Orbitz, Travelocity, et al., one tends to focus on the content aspects of the system—the travel services and information these sites provide—and not the technology that makes them work. The technology is (and should be) transparent. Behind the scenes these travel support systems are enormously complex, and adding further levels of intelligence / user friendliness will only make them more complicated (for more details of current trends in development of intelligent travel support systems see [28] and references enclosed there). From this perspective software agents represent an excellent means of breaking apart complex systems into components (agents), each with its own “knowledge” and “goals”, thus allowing developers to better address the complexity of the system [32]. In addition, travel support (browsing, booking, etc.) is an ideal candidate for agent-based implementation because the services offered by Internet-based travel support systems are those we associate with another kind of an agent—the human travel agent. An agent or agents can be designed to directly adapt the role and functions of the human travel agent by following our conventional intuition of an “agent” as someone/something who represents us, usually in a particular domain such as travel (real estate, insurance, etc.). In this sense, a software agent-driven travel support system can work like a real “travel agency”, with individual “travel agents” in the system serving individual users, just as a human travel agent would serve an individual in person. We would also like to incorporate other perspectives on software agents, namely, the conception of agents as autonomous, intelligent entities, which has arisen from the field of artificial intelligence. It is our belief that in order to imitate the role of a human travel agent with our agent-based travel support system, we must start with a software agent assuming the travel agent role and design the other agents to support this “travel agent”. In fact, we call this central agent the “personal agent,” because it represents the travel support system to the individual user; there is one personal agent running for every user accessing the system. The other agents who support the personal agent have been developed as functional components of a distributed, complex system. These secondary agents support the personal agent in fulfilling the user’s requests: researching available options, developing travel plans and choices to fit the customer’s preferences, booking, etc. The personal agent is responsible for interacting with the user and executing various aspects of her request by directing other agents in the system.

2.1. System Under Development—High Level Overview. Our work on development of an agent-based travel support system was initiated in 2001 and first described in [23, 43]. This work led to the conceptualization of a complete framework presented in [2]. The processes involved in personalization of information delivery (independent of the implementation) were discussed in [25]. Knowledge management in its conceptual and agent-related implementation aspects was considered in [26, 29, 23, 64]. Our system has been designed to imitate the role of a human travel agent in serving individual travelers: presenting a wide selection of travel choices, planning itineraries and representing the customer to airlines, hotels and vehicle rental agencies as well as other, more

information-oriented services such as displaying the hours of operation of museums and historical sites or the locations of nearby restaurants. In order to provide these services the system accumulates semantically-demarcated travel-related information collected from the Internet, which is cross-referenced according to multiple classification schemes (geospatial, ontological, method of access, etc.).

Customers connect to our system using Internet-enabled devices—web browsers, PDAs, WAP phones and others—and request travel-related information. The initial response to a request is prepared from the information stored in the system and filtered to match the user’s personal preferences. The system also attempts to sell additional services to the user through targeted advertisements [25]. In the process of interacting with the system, customer’s queries may be refined as more specific choices are presented. All of the details of customer-system interactions are stored in a user behavior database for later data mining [2, 24]. The complete system framework is specified in terms of software agents, with separate agents performing all of the above-described functions [32].

One of our most important design criteria was to make our travel support system accessible not only to the general Internet audience (typically accessing services through a desktop-based web browser) but also to travelers accessing services through mobile devices such as laptops, PDAs and cell phones. Each of these technologies has a certain set of capabilities and limitations that must be considered in providing access to the different devices. This is the fundamental problem for software agents interacting with non-agents (in this case, client devices).

Because travel support is such an ideal scenario for applying agent technology, there have been a number of agent-related projects in this domain; unfortunately, most of these projects have either been very limited in scope [36, 57, 58] or have never left the initial development stages (a partial list of the projects that have never been completed can be found in [43], while others can be easily located through Internet searching). Unlike most of these projects, our essential goal is not to prove the elegance of design with software agents, but to demonstrate that agent technology, as it exists *today*, can actually be used in a real system. This is demonstrated by the fact that the problem of agent-non-agent interaction only arises when agent systems are actually implemented, and thus it has been largely ignored or downplayed by other researchers [22, 37].

3. Clients of the Travel Support System. In order to develop device-independent support for customer-system communication we have to start from the client side, as it is the diversity of possible clients that is the source of the agent-client interaction problem. Despite the diversity of device types the system must support, we can still assume that all interaction between devices (clients) and the system on the Web will be conducted via the HTTP protocol. Only the *content* that is delivered over HTTP that will differ from client to client. We can divide clients into two broad classes, based on the type of content they expect:

- *Non-interactive clients*, which include but are not limited to web service clients (using SOAP [56]), geospatial clients [39, 40], travel industry systems [41] and external agents [1].
- *Interactive clients*, which include among others conventional web browsers (interpreting HTML) and browsers on personal devices (interpreting HTML or WML) as well as interactive applications that utilize the services of our system.

In this paper we will focus on browsing clients in the latter class. Though the class of interactive clients described above also includes applications (Flash-based, Java applet-based, etc.), these are not the primary client base of the system, and furthermore, we believe that users should not have to (and may not be allowed to) download any special software in order to interact with our system (more on this below). More precisely, if a user can and is willing to load a special applet that will allow her computer to accept mobile agents, then the problems that are of interest to us can be solved in a different way, or cease to exist completely. Henceforth we will refer to browser-like clients (desktop and mobile) as simply “clients”. However, the solution proposed here can be naturally and easily extended to include, among others, scenarios involving software agents entering user devices. Let us now consider the basic limitations define the possible solution space and the existing capabilities that we can exploit for our system.

3.1. Client Limitations and Capabilities. Although the HTTP protocol itself is relatively standard across browsing clients, browsers are often limited in their ability to render content sent

from a server via HTTP (whether in HTML or WML, and in particular XML). Therefore we cannot impose many requirements on browsing capabilities when considering an audience as large as travelers on the Internet. This automatically precludes platform-specific, device-specific, browser-specific or otherwise specialized technologies on the client side such as Flash movies or Java applets, which are still largely inaccessible to non-desktop-based browsers. Some users may also be unable (personally) or not allowed (administratively) to modify settings or install programs on their computers, while others may reject anything they consider “foreign”. Furthermore, despite the widespread distribution of XML parsing technology in web browsers and runtime environments such as Java, we still cannot consistently rely on the end user’s software to interpret or transform content beyond the standard markup languages such as HTML or WML. We must consider the client as “dumb” as possible, so we don’t exclude clients that actually are so restricted.

On the other hand, for a web-based travel site to offer useful content and services it must still require a minimum subset of client-side interpretative capabilities for interactive access, client limitations notwithstanding:

- Client devices / software must be able to display some flavor of *markup* (HTML, WML, XHTML, etc.)
- Clients must be able to *connect one-way* to services over the Internet using HTTP.

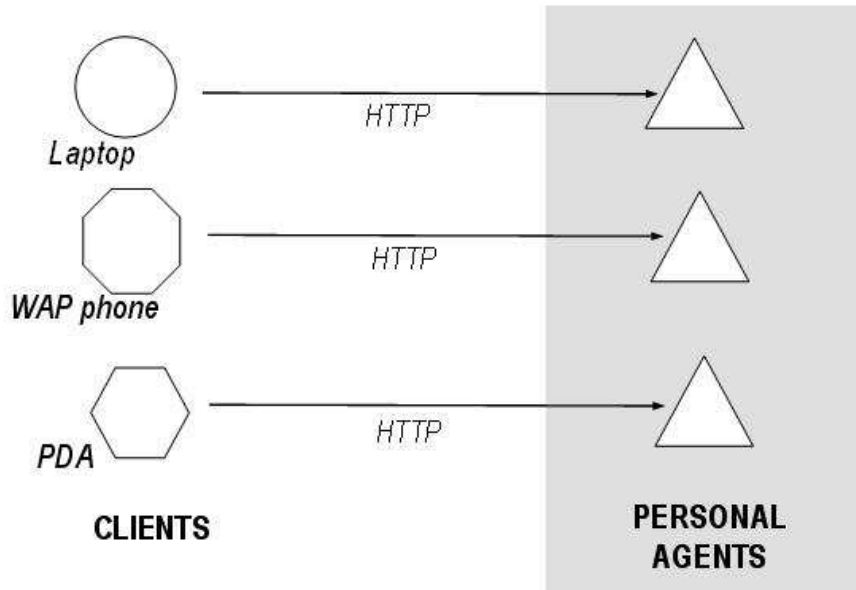
It has to be stressed that the above-described capabilities represent the required minimum. This being the case, the system may still be able to adapt to discovered capabilities of the client: for clients with only minimum interpretative function (e.g. the typical browsers we are focusing on), the server system is responsible for most of the interaction and interface with the user, while more sophisticated client applications may simply call upon the system’s functions directly (as is the case with fully-interactive applications as well as the non-interactive clients specified previously). In this paper we concentrate on providing a *minimum subset* of markup (HTML and WML) and protocol (HTTP) we can assume **all** browsing clients can handle, and leave the extended capabilities of the travel support system for the future.

3.2. Client Agnosticism. One of the strengths of the web’s client-server architecture is the agnosticism of clients with regard to server technology. As long as a web server provides interpretable content (markup, images, etc.) over HTTP, the web browser need never be aware of how this content is produced. The same web page may be derived from static files, querying a database or sending a request message to an agent across the network. The client only sees the end result—a web page. We will take advantage of this client agnosticism by designing a server system that *appears* to work like a normal web server but handles requests much differently on the server side.

4. Communicating Agents of the System. This far we have constructed the following general picture of our work. We are in the process of developing of an agent based travel support system in which software agents will have to communicate with customers accessing the system using devices that we must assume have only very limited capabilities. Let us now consider two basic but unrealistic possibilities for agent-client communication within this system.

4.1. Agents Accessing the Client Devices. In the last section we arrived at the basic assumption that our agent-based travel support system should be web-based and operate within the conventional web browser—web server infrastructure. Unfortunately, this assumption contradicts one of the central ideals of software agent research: agents should be everywhere. The user should have an agent on her desktop—either sent by the server or already present as the user’s personal assistant—and this agent should be responsible for interacting with the web, in lieu of the conventional web browser. Despite the fact that this vision is still only an ideal, many agent-related software projects assume that an agent is/will be present on the user’s system. The reality is that agent platforms generally require a host environment (“agency”, “platform”, “marketplace”) to support agent execution on the local system. Furthermore, there are only a few projects that are aimed at supporting agents entering mobile devices [33]. The presence of such a specialized environment on a range of client devices is obviously not a realistic assumption.

4.2. Agents Exchanging Messages with Clients. Here we consider a scenario in which a conventional browsing device (desktop- or device-based) establishes an HTTP client-server connec-

FIG. 4.1. *Client-agent link*

tion directly with a software agent, as illustrated in Figure 4.1.

4.2.1. FIPA. After some years of uncertainty and in-fighting between different organizations and companies, the Foundation for Intelligent Physical Agents (FIPA) [15] has emerged as the de facto standard bearer for the agent community. FIPA has defined standards and specifications for inter-agent communication—the Agent Communication Language [16]—as well as guidelines for agent platforms. These standards are gradually being adopted by the agent community at large, and have been the basis for several general-purpose agent platforms [30, 7, 8]. One of the more popular of these—the Java Agent Development Environment (JADE)—has been widely deployed because of its close conformance to FIPA specifications and the resulting endorsement by large-scale projects such as the AgentCities network [1], which have a clear interest in platform interoperability. JADE is our platform of choice for the travel support system.

Unfortunately, FIPA and the developers of FIPA-compliant platforms such as JADE have concentrated almost entirely on the development of inter-agent and inter-platform protocols and languages such as ACL, with only marginal regard for the possibility of non-agents (i.e. clients such as ours) interacting with FIPA-compliant agents. Even the inter-agent communication mechanisms based on web standard protocols and formats (e.g. the HTTP Message Transport Protocol [19] and the XML encoding for ACL [17]) are designed only for transporting ACL messages between agents. Non-agents, even if they can “speak” HTTP, cannot interpret the contents of ACL messages sent from agents without special software. The situation is further complicated by the indirection imposed by the software agent platforms; agents may address each other within the platform via a FIPA-specified addressing schema, but are shielded from the larger network by the platform environment. Software agents within the platform cannot be directly addressed via the standard IP, port scheme, and thus they cannot be reached directly by client devices over HTTP.

4.2.2. Agents as Servers? Without speaking a language clients can interpret (i. e. HTML or WML) and without being able to directly travel to the desktop or to access the personal device, the agent is “cut off” from the user. The ideal of clients connecting directly to agents is also a dead end. Finally, agents (as they exist now) were not designed to be servers. An agent that remained in a known location and only responded to requests would not really be an agent by any of the existing definitions, and enforcing these limitations on agent software (existing or specially-designed) would mitigate many of the reasons for using agent technology in lieu of simple daemons in the first place.

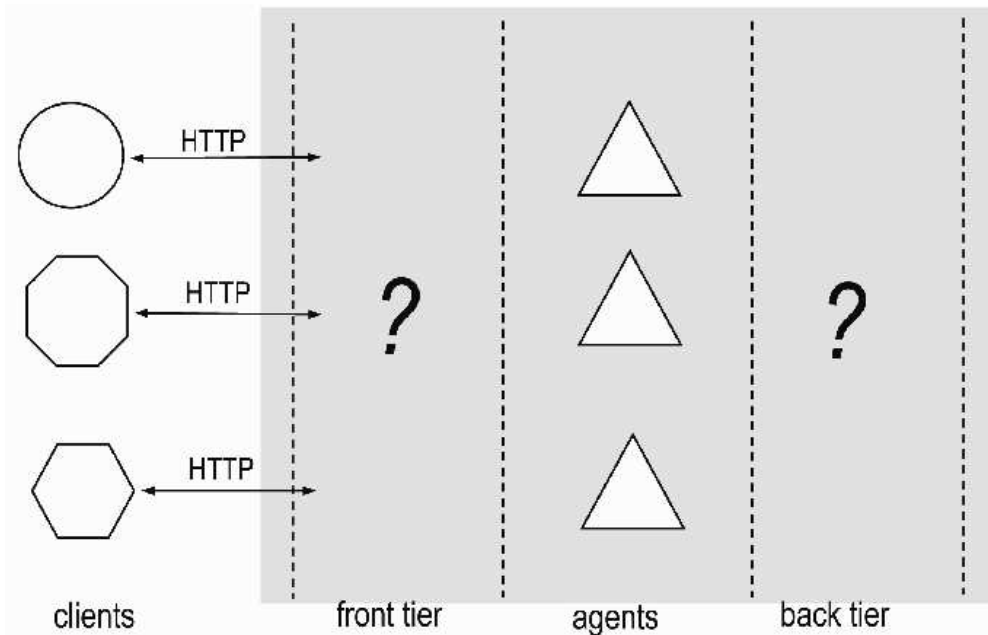


FIG. 5.1. Agents as middleware; clients reside on the Internet; the remaining tiers reside on the server

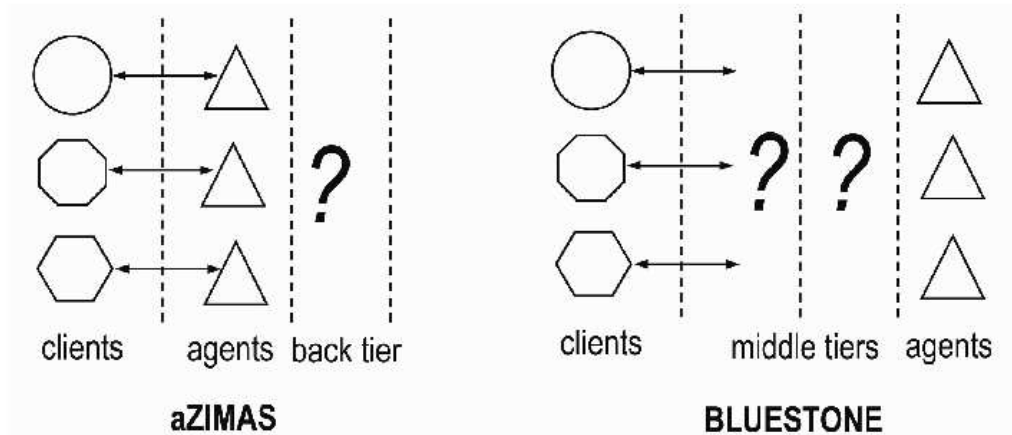
In addition, an agent that internally supported the full HTTP protocol as well as any markups the system is expected to support (HTML, WML, or XML) would be very complex and limited in its mobility and distribution.

In summary, as we have just argued, neither of the basic scenarios (agents entering the device or agents acting as servers), can solve the problem of non-agent-agent communication. Rather, we have to focus on the strengths of agent software, and let other technologies make up for its weaknesses.

5. N-Tier Systems. The conventional n-tier architecture is designed to separate user interfaces from business logic and business logic from the resources it employs, so that each tier is encapsulated and need not be aware of the operations of the others. It is obvious that the n-tier approach applies to our problem. Conceptually, the agent infrastructure becomes the middleware [50], allowing other components of the system to handle the problem of communicating with the user and to interact with back-end resources. A design based on agents acting as middleware is illustrated in Figure 5.1.

Multi-agent systems are well suited for encoding middle-tier business logic. In such a system agents should be allowed to “pretend” that they are actually interacting with and representing the user directly, i.e. on the abstract, conceptual level we want the situation depicted in Figure 4.1 while on the implementation level we will actually have the situation illustrated in Figure 5.1. This situation is actually fortuitous: a personal agent on the server can take advantage of the server environment while still being geared toward serving the individual (personalizing and localizing content derived from the system [2, 25]). There are also additional advantages of agents being located on the server and acting as one of the layers of the n-tier system:

- *Processing power:* agents can either reside on large servers or be spread out on to a distributed system; each has its own advantages and disadvantages, but more importantly, neither relies on the user’s system or bandwidth.
- *Mobility:* an agent could be anywhere on the network, as long as it stays in touch with the server system.
- *Access to back-tier resources:* databases and other information sources can be directly accessed by agents or through other agents in closer proximity to the source.
- *Uniformity:* all agents can be exactly the same as they have been separated from the

FIG. 5.2. *aZIMAS vs. Bluestone*

multiplicity of “languages” spoken by clients.

We would like to construct an n-tier architecture that supports the transparent integration of a general agent platform such as JADE into an otherwise conventional enterprise application, which can then take advantage of the strengths of agents (both distributive and intelligence aspects) while retaining the interoperability of existing technologies such as web servers. This relieves the agents of the problems of communicating with non-agents while still allowing them to be the real intelligence of the application—in the middle. Fortunately, the n-tier approach offers a great deal of flexibility in dividing the functions of the different tiers. Some developers choose to make the presentation layer as “dumb” as possible, even going so far as to have the middle layer output almost-presentable content for delivery to the user. Others focus on the end user functions, and make the middleware only a thin layer above the back-end resources on the network (both approaches are illustrated in Figure 5.2.) Of course these distinctions and the n-tier approach itself largely excludes pure agent-based designs, which are founded on the assumption that there is no need for intermediaries because agents will be everywhere.

5.1. Agents on the Front End. One way to integrate agents into an n-tier server architecture and allow clients to access them over the Internet is to actually embed agents within a web server, tightly binding the front (presentation) and middle (logic) tiers of the system. This approach was taken by the aZIMAS system, which uses web server modules (for Apache or IIS) to route messages with predefined HTTP parameters to agents running in the web server’s process space [3]. This is very efficient approach to the agent-client communication problem, as it integrates agents tightly into an existing server structure with a minimum of customization. Unfortunately, embedding agents in the server required the aZIMAS developers to create an entirely new agent platform to work with the system. From our perspective this is not a true solution to the problem, as it bypasses the problem of linking current agent platforms (such as those that comply with the FIPA standard).

5.2. Agents on the Back End. The opposite approach places agents on the back end of an n-tier system, treating them as service providers, in much the same way that legacy applications are exposed as web services—by representing their functions through a standard interface, and brokering requests for and responses from the functions through a middle tier. This was the approach taken by Hewlett-Packard’s Bluestone middleware system [5, 9], which allows HTTP, SMTP and other client types to communicate (bidirectionally) with agents through an intermediary (the Universal Listening Framework) [6]. A service broker (the Universal Business Service) negotiates service requests from clients with agents and other service providers. After the merger of HP and Compaq, the Bluestone project was cancelled and agent-related development in the context of application services became the BlueJADE project, which similarly attempted to integrate JADE agents into an application

server (JBoss) [4]. One disadvantage of using agents as service providers is that it reduces them to mere request handlers, operating through a generic interface. This limits much of the usefulness of agents, as it ties them too closely into the architecture.

6. Agents as Middleware. For the implementation of our travel support system, we employ a general agent platform (JADE) for serving clients over the Internet, attempting to incorporate the best aspects of both of the extreme approaches described above, taking advantage of agents' strengths, and minimizing exposure of their weaknesses.

In designing our system, we started from the vision of clients connecting directly to personal agents (one client per agent) depicted in Figure 4.1 with the goal of making the layer or layers between clients and agents as transparent as possible—such that conceptually we end up with Figure 5.1 while actually implementing Figure 5.2. To this end we insert a limited number of intermediaries between the client and the personal agent (the HTTP server and the *Proxy agent* in Figure 6.1). These intermediaries pass requests and responses between the user's client software (linked to the system over the Internet) and the personal that is responsible for serving that user, transforming the communications to appropriate formats as necessary. Let us now describe in more detail the processes involved in passing user query from the device to the database and the results back to the user device.

6.1. Proposed Solution. An intermediary is required to communicate with clients of various types, in order to shield/abstract the personal agent and the rest of the system from low-level client protocols (e.g. HTTP, WAP, etc.). In Figure 6.1 we represent only one *Proxy agent* that is to handle HTML-based requests and responses, while each additional protocol would be handled by a separate *Proxy agent*. We assume that requests from the system are from a specific device arrive in the form of HTTP POSTs or GETs, which are usually initiated by filling in predefined forms (see the next Section and Figures 6.2 and 6.3) and pushing a “submit” button. Within the HTTP server a *Proxy agent* receives HTTP requests from all devices utilized by users of the system, transforms them (see below) and forwards as ACL messages to the specific user's personal agent. The type of client device (which indicates the necessary transformations) can be detected implicitly (e.g. from request headers) or explicitly (from URL tokens) and stored in server state.

One should remember that HTTP is a synchronous protocol and agent-based communication is asynchronous. Thus each response from an agent-based environment needs to be matched with an appropriate HTTP request. The proxy agent must keep the HTTP connection alive until final response from the personal agent is returned. Detailed mechanics of servicing a given user-request is as follows:

1. proxy agent receives a request over HTTP,
2. a new thread is started to process the request; the request is given the unique identification *ID*; this ID includes the I/O device type information,
3. the thread forwards the request to the personal agent via an ACL message; from this point all messages relevant to this request must be accompanied with that *ID*,
4. the thread suspends,
5. the personal agent processes the request and sends an ACL response through a transformation infrastructure (e.g. to transform the response into HTML, WML, etc.)
6. the proxy agent notifies the suspended thread that the response is ready, and
7. the resumed thread sends back the response (as an HTTP body) to the user device.

When the proxy agent receives the user's query request (HTTP POST or GET), it extracts the content of the query from the CGI query string, transforms the query into an ACL message, and sends this message to the personal agent. For example, the contents of the message might be:

```
page?formvar1=val1&formvar2=val2
```

(see the next Section for a detailed example). The personal agent receives this message and forwards it to the database (*model*) agent as well as the *interaction logging* agent (omitted in Figure 6.1). The role of this latter agent is to store, for further processing and mining, information about all interactions between users and the system [2, 25]. Further discussion of this part of the system are outside of the scope of this paper. The database agent extracts the query from the ACL message

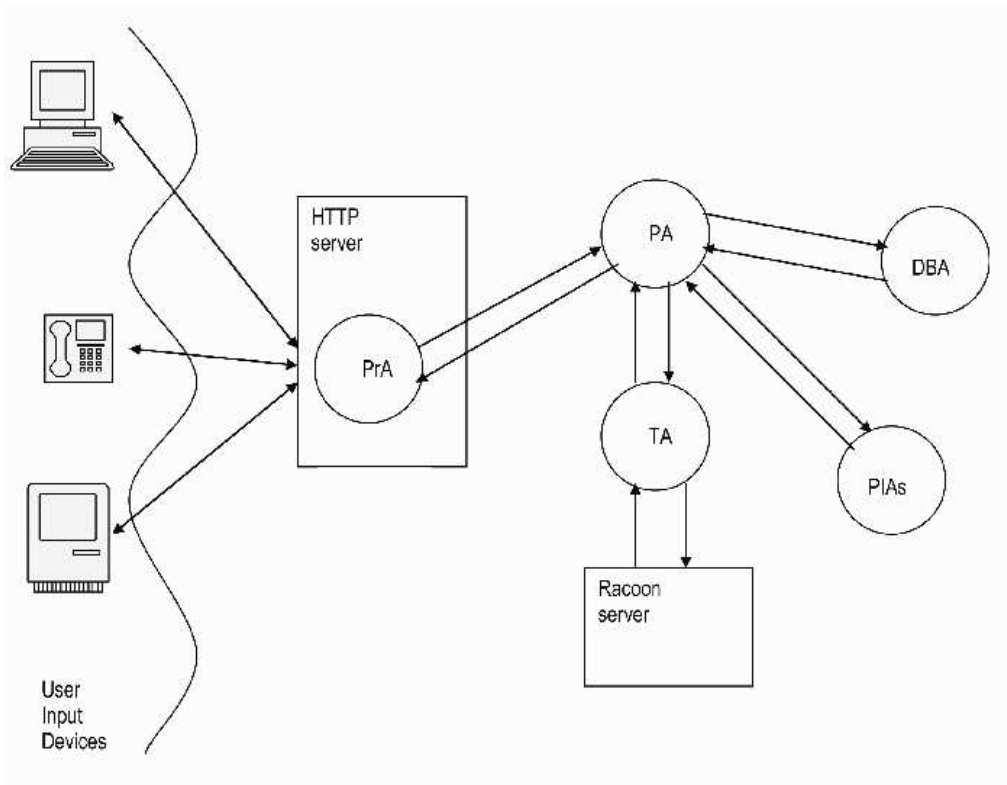


FIG. 6.1. Client side of the system, information flow details.

and transforms it into the appropriate database query language. In our system this database returns a set of RDF [48] triples that define travel objects, which are defined with formal ontologies. The set of RDF triples are serialized and packed into an ACL message and forwarded by the database agent to the personal agent. The personal agent may elect to further expand/reduce the result set by directing a set of personalization agents to add/remove triples before the set is “complete”. The user-ready response set is then returned to the personal agent for the task of transforming the RDF triples into a form that can be displayed on a user device. The personal agent delegates this operation to a *transformation agent*, which in our system is based on the Racoon server [47]. Racoon uses the RxPath language (analogous to XPath 1.0 in terms of syntax and behavior) and RxSLT stylesheets (analogous to XSLT) to transform RDF-demarcated data into displayable forms such as HTML/XML, in a manner similar to that of Apache Cocoon [12]. We have defined a separate set of RxSLT stylesheets for each type of input device we wish to support. After applying the appropriate transformations the transformation agents sends the resulting HTML/WML back to the personal agent, which forwards the displayable content back to the proxy agent for transmission to the user.

6.2. Example scenario. Here a simple example scenario should serve to illustrate the process depicted above. We assume that a user has successfully logged in to the system and established interaction with a personal agent. For this scenario the data consists of sets of RDF triples with elements from a restaurant ontology [11]:

```

:Poland_ZP_Swinoujscie_Albatros_Klub_Nocny1051910264
  a res:Restaurant ;
  res:title "Albatros , Klub Nocny" ;
  loc:streetAddress "ul. Zeromskiego 1" ;
  loc:city "Swinoujscie" ;
  loc:country "Poland" ;
  
```

Property Name	Comment	Input
URL	A restaurant's main web page.	<input type="text"/>
accepts	Payment method accepted by this restaurant. Expect several of these for each restaurant. Comment: All restaurants accept cash, so we don't list it.	-select-
accessibility	String describing how handicapped-accessible the restaurant is.	-select-
accessibility Notes	Details on a restaurant's handicapped accessibility.	<input type="text"/>
alcohol	A string describing the alcohol service.	full bar
breakfast Price	Breakfast Price.	-select-
capacity	Maximum number of people the restaurant can hold.	<input type="text"/>
city		Swinoujscie
clientele	The type of people who usually frequent this restaurant.	<input type="text"/>
country		Poland
cross street	The nearest street that crosses the street that the restaurant is on.	<input type="text"/>
cuisine	The type of food a restaurant serves. We repeat this field up to three times.	Bar / Pub / Brewery
delivery phone	A restaurant's delivery phone number. Defined only if there is a separate phone number for delivery. Same format as Phone.	<input type="text"/>
delivery URL	An URL where the user may order food from this restaurant online.	<input type="text"/>
price	The cost of an average dinner at this restaurant, including entree, non-alcoholic drink, and half an appetizer or dessert. If the restaurant does not serve dinner, we use the closest meal.	-select-
dress	A string describing acceptable dress for the restaurant.	-select-
fax	A restaurant's main fax number. Same format as Phone.	<input type="text"/>
feature	A string describing a feature of the restaurant. One of these tags exists for each of the restaurant's features.	-select-
hours	A string describing the hours the restaurant is open.	<input type="text"/>

FIG. 6.2. *Input form for the system*

```

loc:phone "+48_(91)_321_18_66" ;
loc:state "ZP" ;
loc:zip "72-600" ;
res:accepts mon:AmericanExpressCard ,
            mon:DinersClubCard ,
            mon:JCBCard ,
            mon:MasterCardEuroCard ,
            mon:VisaCard ,
            mon:DebitCard ;

```

```

res:alcohol res:FullBar ;
res:cuisine res:BarPubBreweryCuisine ;
res:hours "24h" ;
res:locationPath "Poland/ZP/Swinoujscie" ;
res:parsedHours "0-24|0-24|0-24|0-24|0-24|0-24|0-24" .

```

In order to query the restaurants in the database the user fills in a form defining criteria of a restaurant search and clicks the <query> button. An example of the form used in our system is shown in Figure 6.2.

Here the user is looking for a pub in the city of Swinoujscie. The submitted CGI query string for this request is as follows

```

http://www.agentlab.net/restuarant/page?action=getdata&alcohol=
FullBar&cuisine=BarPubBreweryCuisine&city=Swinoujscie

```

The proxy agent that receives the HTTP requests maps the variables of the CGI query string to the temporary form `[[alcohol{{FullBar[[cuisine{{BarPubBrewery[[city{{Swinoujscie)`, and packs this string into an ACL message, which is sent to the database agent via the personal agent (as described above). The database agent extracts the variables and and composes them into an RDQL query [49]:

```

SELECT
    ?r
WHERE
    (?r, <res:cuisine>, <res:BarPubBreweryCuisine>),
    (?r, <res:alcohol>, <res:FullBar>)
    (?r, <loc:city>, "Swinoujscie")
USES
    res for <http://www.agentlab.net/schemas/restaurant#>,
    loc for <http://www.agentlab.net/schemas/location#>,
    alcohol for <http://www.agentlab.net/schemas/alcohol#>

```

The query is executed against our RDF database and matching RDF triples are serialized to create an RDF/XML document:

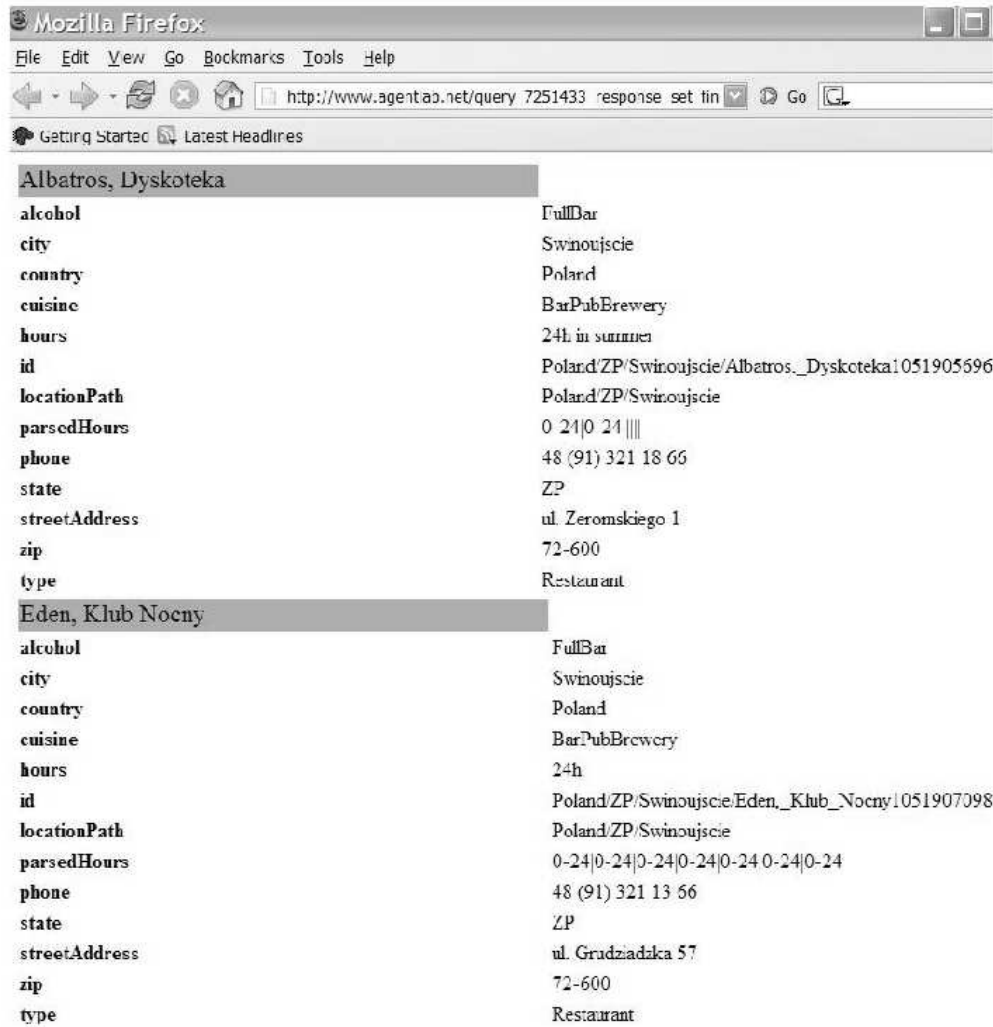
```

:Poland_ZP_Swinoujscie_Albatros_._Dyskoteka1051905696
    a res:Restaurant ;
    loc:streetAddress "ul. _Zeromskiego_1" ;
    res:alcohol res:FullBar ;
    loc:city "Swinoujscie" ;
    loc:country "Poland" ;
    res:cuisine res:BarPubBreweryCuisine ;
    res:hours "24h_in_summer" ;
    res:locationPath "Poland/ZP/Swinoujscie" ;
    res:parsedHours "0-24|0-24||||" ;
    loc:phone "+48_(91)_321_18_66" ;
    loc:state "ZP" ;
    loc:zip "72-600" ;
    res:title "Albatros, _Dyskoteka" .

```

The process then proceeds as outlined above. The user sees the HTML-rendered query results as a normal web page (v. Figure 6.3).

The entire chain of interactions between software agents on the server side remains transparent to the user. The problem of agent-client interaction has been successfully addressed without resorting to a proprietary agent platform or unrealistic assumptions. Furthermore, the extensive features of the general-purpose JADE agent platform allow us to exploit other properties of agent systems, such as the ability to distribute and migrate agents (specifically, the database, transformation, and personal agents) between multiple agent machines. From a conceptual perspective, the use of software agents has resulted in a highly modular and encapsulated design, with agents for each of the tasks in the



The screenshot shows a Mozilla Firefox browser window with the address bar displaying 'http://www.agentab.net/query 7251433 response set fin'. The page content is a search results page with two entries:

Albatros, Dyskoteka	
alcohol	FullBar
city	Swinoujscie
country	Poland
cuisine	BarPubBrewery
hours	24h in summer
id	Poland/ZP/Swinoujscie/Albatros_Dyskoteka:051905696
locationPath	Poland/ZP/Swinoujscie
parsedHours	0 24 0 24
phone	48 (91) 321 18 66
state	ZP
streetAddress	ul. Zeromskiego 1
zip	72-600
type	Restaurant
Eden, Klub Nocny	
alcohol	FullBar
city	Swinoujscie
country	Poland
cuisine	BarPubBrewery
hours	24h
id	Poland/ZP/Swinoujscie/Eden_Klub_Nocny:051907098
locationPath	Poland/ZP/Swinoujscie
parsedHours	0-24 0-24 0-24 0-24 0-24 0-24 0-24
phone	48 (91) 321 13 66
state	ZP
streetAddress	ul. Grudziadzka 57
zip	72-600
type	Restaurant

FIG. 6.3. Results page

system. Furthermore, we believe that our use of the personal agent is a particularly intuitive means of representing and coordinating per-user interaction.

7. Concluding remarks. In this paper we have presented a solution to the client-agent communication problem, one of the key challenges in developing realistic agent systems that interact with user devices on the Internet. The proposed solution is based on inserting proxy agents into an HTTP server, which act as a gateway between the outside world and the agent system. Thus far we have completed the implementation for the HTML-based browser used as an I/O device in the system. In the next step we will add another class of proxy agents to process WML-based interactions as well as agent to serve Java-enabled mobile devices. We will report on our progress in subsequent papers.

REFERENCES

- [1] AGENTCITIES NETWORK SERVICES, <http://www.agentcities.net>
- [2] R. ANGRYK, V. GALANT, M. PAPRZYCKI, M. GORDON, Travel Support System - an Agent-Based Framework, *Proceedings of the International Conference on Internet Computing IC'2002*, CSREA Press, Las Vegas, NV, 2002, pp. 719-725

- [3] S. ARUMUGAM, A. HELAL, A. NALLA, *aZIMAS: Web Mobile Agent System*, http://www.harris.cise.ufl.edu/projects/publications/ma02_final.pdf
- [4] BLUEJADE PROJECT, <http://sourceforge.net/projects/bluejade>
- [5] PROJECT BLUESTONE; SUMMARY, http://www.bluestone.com/downloads/pdf/06-21-01_Total-e-Server_white_paper.pdf, 2001
- [6] PROJECT BLUESTONE; UNIVERSAL LISTENING FRAMEWORK, http://www.hpmiddleware.com/downloads/pdf/02-27-01_ULFWhitePaper.pdf, 2001
- [7] P. BUCKLE, FIPA and FIPA-OS Overview, Invited talk, *Joint Holonic Manufacturing Systems and FIPA Workshop*, London, September, 2000,
- [8] FIPA-OS, <http://fipa-os.sourceforge.net>
- [9] B. BURG, em Agents in the World of Active Web-services, <http://www.hpl.hp.com/org/stl/maas/docs/HPL-2001-295.pdf>, 2001
- [10] M. BUTLER, F. GIANETTI, R. GIMSON, T. WILEY, Device Independence and the Web, *IEEE Internet Computing*, September/October, 2002, pp. 81-86
- [11] CHEFMoz, <http://chefmoz.org/>
- [12] COCOON PROJECT, <http://cocoon.apache.org/>
- [13] DAML ONTOLOGY LANGUAGE, <http://www.daml.org>
- [14] D. FENSEL, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag; Berlin, 2001
- [15] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, <http://www.fipa.org>
- [16] FIPA, *FIPA ACL Message Structure Specification*, <http://www.fipa.org/specs/fipa00061>, 2001
- [17] FIPA, *FIPA Agent Message Transport Envelope Representation in XML Specification*, <http://www.fipa.org/specs/fipa00085>
- [18] FIPA, *FIPA Agent Message Transport Service Specification*, <http://www.fipa.org/specs/fipa00067>, 2001
- [19] FIPA, *FIPA Agent Message Transport Protocol for HTTP Specification*, <http://www.fipa.org/specs/fipa00084>, 2001
- [20] FIPA, *FIPA Agent Message Transport Protocol for IIOP Specification*, <http://www.fipa.org/specs/fipa00075>, 2000
- [21] FIPA, *FIPA Agent Message Transport Protocol for WAP Specification*, <http://www.fipa.org/specs/fipa00076>, 2000
- [22] FIPA, *FIPA Personal Travel Assistance Specification*, <http://www.fipa.org/specs/fipa00080/XC00080B.html>, 2001
- [23] V. GALANT, M. GORDON, M. PAPRZYCKI, Knowledge Management in an Internet Travel Support System, *Proceedings of ECON2002*, ACTEN Press, Wejcherowo, 2002, pp. 97-104
- [24] V. GALANT, J. JAKUBCZYC, M. PAPRZYCKI, Infrastructure for E-Commerce, *Proceedings of the 10th Conference on Extraction of Knowledge from Databases*, Wroclaw University of Economics Press, Wroclaw, Poland, 2002, pp. 32-47
- [25] V. GALANT, M. PAPRZYCKI, Information Personalization in an Internet Based Travel Support System, *Proceedings of the BIS'2002 Conference*, Poznan University of Economics Press, Poznan, Poland, 2002, pp. 191-202
- [26] M. GORDON, J. JAKUBCZYC, V. GALANT, M. PAPRZYCKI, Knowledge Management in an E-commerce System, *Proceedings of the Fifth International Conference on Electronic Commerce Research*, Montreal, Canada, October, 2002, CD, 15 pages
- [27] IEEE INTELLIGENT SYSTEMS JOURNAL, Special Issue, Vol. 16, No. 2, 2001, <http://www.computer.org/intelligent/ex2001/x2toc.htm>
- [28] IEEE INTELLIGENT SYSTEMS JOURNAL, Special Issue on Intelligent Systems for Tourism, Vol. 17, 2002, pp. 53-66
- [29] M. GORDON, M. PAPRZYCKI, A. GILBERT, Knowledge Representation in the Agent-Based Travel Support System, *Advances in Information Systems*, Springer-Verlag, Berlin, 2002, pp. 232-241
- [30] JAVA AGENT DEVELOPMENT ENVIRONMENT(JADE), Telecom Lab Italia, <http://jade.cselt.it> and <http://jade.cselt.it/papers.htm>
- [31] JENA, <http://www.hpl.hp.com/semweb>
- [32] N.R. JENNINGS, An Agent-based Approach for Building Complex Software Systems, *CACM*, Vol. 44, No. 4, 2001, pp. 35-41
- [33] LEAP PROJECT, <http://leap.crm-paris.com/>
- [34] P. MAES, Agents That Reduce Work and Information Overload, *CACM*, Vol. 37, No. 7, 1994, pp. 31-40
- [35] MOZILLA, <http://www.mozilla.org>
- [36] D. NDUMU, J. COLLINS, H. NWANA, Towards Desktop Personal Travel Agents, *BT Technological Journal*, Vol. 16 No. 3, 1998, pp. 69-78
- [37] H. NWANA, D. NDUMU, A Perspective on Software Agents Research, *The Knowledge Engineering Review*, Vol. 14, No. 2, 1999, pp. 1-18
- [38] OASIS/EBXML REGISTRY SERVICES SPECIFICATION v2.0, <http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf>
- [39] OPEN GIS CONSORTIUM, INC., <http://www.opengis.org>
- [40] OpenGIS Catalog Services Implementation Specification, <http://www.opengis.org/techno/specs/02-087r3.pdf>
- [41] OPENTRAVEL ALLIANCE, <http://www.opentravel.org>
- [42] OWL ONTOLOGY LANGUAGE, <http://www.w3.org/TR/owl-ref/>
- [43] M. PAPRZYCKI, R. ANGRYK, K. KOLODZIEJ, I. FIEDOROWICZ, M. COBB, D. ALI AND S. RAHIMI, Development of a Travel Support System Based on Intelligent Agent Technology, *Proceedings of the PIONIER 2001*

- Conference*, Technical University of Poznan Press, Poznan, Poland, 2001, pp. 243-255
- [44] M. PAPRZYCKI, C. NISTOR, R. OPREA AND G. PARAKH, The Role of a Psychologist in E-commerce Personalization, *Proceedings of the 3rd European E-COMM-LINE 2002 Conference*, Bucharest, Romania, 2002, pp. 227-231
- [45] M. PAPRZYCKI, P. J. KALCZYNSKI, I. FIEDOROWICZ, W. ABRAMOWICZ, M. COBB, Personalized Traveler Information System, *Proceedings of the 5th International Conference Human-Computer Interaction*, Akwila Press, Gdansk, Poland, 2001, pp. 445-456
- [46] S. POSLAD, H. LAAMANEN, R. MALAKA, A. NICK, P. BUCKLE, A. ZIPF, CRUMPET: Creation of User-friendly Mobile Services Personalised for Tourism, *Proceedings of: 3G 2001 - Second International Conference on 3G Mobile Communication Technologies*, 26-29 March 2001, London, UK.
<http://conferences.iee.org.uk/3G2001/>, 2001
- [47] RACOON, <http://rx4rdf.liminalzone.org/Racoon>
- [48] RDF PRIMER, <http://www.w3.org/TR/rdf-primer/>
- [49] RDQL - A QUERY LANGUAGE FOR RDF,
<http://www.w3.org/Submission/2004/SUBM-RDQL-20040109>
- [50] G. RIMASSA, *Perspectives for Agent Middleware*, <http://citeseer.nj.nec.com/551140.html>
- [51] RXPATH, <http://rx4rdf.liminalzone.org/RxPath>
- [52] RXSLT, <http://rx4rdf.liminalzone.org/RxSLT>
- [53] W3C SEMANTIC WEB, <http://www.w3.org/2001/sw/>
- [54] SEMANTICWEB PROJECT, <http://www.semanticweb.org>
- [55] SEMANTICWEB GENERAL DESCRIPTION,
<http://www.semanticweb.org/about.html#bigpicture>
- [56] SOAP, World Wide Web Consortium, <http://www.w3.org/2002/ws>, 2002
- [57] J.N. SUAREZ, D. O'SULLIVAN, H. BROUCHOUD, P. CROS, Personal Travel Market: Real-Life Application of the FIPA Standards, *Technical Report*, BT, Project AC317, 1999
- [58] J.N. SUAREZ, D. O'SULLIVAN, H. BROUCHOUD, P. CROS, C. MOORE, C. BYRNE, Experiences in the Use of FIPA Agent Technologies for the Development of a Personal Travel Application, *Proceedings of the Fourth International Conference on Autonomous Agents*, Barcelona, Spain, 2000
- [59] W3C, WSDL specification, <http://www.w3.org/TR/wsd1>
- [60] W3C, XUL specification, <http://www.mozilla.org/xpfe/xptoolkit/xulintro.html>
- [61] W3C, XUP specification, http://www.w3.org/TR/xup/#normalop_request_ex
- [62] WEB ONTOLOGY (WEBONT) WORKING GROUP, <http://www.w3.org/2001/sw/WebOnt/>
- [63] WEB ONTOLOGY (WEBONT) WORKING GROUP, <http://www.w3.org/2001/sw/WebOnt>
- [64] J. WRIGHT, M. GORDON, M. PAPRZYCKI, P. HARRINGTON, S. WILLIAMS, Using the ebXML Registry Repository to Manage Information in an Internet, *Proceedings of the BIS 2003 Conference*, Poznan University of Economics Press, Poznan, Poland, 2003, 81-89

Edited by: Dan Grigoras, John Morrison

Received: May 10th, 1997

Accepted: October 21th, 1997