Query Routing Process for Adapted Information Retrieval using Agents

Angela Carrillo-Ramos[2], Jérôme Gensel[1], Marlène Villanova-Oliver[1], Hervé Martin[1], and Miguel Torres-Moreno[2]

[1] LIG Laboratory, STEAMER Team. 681 rue de la Passerelle, 38402 Grenoble, France
[2] Pontificia Universidad Javeriana, Computer Science Department, Bogotá, Colombia
{gensel,villanov,martin}@imag.fr; {angela.carrillo, metorres}@javeriana.edu.co

Abstract. In nomadic environments, it is difficult to provide users with the best-adapted information to his needs and those of his access device. These issues give rise to the need to previously analyze the queries, in order to identify the most appropriated sources that will give answer to the query, then to evaluate and integrate the results. These activities correspond to a traditional *query routing process*. However, these activities do not consider explicitly the adaptation of information. Our approach is based on a query enrichment process, which comes upstream from the routing process and "*augments*" the initial query, adding criteria such as *user preferences* and characteristics of the *context of use* in order to adapt information. In our approach, the *query routing* and *enrichment processes* are achieved by *PUMAS*, a framework based on agents and *Peer to Peer* (*P2P*) approaches, which allows nomadic users to access information sources through different types of devices (eventually mobile), and provides users with adapted information in nomadic environments.

Keywords: Agents, *P2P* systems, Query Routing, Adaptation.

1 Introduction

Nowadays, *nomadic users* access different types of information sources through *Mobile Devices* (*MD,* such as *PDA*, phones, laptops, *etc.*). Additionally, query handling of requested queries by *nomadic users* has become an increasingly complex process because the query's resulting data can be held in different information sources. In this context, to guarantee access to several information sources and to actually share these resources among users, it is necessary to have architectures and technologies with high communicative potential. This potential is one of the main characteristics of *Peer to Peer* (*P2P*) systems.

Regarding query handling, some issues arise: *i)* the number of information sources queried by nomadic users, *ii)* the heterogeneity of the source's structure and access methods. Finally, *iii)* it is difficult to provide the user with the best-adapted information to his needs and those of his access device. These issues give rise to the need to previously analyze the query, in order to identify the most appropriated sources that will answer it (this supposes a previous knowledge of the information sources that is managed by the Information System), then to evaluate and integrate the results. These activities correspond to a traditional *query routing process* [12].

However, these activities do not consider explicitly the adaptation of information. Our approach is an answer to this issue and it is based on a *query enrichment process*, which comes upstream from the *routing process*. This enrichment "*augments*" the initial query, adding criteria which consider *user preferences* and *context of use* of the current session in order to adapt information. In our work, this *context* is composed of: information about user location, *MD* characteristics, the access rights of the user and his activities in the system [5].

This proposal of *query enrichment* is based on the fact that several factors can influence the routing: *i)* Changes of the user's location can produce changes in terms of access and information needs [11]. We believe that this is also sometimes valid when there is a device change; *ii)* User preferences are context-aware [5]. When the query is submitted, all context changes involve potential consequences on the query, and consequently on the selection of information sources. *iii)* The characteristics and technical constraints of the *MD* of a nomadic user can give rise to problems of information display, which are difficult to anticipate.

In order to consider these factors, we propose *PUMAS* (***P**eer **U**biquitous **M**ulti-**A**gent **S**ystem*), an agent based framework aimed to provide a nomadic user with adapted information according to his *preferences* and to the *context of use*. *PUMAS* also offers means for questioning several sources, which correspond to *Information Systems* (*IS*) executed on servers, or simple files stored on other *MD*. *PUMAS* agents perform the enrichment mechanism of the initial query by adding different criteria, which considers *user preferences* and *context of use*. This phase of query enrichment leads to a *routing process*.

This paper is organized as follows: Section 2 defines the *Query Routing process* in *P2P* systems. Section 3 depicts a brief *PUMAS* framework overview. Then, section 4 is dedicated to the presentation of one scenario which describes how a query is enriched in order to adapt user's information; we also present the *query routing process* which is based on the analysis of the query and its redirection to the sources able to answer it. We particularly present the algorithms related to these activities. Finally in section 5 we present the conclusion and future work derived from this research.

2 Query Routing Process in P2P Systems

Xu *et al.* [12] define *Query Routing* (*QR*) as a general problem which is based on two main activities: *i) Query Evaluation* using the most relevant sources, and *ii) Result Integration* of the data generated by those sources. In order to perform these two activities, several issues must be considered: *i) Source Selection* which consists of the analysis of the user's query in order to determine the sources that are able to fulfill the query. In order to solve this problem, the system must know the kind and structure of the information managed by each source; *ii) Query Evaluation* performed by the sources selected in the previous step; *iii) Result Integration*: results must be integrated into a single one that will be returned to the user. In this paper, we focus on the first identified problem (*Source Selection*). In order to handle this question, we have classified the possible solutions into three main categories: *i)* the ones which use a gathering of peers for query answering, *ii)* the ones based on filters and data *matching* among terms of a query and the data of an information source, and *iii)* those based on *Trust and Reputation* strategies.

Among the works which present peers that are able to answer to a query, the work of Brunkhorst *et al.* [2] presents how *Super-peers* form a small subset of peers, where each peer has specific responsibilities and the capability of gathering with others in order to perform tasks such as *QR*, mediation in conflict resolution and teamwork. Kokkinidis *et al.* [7] propose to gather peers that share the same information in order to define plans for query answering in a distributed way. *Super-peers* are responsible for the *QR*, and peers handle the queries. Xu *et al.* [12] present some strategies based on techniques of *clustering* for sources' selection: when a query corresponds to a cluster, it is normal that several *sources* of this cluster can answer in a relevant way to the query. Additionally, when a query corresponds to a significant number of clusters, this source is relevant in order to answer the query.

Koloniari *et al.* [8] define *QR* as a mechanism for determining the location of nodes containing documents, which can fulfill the query. This location is determined based on a mechanism of document and filter *matching*. *Filters* are specialized data structures that gather general information of large collections of documents, and which redirect queries only towards the nodes that contain relevant information.

In order to optimize the *QR process*, Agostini *et al.* [1] propose a strategy of *Trust and Reputation* allowing the selection of peers that are able to answer the query. This strategy is based on the following process: a component named "*seeker*" selects the peers that are able to answer to the query *Q* with the highest probability considering established criteria (*e.g.* the quickest to answer, the most reliable answer, *etc.*). In order to decide, the *seeker* creates and manages a list $<p_1, p_2 \ldots p_k>$ of trusted peers for sending the query. The list is sorted using a decreasing trust level. The *seeker*'s strategy to answer to a query is as follows: first, the *seeker* asks p_1, then p_2, and so on, until it receives relevant answers based on the established criteria. It is important to notice that the list of trusted peers can evolve in time.

In a nomadic environment, information required by a user can evolve in function of the *context of use*. For instance, a user can ask for a list of restaurants and to implicitly obtain those located in the street where he is at the time and which are open when the query is formulated. The techniques of *QR* presented previously do not allow by themselves the management of this kind of context evolution. The following section presents *PUMAS*, a framework which considers these aspects by means of enrichment mechanisms applied to the initial query in order to fulfill it.

3 Overview of the PUMAS Framework

During an information search process in nomadic environments, a user can be confronted with several problems such as: *i)* access problems related to the characteristics of networks; *ii)* lack of adaptation of the resulting information considering user's characteristics and preferences, and technical constraints of the user's *MD*; *iii)* lack of mechanisms for searching distributed information on several sources and devices (servers or *MD*). In order to solve these problems, we propose *PUMAS*, a framework based on agents and a *P2P* approach, which provides nomadic users with information adapted to their characteristics and those of their *MD*. *PUMAS* also provides means of interrogating several sources (*e.g. Information Systems*, *IS*). The *PUMAS* architecture is composed of four *MAS* (for a complete description see [4]): *i)* the *Connection MAS* which provides connection mechanisms for different types of *MD* to different *IS*. *ii)* The *Communication MAS* which assures transparent communication between the users' *MD* and the system, and applies a *display filter* in order to present information in an adapted way, considering the technical constraints of the *MD*. *iii)* The *Information MAS* which receives queries from users, redirects them to the *IS* that is/are able to answer them, applies a *content filter* considering the user profile, and returns the results to the *Communication MAS*. *iv)* The *Adaptation MAS* which communicates with agents of the three other *MAS* in order to exchange information about the user, connection, *MD,* communication characteristics, *etc*. The *Adaptation MAS* is composed of *user agents* (*UA*), the *content filter agent* (*CFA*) and the *display filter agent* (*DFA*).

4 Enrichment and Query Routing Processes in PUMAS

In this section, we present one scenario associated to the submission of a query to *PUMAS,* and its correspondent enrichment and sending. We also present the *query routing process* achieved in *PUMAS.* In our proposal, the interactions among agents are based on *communication acts* [9]. All the files are written in *OWL* (*Ontology Web Language*), following the extensions of *CC/PP* proposed by Indulska *et al.* [6]. These files store information regarding user, *MD*, session and location characteristics.

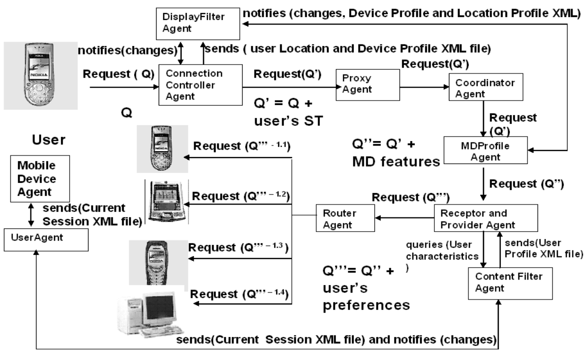


Fig. 1. Enrichment and sending of a query in PUMAS.

When a user sends a query *Q* (see Fig. 1), the *MDAgent* (*MDA*, executing on the user's *MD*) transmits it to the *connection controller agent* (*CCA*). Previously this agent has received new location and device files from the *MDA*, then it sends them to the *display filter agent* (*DFA*). If the user has established as a preference (defined in the *user file*) the fact that his query *Q* depends on his location and the connection time, the *CCA* adds into *Q* information about connection time, user location, *MD* and connection characteristics. This addition creates a new query *Q'* (see Fig. 1), defined by *Q' = Q + Spatial-Temporal characteristics* (*ST*). We illustrate this proposal by means of an example, which describes an enterprise manager who wants to consult his agenda (corresponding to a query *Q*), and he has defined that this activity depends on the location and the connection time. Indeed, when he is at the airport, he wants to know his itinerary flight, hotel address and meetings at the destination city; if he is at his office, he wants to know about meetings that he has within the next few hours. *PUMAS* adds this information into *Q* thus generating the enriched query *Q'*.

The query *Q'* is then sent to a *proxy agent* (which represents the user in the system). Then *Q'* is transmitted to the *coordinator agent*, which transmits it to the *MD profile agent*. This agent also receives from the *DFA*, the *device file*, if the latter was received a new *device file* or notifications about device are changed. The *MD profile agent* adds into *Q'* the *MD* features. These features are provided by the *DFA* which has deduced them from previous queries or has extracted them from its knowledge base. For example, if the enterprise manager is connected through a *Pocket PC 5500* which does not support videos, the answers to his queries can only be displayed as images or text. The *MD profile agent* asks the *DFA* for information regarding the *MD*. The *MD profile agent* can receive from the *DFA* facts defined in the following way (according to the definition given in [3]):

(*deffacts MDCharacteristic* (*MDType "Pocket PC 5500"*)
(*characteristic* (*type"video_not_supported"*) (*description"any network"*))
(*characteristic (type "image"*) (*description "all type of network"*)
(*characteristic (type "text"*) (*description "all type of network"*)
(*characteristic* (*type "all type of network"*) (*description "Wi-Fi, Bluetooth"*)))

The new query *Q"* (see Fig. 1), defined by *Q" = Q' + MD features*, is sent by the *MD profile agent* to a *receptor and provider agent*. The latter adds into *Q",* specific user characteristics. These characteristics are provided by the *content filter agent* (*CFA*) (see Fig. 1). The query *Q'''* is defined by *Q''' = Q" + user's preferences*. A manager can express his *information preferences* using an interface. In this paper, we focus on the *information preferences* which are translated by the *user agent* (*UA*). An example of *information preference* is the following: "*each time that Mr. Joseph Doe asks for his flight itinerary, he also wants to know the hotel information and meetings' schedules at the destination city; if his MD does not support this format, he must receive them as text*":

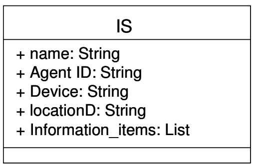
(1)(deffacts Augmented_Information_Preference
(2)(userID "Mr. Joseph Doe")
(3)(required_info "flight itinerary")
(4)(complementary_info "hotel information" "meetings' schedules")
(5)(action
(6)(name "show")
(7)(attribute (name "order")
(8)(items "flight itinerary" "hotel information" "meetings' schedules"))
(9)(attribute (name "graphical_format")
(10)(items "JPEG"))
(11)(problem
(12)(name "JPEG format Not Supported by MD")
(13)(type "incompatibility")
(14)(causes "Only GIF Supported"))
(15)(action
(16)(name "show")

```
(17)(attribute (name "order")
(18)(items "flight itinerary" "meetings' schedules" "hotel information"))
(19)(attribute (name "text_format") (items "XML" "txt")
(20)(attribute (name "image_format") (items "GIF")))
```

Lines 1 to 4 correspond to the fact of an *information preference:* the *required_info* corresponds to *flight itinerary* and the *complementary_info* corresponds to the *hotel information* and to the *meetings' schedules* at the destination city. Lines 5 to 10 are related to the way in which the system must react when displays the information. The system must display information in a specific order using a graphical format. In lines 11 to 14, we illustrate how a specific problem can be described. Lines 15 to 20 are related to the way in which the system must react if this problem occurs. In this case, the system must display information in a given order using text format.

The *MDAgent* sends to the *UA* a new *user file* (see Fig. 1) in case that the user expresses new preferences for the current session. The *UA* communicates the new preferences to the *CFA* in order to generate a single *user profile*, assigning priority to the new preferences. A *receptor and provider agent* transmits *Q'''* to a *router agent.* The latter executes the *QR process* that we describe below.

The activities of the *QR process* in *PUMAS* are inspired in Xu *et al.* [12]. This process is achieved by the *router agents* (*RA*) which receives the enriched queries. When a *RA* receives a query, this agent can send it to a specific or several *ISAgents*, or it can split the query in sub-queries which are sent to one or several *ISAgents*. Fig. 1 shows a scenario in which *Q'''* is split into $Q^{''' - 1.1}$, $Q^{''' - 1.2}$, $Q^{''' - 1.3}$ and $Q^{''' - 1.4}$ which are sent to the *ISAgents* located on different servers or *MD*.




```
(assert (IS
(name TouristInformationOfficeIS)
(agentID TouristInformationOfficeISA)
(device server)
(locationD "destination city")
(information_items "hotel" "tourist places" "rental car" "transport")))
```

Fig. 2. IS Representation: Class and JESS Fact**.**

The ***first activity*** of the *QR process* is the *query analysis*. In this activity, the *RA,* which receives the query, analyzes its complexity. A query is labeled as "*simple*", if it can be processed by only one *ISAgent*, and labeled as "*complex"* if several *ISAgents* are required to process it. In this case, this activity will split the query in sub-queries. This analysis is based on the *facts* related to the *IS*, stored in a knowledge base managed by the *RA*. A *fact* is a knowledge piece which describes a real world element. In order to clarify its definition, we also show its representation as a *UML* class (see Fig. 2). The *RA* also analyzes the adaptation criteria of a query, the list of the addressees of a query, *etc.* After this analysis, the *RA* decides if it must split the query in sub-queries. Let us suppose that an enterprise manager wants to consult his *flight itinerary, hotel information* and *meetings' schedules* at the destination city. We suppose that this query cannot be completely answered by any *IS*. This query is labeled as "*complex"*, that is why it is split by the *RA* in three sub-queries, concerning respectively "*flight itinerary*", "*hotel information*" and "*meetings' schedules*". Such division is based on the stored knowledge obtained by the *RA* regarding the managed information by the different *IS* as a *JESS* fact (see Fig. 2).

In order to split the query, the *RA* identifies the query items [$item_1$, $item_2$... $item_n$]. In our example, $item_1$ corresponds to "*flight itinerary*", $item_2$ to "*hotel information*" and $item_3$ to "*meetings' schedule*". Then, this agent searches for the *IS* which manages an equivalent item. Two items are "*equivalent*" if they are equals semantically (*i.e.*, if both items have the same meaning) or syntactically (*e.g.*, equality in strings of characters). The equivalence relationship is defined by the applications' designers. Additionally, they must implement our *Matching Algorithm*:

```
(1) Initialize ISL answering particularly to the query (ISL)
(2) nIS← 0 ;
(3) i← 1 ; // index of the query items
(4) While i ∈ [1, n] do // index of the query items
(5)         j← 1 ; // index of the IS
(6)         While j ∈ [1, s] do // index of the IS known by the RA
(7)                 m← 0 ; // index of the information items of IS_j
(8)                 While m < size (list of info items managed by IS_j) do
(9)                         If Compare (item_i,info_item_m managed by IS_j) then
(10)                                nIS ← nIS +1 ;
(11)                                ISL← ISL ⊕ (IS_j, <item_i>) ;
(12)                        End If
(13)                        m ← m + 1 ;
(14)                End While
(15)                j ← j + 1 ;
(16)        End While
(17)        i ← i + 1 ;
(18) End While
(19) If nIS is equals to 0 then
(20)    query_type ← "without answer"
(21) else
(22)    sid← countDifferent (ISL) ;
(23)    If sid is equals to 1 then
(24)            query_type ← "simple"
(25)    else  // sid is upper to 1
(26)            query_type ← "complex"
(27)    End If
(28) End If
(29) ISL ← Compact(ISL)
```

This algorithm generates an *Information System List* (*ISL*) containing tuples composed of (*IS*, <*list of items of the query managed by the IS*>). First, the *ISL* is empty (line 1). Then, the variable *nIS* (containing the number of *IS* managing the equivalent items to those of the query - line 2), is initialized. For each item of the query ($item_i$ with $i \in [1, n]$, lines 3 to 18, it searches the IS_j (IS_j with $j \in [1, s]$) which manage information items equivalent to the analyzed one. The "*Compare*" method verifies the equivalency of the items (line 9). If they are equivalent, the algorithm increases the variable *nIS*, and adds into the *ISL* a tuple whose first term corresponds to an IS_j and the second one corresponds to the analyzed item (lines 10 and 11). When all the query items have been analyzed, the algorithm verifies the value of *nIS* (lines 19 to 27). If this value is zero means that there is not an *IS* able to manage equivalent items from the query. In this case, the query is tagged as "*simple*". In the other case, the algorithm analyzes the *ISL* in order to know the number of different *IS* managed items equivalent to those in the query, in order to characterize the query as "*complex*". This number is calculated using the "*countDifferent*" method (line 22). Finally, the algorithm uses the "*Compact*" method which leaves in the *ISL* only one tuple associated for each *IS*. The second term of this tuple corresponds to all the query items managed by the *IS*. We illustrate "*countDifferent*" and "*Compact*" methods in order to clarify them: Let us suppose that the *RA* has identified items: i_1, i_2, i_3, i_4 and i_5, and it knows the following *IS*: IS_1, IS_2, IS_3, IS_4, IS_5 and IS_6. After the items' comparison (lines 4 to 19), we suppose that the *ISL* is composed of the following tuples:

$$(\mathrm{IS}_1, <i_1>)\ (\mathrm{IS}_3, <i_1>)\ (\mathrm{IS}_5, <i_1>)\ (\mathrm{IS}_1, <i_2>)\ (\mathrm{IS}_3, <i_2>)\ (\mathrm{IS}_4, <i_2>)\ (\mathrm{IS}_5, <i_2>)\ (\mathrm{IS}_1, <i_3>)$$
$$(\mathrm{IS}_3, <i_3>)\ (\mathrm{IS}_4, <i_4>)(\mathrm{IS}_5, <i_4>)\ (\mathrm{IS}_1, <i_5>)\ (\mathrm{IS}_3, <i_5>)\ (\mathrm{IS}_4, <i_5>)\ (\mathrm{IS}_5, <i_5>)$$

The "*countDifferent*" method will return a value of 4 because in the tuples it only appears IS_1, IS_3, IS_4 and IS_5. The "*Compact*" method leaves only one tuple for each *IS*:

$$(\mathrm{IS}_1, <i_1, i_2, i_3, i_5>)\quad (\mathrm{IS}_3, <i_1, i_2, i_3, i_5>)\quad (\mathrm{IS}_4, <i_2, i_4, i_5>)\quad (\mathrm{IS}_5, <i_1, i_2, i_4, i_5>)$$

For the example, the *ISL* is composed of:

(AirlineIS, <"flight itinerary">) (AgendaIS, <"meetings' itinerary">)
(TouristInformationOfficeIS, <"hotel information">)

Following the *matching algorithm*, we can conclude that the query is "*complex*". An item of the query can be managed by several *IS*, then it is necessary to select the most appropriated *IS* for answering them.

The ***second activity*** corresponds to the *IS selection*. In this activity, a query can be rerouted towards a specific agent or towards a group of agents. If the addressees of a query are known, the selection is relatively easy. Otherwise, the *RA* selects the *IS* and composes the *network of neighbors* (considering the *ISL* produced during the previous activity). This process is based upon the ideas of Yang *et al.* [13]: when a user submits a query, his node becomes the sender of the query and it can send messages (including the query) to several of its neighbors. When a neighbor receives the query, it handles the query using first its local information. If the node finds some results, it returns them to the sender node. In our proposal, a *peer* is named "*neighbor*" of other peers, if it satisfies a set of characteristics (criteria defined in the *user preferences*), such as a close location, similar activities, roles, knowledge, colleagues working in the same group, *etc*. However, characteristics are not restricted to proximity criteria.

We distinguish three cases during the composition of *network of neighbors* in which each node is an *ISAgent* that can potentially answer to a query. The *simplest case* is that in which an agent only handles a query. The *RA* contacts the *ISAgent* which can answer to the query. In the *second case*, several *ISAgents* can answer to the same query. The simplest way of composing this *network of neighbors* is gathering all of these *ISAgents*. This gathering is useful when the *RA* does not have any information about the *ISAgents* or when it is the first time that this *ISAgent* works with the neighbors. In order to avoid useless, redundant or unusable communications, and to select the most relevant neighbors, the *RA* applies query's adaptation criteria. For example, if the criterion is location, the *network* is composed of the close neighbors; if the user queries depend on his previous queries, the *RA* must redirect them to the most trusted neighbors; if the criterion is similarity, the network must be composed of neighbors with a similar profile, similar tasks, *etc*. If there is not a defined criterion, the *RA* analyzes the trust level of its neighbors. The *RA* associates a trust level to each neighbor, based on previous answers applying the *Trust and Reputation* strategy proposed by Agostini *et al.* [1] (see section 2). In the *third case*, a query has been split in several sub-queries at the time of the *analysis activity*. The *RA* analyzes the *ISAgents* that can answer to each sub-query. These *ISAgents* compose the *network*. For each sub-query, it is necessary to select the *IS* that are able to answer. Finally, the *network* is composed of the aggregation of the different generated sub-networks for each sub-query. The *RA* considers the *ISL* produced in the *analysis activity* in order to select the *IS* which are able to answer the (sub) query. For our example, the *RA* selects as *IS* the *AirlineIS*, the one of the *Tourist Information Office* and the one of his *Agenda*. These *IS* are the only ones that are able to answer to each one of the three sub-queries. These sub-queries are redirected to the corresponding *ISAgents*. We give an example of the sub-query $Query_1$ produced by the *TouristInformationOfficeIS* where the *ISAgent* is named *TouristInformationOfficeISA*. According to an equivalent principle, $Query_2$ and $Query_3$ will be redirected respectively to the agents *AgendaISA* of the *AgendaIS* and the *AirlineISA* of the *AirlineIS*:

(*assert* (*Query* (*QueryID Query1*)
(*UserID "Mr. Joseph Doe"*)
(*ISA TouristInformationOfficeISA"*)
(*IS "TouristInformationOfficeIS"*)
(*required_info "hotel information"*)
(*parameters "hotel name" "check-in" "check-out" "hotel reservations"*)))

The ***third activity*** is the *redirection of queries* to the *IS*. In this activity, once the *RA* has identified the potential *ISAgents*, it must analyze the trust level associated to each one of them, to determine a redirection protocol of the query. This information about trust level may be not available, if it is the first time that the *RA* executes this query or if it is the first time it works with these *ISAgents*. In the same way, the *ISAgents*' trust levels can be similar. In these conditions (absence of established trust), the *RA* sends the query in "*broadcast*". When the *RA* has information exploitable in terms of trust, it redirects the query in a sequential way, starting by the most trusted agent. The answer to the query will be the one generated by the first *ISAgent* that fulfills it. If the *RA* does not receive any answer, the user will be informed about the query problem. If the *RA* only knows the *ISAgents* able to answer the sub-queries ($query_1$, … queryN), the *RA* sends directly the sub-queries to these *ISAgents*. In our example, the *RA* sends $Query_1$ to the *TouristInformationOfficeISA*, $Query_2$ to the *AgendaISA* and $Query_3$ to the *AirlineISA*. The *RA* must collect and classify the obtained answers from the different *ISAgents* and select the most relevant ones taking into account established adaptation criteria. A scenario presenting the reception of results can be found in [4].

5 Conclusion

When a user formulates queries, the results can come from different *Information Sources* (*IS*). In this paper, we have defined a *query routing process* as a mechanism which analyzes the query, and performs the item (of the query) and information (managed by the *IS*) matching (semantic or syntactic). This matching is achieved in order to select the *ISAgents* able to answer to user queries. After identifying items and the *IS* that are able to manage equivalent items, this process splits the query. A *Router Agent* (belonging to the *Information MAS* of *PUMAS*) considers adaptation criteria provided by the user in order to choose the most appropriated *ISAgents* for answering the query. Finally, this process must collect and classify the query results. We have illustrated the *query routing* and *enrichment processes* by means of an example implying different *IS* in which an enterprise manager asks for the *flight itinerary*, the *hotel information* and the *meetings' schedules* at a destination city.

Our future work is aimed at the definition of an algorithm for the collection and analysis of results coming from one or several *ISAgents.*

References

1. Agostini, A., Moro, G.: Identification of Communities of Peers by Trust and Reputation. In: Bussler, C. and Fensel, D. (eds.): AIMSA 2004, LNCS, vol. 3192, pp. 85-95, Springer, Berlin (2004)
2. Brunkhorst, I., Dhraief, H., Kemper, A., Nedjl W., Wiesner, C.: Distributed Queries and Query Optimization in Schema-Based P2P Systems. In: Aberer, K., Kalogeraki, V. and Koubarakis, M. (eds.): DBISP2P, LNCS, vol. 2944, pp. 184-199, Springer, Berlin (2003)
3. Carrillo-Ramos, A., Villanova-Oliver, M., Gensel, J., Martin, H.: Knowledge Management for Adapted Information Retrieval in Ubiquitous Environments. In: Filipe, J., Cordeiro, J. and Pedrosa, V. (eds.): WEBIST 2005 and WEBIST 2006, Revised Selected Papers, LNBIP, vol. 1, pp. 84-96, Springer, Berlin (2006)
4. Carrillo-Ramos, A., Gensel, J., Villanova-Oliver, M., Martin, H.: A Peer Ubiquitous Multi-agent Framework for Providing Nomadic Users with Adapted Information. In: Despotovic, Z., Joseph, S. and Sartori C. (eds.): AP2PC 2005, Revised Papers, LNAI, vol. 4118, pp. 159-172, Springer, Berlin (2006)
5. Carrillo-Ramos, A., Villanova-Oliver, M., Gensel, J., Martin, H.: Contextual User Profile for Adapting Information in Nomadic Environments. In: Weske, M., Hacid, M-S. and Godart, C. (eds.): Web Information Systems Engineering - WISE 2007 Workshops, LNCS, vol. 4832, pp. 337-349, Springer, Berlin (2007)
6. Indulska, J., Robinson, R., Rakotonirainy, A., Henricksen, K.: Experiences in Using CC/PP in Context-Aware Systems. In: Chen, M.-S., Chrysance, P.K., Sloman, M., and Zaslavsky, A. (eds.): MDM 2003, LNCS, vol. 2574, pp. 247-261, Springer, Berlin (2003)
7. Kokkinidis, G., Christophides, V.: Semantic Query Routing and Processing in P2P Database System: The ICS-FORTH SQPair Middleware. In: Lindner, W., Masiti, M., Türker, C., Tzitzikas, Y. and Vakali, A. (eds.): EDBT 2004, LNCS, vol. 3268, pp. 486-495, Springer, Berlin (2004)
8. Koloniari, G., Pitoura, E.: Content-Based Routing of Path Queries in Pair-to-Pair Systems. In: Lindner, W., Masiti, M., Türker, C., Tzitzikas, Y. and Vakali, A. (eds.): EDBT 2004, LNCS, vol. 3268, pp. 29-47, Springer, Berlin (2004)
9. Odell, J., Van Dyke Parunak, H., Bauer, B.: Representing Agent Interaction Protocols in UML. In: Ciancarini, P. and Wooldridge, M.J. (eds.): AOSE 2000, LNCS, vol. 1957, pp. 121-140, Springer, Berlin (2001)
10. Röhm, U., Böhm, K., Schek, H.: OLAP Query Routing and Physical Design in a Database Cluster. In: Zaniolo, C., Lockemann, P.C., Scholl, M.H. and Grust T. (eds.): EDBT 2000, LNCS, vol. 1777, pp. 254-268, Springer, Berlin (2000)
11. Thilliez M., Delot T.: Evaluating Location Dependent Queries Using ISLANDS. In: Ramos, F., Unger, H. and Larios, V. (eds.): ISSADS 2004, LNCS, vol. 3061, pp. 126-136, Springer, Berlin (2004)
12. Xu, J., Lim, E., Ng, W.K.: Cluster-Based Database Selection Techniques for Routing Bibliographic Queries. In: Bench-Capon, T., Soda, G. and Min Tjoa, A. (eds.): DEXA 99, LNCS, vol. 1677, pp.100-109, Springer, Berlin (1999)
13. Yang, D., Xu, L., Cai, W., Zhou, S., Zhou, A.: Efficient Query Routing for XML Documents Retrieval in Unstructured Peer to Peer Networks. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.): APWeb 2004, LNCS, vol. 3007, pp. 217-223, Springer, Berlin (2004)