Architecture of a Mobile-Agent of a Distributed Knowledge Management System

Esteban Bailone[1], Pablo Villarreal[1], María Rosa Galli[1,2], Omar Chiotti[1,2]

[1]GIDSATD – Universidad Tecnológica Nacional – Facultad Regional Santa Fe,

Lavaisse 610 – 3000 SANTA FE - Argentina

[2]INGAR-CONICET, Avellaneda 3657 – 3000 SANTA FE – Argentina

pvillarr@frsf.utn.edu.ar, chiotti@ceride.gov.ar

Abstract

This work describes a multi agent system designed to support the management of tacit knowledge that belongs to people of an organization. This is a distributed knowledge management system based on the use of mobile agents, which receive the user's queries and visit the organization domains where this information can be generated. The system has been developed using an approach based on the organizational concept of business processes to identify roles and protocols as part of the analysis stage of a methodology for agent-oriented analysis and design. The mobility of the agent is defined using an approach based on both the quality attributes specified for the multi-agent architecture and the execution environments of the multi-agent system. Particularly, this work is focused on describing the designed mobile agents' architecture and some implementation details of it.

Keywords: multi-agent systems; knowledge management; mobile agent

1. Introduction

The knowledge is the main asset of current organizations, not only to high technology enterprises but also to conventional productive systems. The organization knowledge mainly resides in the people that integrate it, which are generally spatial and timely distributed. Wherefore, a current challenge of organizations is to adequately manage its knowledge. This activity is called knowledge management. The aim of the knowledge management is not to solely manage the previous knowledge. The knowledge of the past, it is only valuable if it can proportionate a perspective of the future: the main usefulness of the knowledge management is its innovation support.

The challenges associated with knowledge management, can be classified into three general categories: acquisition, organization, and distribution. Knowledge acquisition deals with the issues involved in knowledge extraction in its various forms. That is, from the organization's knowledge bases, databases, printed resources, and people. The knowledge acquisition implies to detect who are the people that have the knowledge on a specific area, how is the knowledge in that area currently stored, and how this knowledge can be made machine-readable. Knowledge organization deals with the issues about the best way of storing knowledge so that it can be retrieved when it is relevant. Knowledge distribution, must tackle the problem of getting the right knowledge to the right place at the right time.

We define the knowledge as the information that is integrated and understood in the mind of a given subject. Two knowledge types can be distinguished: explicit and tacit [7]. Explicit knowledge is easily shared whereas tacit knowledge is highly personal. A characteristic of tacit knowledge is that it is task specific and it is related to ability [17]. That is, tacit knowledge gives some abilities that are dependent on the context of application. A part of the tacit knowledge, which implicitly belongs to somebody, can be made explicit, but there is a part of the tacit knowledge that definitely cannot be made explicit [1]. Then, it cannot be automated.

An approach to support the knowledge management is based on developing a corporate or organizational memory. Several works are focused in this direction [12], [4], [10]. An organizational memory is defined [14] as an explicit, disembodied, persistent representation of crucial knowledge and information in an organization, in order to facilitate their access, sharing, and reuse by members of the organization, for their individual or collective tasks.

A corporate memory is appropriate to represent the part of the tacit knowledge and its context that can be made explicit, but the other part of the tacit knowledge and its context, which belong to people, cannot be represented in the corporate memory. That is, in an organization there is information that can only be generated by people, then, when a decision maker in some time and somewhere of an organization requires this information, he/she has to solicit it to who has the ability of generating it. In this way, another type of support to facilitate the access to this knowledge is required.

Our group is developing a multi agent system to support the management of tacit knowledge that belongs to people of an organization. This is a distributed knowledge management system based on the use of mobile agents, which receive the user's queries and visit the organization domains where this information can be generated [2]. The system has to analyze the query in order to define the domains that have the potential of answering it, and then, it has to send this query to each of those domains until to find a domain that can answer it. To support the first task, our group have developed an intelligent agent able to interpret a natural language query and to classify the domains according to their possibility of answering it [13][6]. The second task is in charge of a mobile agent called Query Coordinator Agent (QCA), whose design and implementation details are presented in this paper.

Besides autonomy, reactivity, pro-activeness and social ability [19], another software agent's property is mobility. Mobility refers to the ability of an agent for dynamically transferring its execution onto different sites [18]. Although mobile agents technology present several advantages when compared to middleware technologies used in developing distributed applications, such as RPC or CORBA [3], the advantage of using mobile agents versus using static agents is not quite clear. Both mobile and static

agents can be used to solve the same problems. The difference lies on how each alternative solves the problem. Other mobile agent feature is its clonation ability.

In spite of the wide diffusion that agent mobile technology is having currently, there are few proposes that guide a designer to define the convenience of to use mobile agents. In this work we use the guide proposed by [15] to identify mobile agent.

The aim of this work is to describe the developed architecture of a mobile agent of a distributed knowledge management system and some implementation details of it. In Section 2, we describe the development of the multi agent architecture of the system. In Section 3, we discuss the mobility requirements of the QCA. In Section 4 we present the architecture proposed for this agent and in Section 5 we describe some details of its implementation.

2. Development of The Multi-Agent Architecture

One methodology for analysis and design of Multi-agent systems (MASs) is Gaia [20]. The key concepts of the analysis in Gaia are roles and protocols. Roles can interact with one another in certain institutionalized ways, which are defined in the protocols. Although Gaia claims to allow an analyst to go systematically from a statement of requirements to a sufficiently detailed design, it does not present a procedure to guide roles and protocols identification. Thus, we have decided to apply the approach described in [15] that allows the roles and protocols identification viewing the system as a processes set. This approach consists of 3 stages:

Stage 1: Define the system's goal.

Stage 2: Define the processes through its inputs, outputs and activities.

Stage 3: Identify the necessary roles and protocols in order to realize the activities previously mentioned.

Once the third stage is completed, the Gaia methodology can be applied.

In this section we describe the analysis and design of the system to support the distributed knowledge management, which have been done following the approach described above.

2.1 Stage 1: Define the system's goal

The idea of the system is to offer to the decision makers of each enterprise domain a query mechanism that allows them to find people who have the knowledge to generate the information required by them. Therefore, the goal of the system is "to process decision maker's queries in natural language and to find the possible information sources that allow to obtain answer to the queries requirements".

2.2 Stage 2: Define the processes and its activities

Considering the system as an organization, we define three processes that the system has to carry out to achieve its goal. The inputs and outputs of these processes are showed in Figure 1.

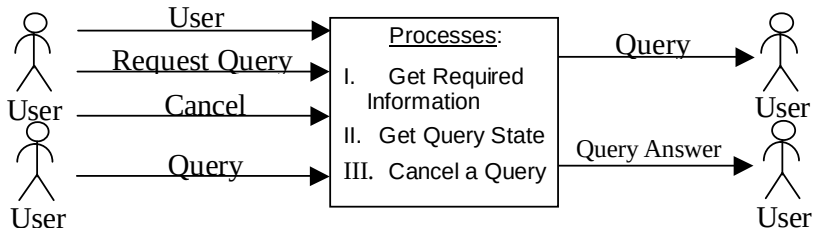

Figure 1. Inputs and outputs of the processes

The three processes defined are: (I) get the required information, (II) get the states of an issued query for information and (III) cancel a query. The first process, showed in Figure 2, is the main one. It consists of seven activities and its goal is to obtain an answer to a user's query. The first activity of this process has the purpose of obtaining the user's query. The second activity has the purpose of coordinating all users' query generated in the domains selecting the next query to be processed. The third activity determines the possible information sources (domains) for providing the required information to the user's query. The forth activity is to visit the domains that the previous activity classified as possible providers of the required information. The fifth activity consists on obtaining the query's answer. The sixth activity has the purpose of delivering the answer to the user that issued the query and the seventh activity has the object of learning from this search, thus improving the knowledge about domains as regards the information they can provide. The other processes are showed in Figure 2 (b) and (c).

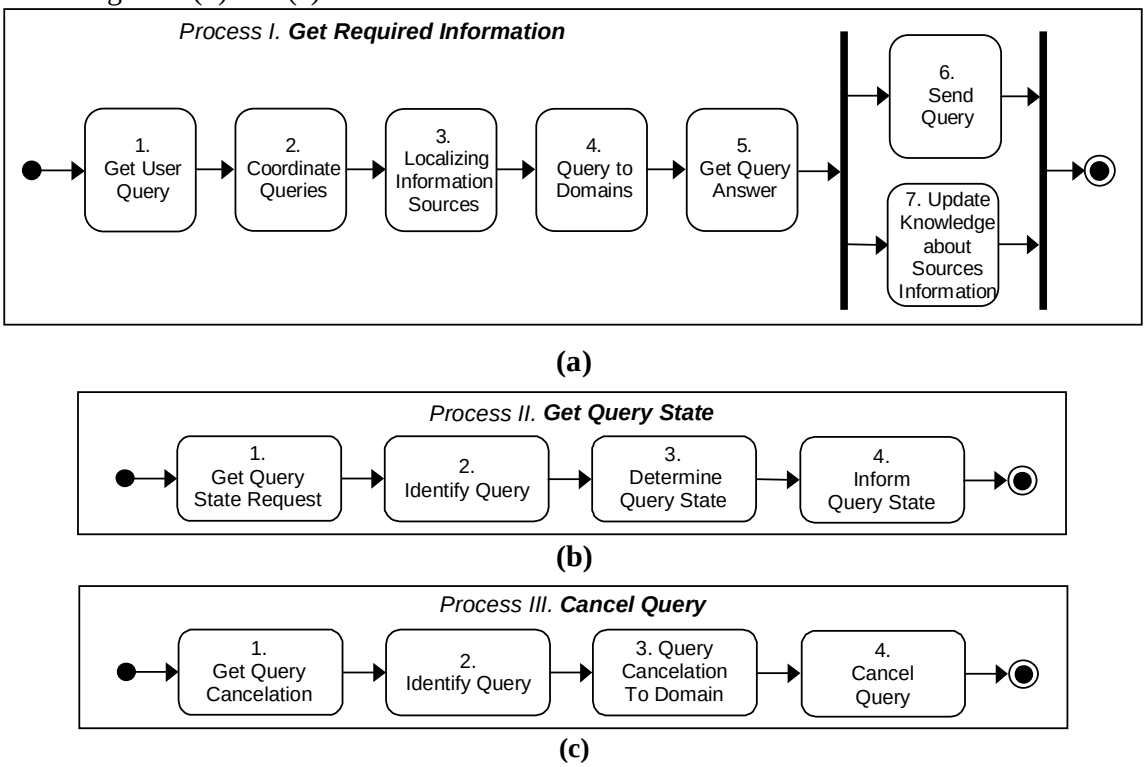

Figure 2. The processes to be carried out by the system viewed as an organization

2.3 Step 3: Identify Roles and Protocols

Analyzing the processes with their activities we have identified the roles and protocols of the system. The system roles and their means are described next.

DomainUser: This role makes queries and query answers. It can also request the query state or can cancel a query. This role is not mapped to an agent because it represents the user and his interaction with the system. However, it is necessary to represent it as this role sets protocols with other roles of the system.

DomainRepresentative: This role carries out tasks that involve the interaction with users. It makes actions that allow a domain to receive queries from the domain users and to receive answers sent by other domains. In addition, it allows a user to request for the state of a query and to cancel a query. The goal of this role is to perform *GetQueryUser*, *GetQueryState* and *CancelQuery* activities, which are part of the defined processes. There is a *DomainRepresentative* role in each domain.

QueryCoordinatorsAdministrator*:* The goal of this role is to perform the *CoordinateQueries* activity of the main process. Its main functions are to coordinate the user's query from different domains and to select the next query to be processed in the system. In addition, it has to perform the *IdentifyQuery* activity to identify the QCA to which a query has been assigned. This is required to give the QCA a user's request for the state of the query and to cancel the query. These are activities of the processes II and III.

InformationSourceLocator*:* The goal of this role is to perform the *LocalizingInformationSource* and *UpdateKnowledgeAboutInformationSources* activities of the main process. This role manages the knowledge about information managed by each domain and based on this information it generates a domain ranking list for a query. This list includes the possible domains that might answer the query.

QueryCoodinator: The goal of this role is to perform the *QueryToDomains* activity of the main process and the activities *DetermineQueryState, InformQueryState, QueryCancelationToDomain* and *CancelQuery* of other two processes. Its main functions are to deliver the query to the possible domains, to obtain the query answer generated by some domain and to inform the query results.

ResponsesSupplier: The goal of this role is to perform the *SendQueryResponse* activity of the main process. There is a *ResponseSupplier* role in each domain that sends the query answer to the domain that originated it.

Once these roles have been identified, the protocols are defined following the activities and their relationships in the process (more details can be see in [15]). Figure 3 presents the roles and protocols derived from the defined processes. The activities assigned to each role are shown by a tuple (Process_Number; Activity_Number). Protocols that settle interactions among roles are shown with arcs among roles, and the main internal activities that each role has to perform are indicated with a loop.

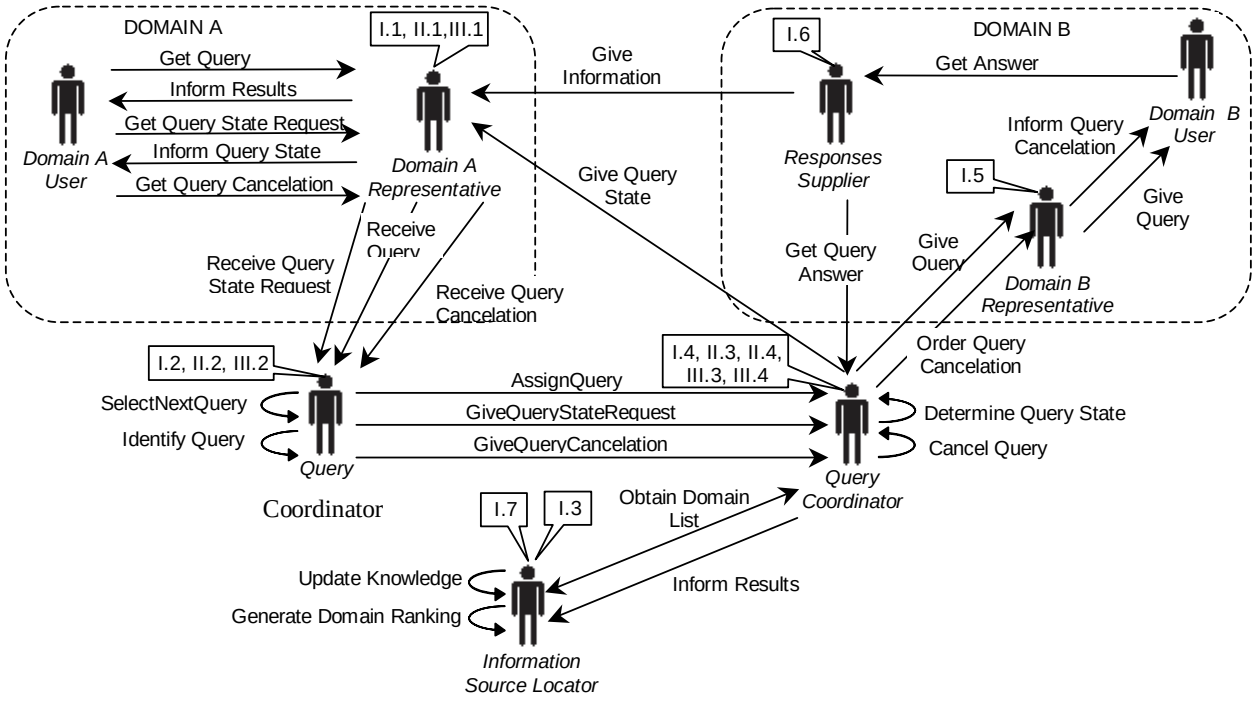

Figure 3. The roles and protocols from the processes

At this point, Stage 3 is finished. From now on, we can keep applying Gaia to obtain the Agent, Interaction, Acquaintance and Services Models. As there is not space enough to fully explain all these models, only the Agent and Acquaintance models are presented in this work.

2.4 Defining the schema of the roles and the protocol attributes

Once the roles and the protocols are identified, we defined the schema of the roles and the protocol attributes to complete the role model and the interaction model. This is made following the Gaia methodology. As an example, Figure 4 shows a schema of the Q*ueryCoordinator* role.

Role Schema: *QueryCoordinator*

Protocols and Activities:

AssignQuery, ObtainDomainsList, GiveQuery, WaitInDomain, GetQueryAnswer, InformResults, GiveQueryStateRequest, DetermineQueryState, InformQueryState, RePlan, EndConsult, GiveQueryCancelation, CancelQuery, OrderQueryCancelation, GenerateSearchPlan, UpdateQueryState, CreateQueryAnswersList

Permissions:

reads supplied UserQuery

reads supplied DomainsList

generates QueryStateReport

generates QueryAnswerList

Responsibilities

Liveness:

QUERYCOORDINATOR = AssignQuery.ObtainDomainsList.GenerateSearchPlan.
(SelectDomainToConsult.(CONSULTDOMAIN | RePlan | EndConsult))$^{+}$.
GIVEQUERYANSWERLIST || QUERYSTATEREPORT || QUERYCANCEL.

CONSULTDOMAIN = GiveQuery.(WAITANSWER | RePlan)

WAITANSWER = (WaitInDomain.GetQueryAnswer.UpdateQueryState)

GIVEQUERYANSWERLIST = (CreateQueryAnswersList.InformResults.EndConsult)

QUERYSTATEREPORT = (GiveQueryStateRequest.DetermineQueryState.InformQueryState)

QUERYCANCEL = (GiveQueryCancellation.CancelQuery.OrderQueryCancellation)

Safety:

- True
  UserQueries < 1 // number of user queries simultaneously attended
  QueryAnswers < m // number of query answers simultaneously attended

Figure 4. Schema of the *QueryCoordinator* role

2.5 Defining the agent, service and acquaintance models

Defining the agents model implies to define the mapping among roles and agents. In this way, the roles to be carried out by agents are settled. Figure 5 shows the four agents types that constitute the multi-agent system. These agents are the Domain Representative Agent (DRA), the Information Source Locator Agent (ISLA), the Query Coordinator Administrator Agent (QCAA) and the Query Coordinator Agent (QCA). In this model, it can be see that the roles that each agent has to perform to achieve the goal of the system. The number of DRA's instances depends on the number of domains to be supported by the system, as there is a DRA for each domain, and regards to the QCA's instances,

there will be one for each query. Once the agent model has been defined we have to define the services model. This model is out of the scope of this paper.

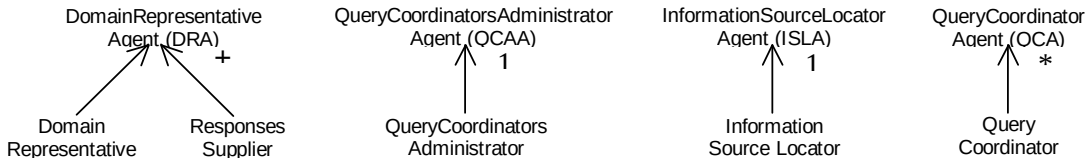

Figure 5. The agent model

Figure 6 shows the acquaintance model. This is the last model to be generated. It shows the communication links between agent types. The DRA has a loop that represents a communication link between agents of the same type.

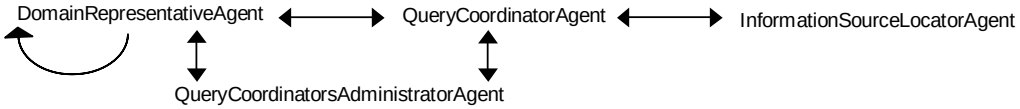

Figure 6. The Acquaintance Model

3. Identification of Mobile Agents

An agent should meet the following properties: autonomy, reactivity, pro-activeness and social ability. These properties are present in roles performed by an agent and thus they are taken into consideration when modeling the agent. Agents' internal activities, which are defined by the roles they perform and the interactions (protocols) among them, can be always carried out locally in the same environment (using mobile agents) or in a distributed way (using static agents). The difference lies on the advantage a solution may have over another. These advantages can be identified if we take into account the software quality attributes the developing MAS should possess. Then, it is possible to settle some criteria to decide whether an agent must meet the mobility property according to certain quality attributes the MAS is intended to have.

Some software quality attributes for multi-agent architectures were identified from a perspective of organizational styles as architectural styles: predictability, security, adaptability, coordinability, and so on [5]. Quality attributes are not all related with choice between mobile agents o static agents. For example, security is a quality attribute that can be accomplished using a solution with an architecture where there are some static agents that support the security requirements or another architecture where some mobile agents exist. Using mobile agents does not contribute to accomplish the security requirements. In addition, there are different approaches to support security in applying mobile agents because security is a concern in these applications. However, there are some quality attributes of MASs that mobile agents can contribute to reach. These quality attributes are:

Performance. If an agent must interact with a great number of agents that are located in distributed execution environments, the use of mobile agents may allow for an improved system performance, considering both performance perceived by the user and the network performance. This can be achieved because one of the mobile agents' characteristics is that they can perform the same activity in different sites in a parallel way. This parallelism is possible as the mobile agents' ability to clone, which enables the same task or different tasks to be performed by the same type of agent in different sites and at the same time. A comparison between mobile and static MASs from the performance point of view was done in [9]. In that work, solutions to a problem of text search distributed in a network

files system were developed, using different approaches to the MAS (static versus mobile) and these alternatives were tested. They conclude that the mobile agents system yields a better performance than the static agents system when cloning is used to carry out the search.

Reliability. Using mobile agents may increase reliability and availability measures related to certain aspects. For example, an agent may have to interact with several agents in different execution environments. If this agent is static and the environment where it is executed is shut down, the agent is shut down too. In contrast, if this agent is mobile, once it has moved to another execution environment, it can go on with its execution and reach its goal, even when its execution environment of origin is down. In addition, a mobile agent can choose the activities to carry out according to its knowledge about the state (up or down) of others agents and their execution environment. So, mobile agents contribute to improve the ability of a MAS to keep on operating over time to reach its goals.

Scalability. The number of agents of some type in the system can be increased in runtime using cloned mobile agents, which allows dynamically increasing the number of tasks to be carried out by the system.

Adaptability. Mobile agents can improve the ability of a MAS to be suited to different execution environments, changing in these environments some rules that control the way in which mobile agents interact.

Agents are situated entities of a MAS considered as an organization. As situated entities, agents have an environment in which they are executed and interact with other agents. Generally, agents are executed in environments that may be dynamic, open, distributed and heterogeneous. Different environments in a MAS can affect the way in which system goals are reached so they can affect the design of activities and interactions of each agent. Identification of execution environments of MASs will help us to decide when using a mobile agent is appropriate.

To identify mobile agent, we have used the guide proposed by [15].This guide proposes the following steps:

a. Identify the execution environments in which each system agent will carry out its tasks and interactions.

b. Identify communication pathways between agents of different execution environment. This may be carried out considering the acquaintance model and the defined execution environments. In this way, we can identify a priori the agents that may be defined as mobile ones.

c. If some quality attributes for the multi-agent architecture related with mobile agents are specified, it is possible to define the mobility property for the agents identified in the previous step.

In this way, analyzing execution environments and looking in the acquaintance model, the use of mobile agents to reach some quality attributes of a MAS can be decided. In the MAS that we are developing, we have identified two types of execution environments. Firstly, let's consider the DRA associated with a domain; anytime, a user may want to make a query or queries from other domains may arrive asking for information. In this way, the DRA, which is in charge of interacting with the user, must be executed in the domains environment. Since domains are geographically distributed, there will be an execution environment for each domain. Secondly, as reliability reasons, it is convenient to have another distributed environment different from those of the domains where the ISLA and the QCAs will be executed.

As regards the previously defined approach, communication links among agents of different environments were identified. If we observe communication links from the acquaintance model in Figure 6, it can be identified that interactions among DRAs, ISLA and QCAs is performed through distributed environments just as interactions among DRAs.

Then, it is convenient to develop the DRA as a static agent since it must be always available to receive queries or answers from users of its domain or queries or answers of other domains. As its main role in the architecture and to the fact that there will be a sole instance of it, the ISLA should be developed as

a static agent. Moreover, the ISLA makes it possible that DRAs need to know only one agent to try to obtain an answer to any kind of query. As regards QCAs, there is no requirement that compel them to be static. In this way, the mobility property could be considered for QCAs.

In order to decide whether it is necessary for QCAs to be mobile, we resorted to the quality attributes specified for the MAS architecture. Among the attributes defined for the system, we mention performance and reliability. As we have previously discussed, these attributes may be met by using mobile agents.

Performance was defined from the reduction in the network load and the time to response the user. To reduce the network load, it is necessary to reduce messages in the network. This can be achieved by using mobile agents. Response time can be reduced by using mobile agents, with the cloning technique, as show in [9].

Reliability was defined as the need for the system to be able to achieve its goal, no matter whether the DRA that sent the query and the ISLA are in execution or not. Maybe a query can be solved in hours because a domain that must answer is not available. Therefore, it is necessary to settle a reliable mechanism that allows the system to keep on operating to satisfy the search. This can be fulfilled by adding mobility to QCAs. In this way, QCAs can visit domains and stay there waiting for a DRA's answer for a certain period of time, no matter whether the ISLA and its execution environment are active or not. Moreover, as it is mobile, the QCA can decide which domain it will visit taking into account the ranking of domains, the DRA and its execution environment availability, to which it must deliver the answer.

Therefore, given the performance and reliability quality attributes to be met and according to what has been previously mentioned, we have added the mobility property to the QCA agent type. Thus, the system architecture is now conformed by two types of static agents, namely DRAs and ISLAs, and one type of mobile agents, namely QCAs.

4. Development of the Query Coordinator Agents Architecture

The main QCAs functions are to deliver a query to the possible domains, to obtain the query answers generated by these domains and to inform the ISLA the query results. But they are also responsibly for to cancel their work or to inform the query state when it is required by the user. To perform these functions QCAs have to interact with several agents types and to carry out diverse activities as events that happen in the environment. In this way, a QCA has to take out several decisions from the time that it receives a query to the time that it ends the work.

Observing the schema of the *QueryCordinator* role described in Figure 4 and the roles and protocols that the multi-agent system has to perform, graphically represented in Figure 2, it can see that when the QCAA receives a query performs its *SelectQuery* activity and assigns the query to a QCA performing the *AssignQuery* protocol. The QCA obtains the domains list to be visited performing the *ObtainDomainList* protocol with the ISLA. Using this protocol, the QCA gives the query to the ISLA performing the *RequestDomainsList* subprotocol. Then, the ISLA performs the *GenerateDomainRanking* to generate the domain ranking list activity and finally it returns this list to the QCA performing the *GiveDomainList* subprotocol. Following, the QCA generates a plan to visit each domain of the list performing the *GenerateSearchPlan* activity. Once a plan has been defined, it selects a domain to be consulted performing the *SelectDomainToConsult* activity. Depending on the environmental state and its beliefs (for example, the query has been positively answered, the domain ranking list becomes empty, the query has been partially answered, the domain to be visited is down, a timeout has occurred) the QCA can make the decision of to consult the domain, re-plan or end the consult. When QCA decides to consult the selected domain interacts with its DRA performing the *GiveQuery* protocol to deliver the query to the DRA. This protocol includes the *AnswerTimeNegotiation* subprotocol that allows the QCA and DRA to negotiate the time to wait for an answer. Depending on the results of this negotiation, the QCA can decide both to re-plan its activities, or to wait for an answer. In this last case, when the domain has the result, through its DRA interacts

with the QCA performing the *GetQueryAnswer* protocol to deliver the result to the QCA. Following, the QCA update the query state performing the *UpdateQueryState* activity.

When the QCA decides to end the search it performs the *CreateQueryAnswerList* activity to generate a results list and then interacts with the ISLA performing the *InformResults* protocol to give it this list.

Whilst the QCA is performing its roles belonging to the main process, it could be required to perform some roles belonging to the secondary processes: Get Query State or Cancel a Query. These define the *QueryStateReport* and the *QueryCancel* liveness properties for the QCA. Performing the *GiveQueryStateReport* protocol the QCAA interacts with the QCA to give it an user's request about the state of his/her query. The QCA identify the query state performing the *DetermineQueryState* activity and then inform this state to the DRA of the user's domain performing the *InformQueryState* protocol.

When the user decides to cancel a query, the QCAA througt the *GiveQueryCancellation* protocol interact with the QCA to give it a query cancellation order. The QCA performs the *CancelQuery* activity to cancel the search, and interacts with the DRA of domains to which the query has been sent performing the *OrderQueryCancellation* protocol.

To satisfy the required functions of the QCA we have developed it following the BDI (Beliefs, Desires, Intentions) model [11]. The defined architecture is schematically represented on Figure 7.

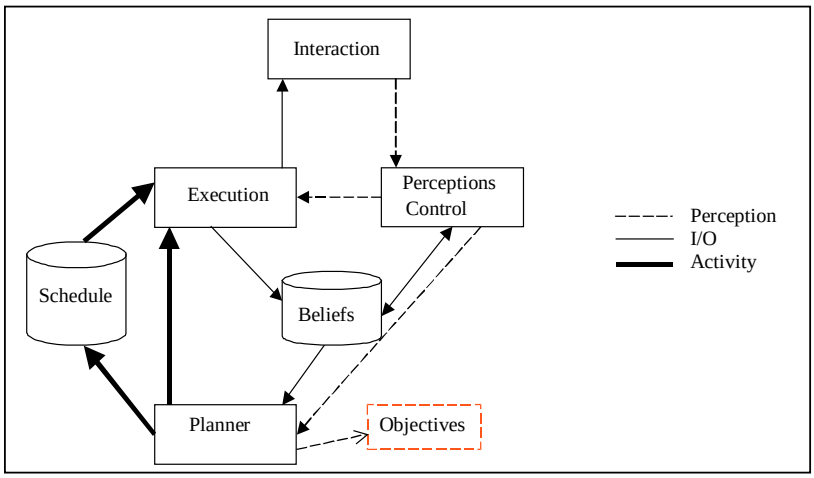

Figure 7: The QCAs architecture

Interaction: This component is in charge of the communication with the other agents of the system (QCAA, ISLA, DRAs). Then, it implements all protocols that the QCA has to perform. Also, it informs the *Perceptions Control* component the perceptions that the agent receives.

Perceptions Control: This component stores in the *Beliefs* component each perception received from the *Interaction* component and then it decides among three options:

- Send this new perception to the *Execution* component as it is required to resume the current activity.
- Send this new perception to the *Planner* component in order that it defines the new activity to be carried out.
- Make anything:

Beliefs: This component stores information about the environment. For example, the active domains list, the agent's current location, the clones location, the domain ranking list and so on. This information is registered whilst the processes are carried out.

Planner: This component receives perceptions from *Perceptions Control* component and analyzing its beliefs defines a new activity to be carried out and its location in the schedule. This component is in charge of the decisions that the agent makes in the run time.

Schedule: This component schedules the tasks that the agent has to execute. This schedule changes every time that the *Planner* component takes a decision.

Execution: This component usually takes from the *Schedule* component the next activity to be executed. Occasionally, it receives from the *Perceptions Control* component a perception that allows it to resume the activity that it is carrying out.

It can also happen that the *Planner* component decides that a new activity to be scheduled can be simultaneously executed with the current activity.

An activity executed by the *Execution* component can change the beliefs of the QCA.

5. Prototype Implementation

A prototype of the QCA has been implemented using Aglets Software Development Kit of IBM [8]. All the components about described were implemented. However, only the most interesting aspects will be reported in this section.

5.1 Clonation Mechanism

The QCA has been implemented with a clonation mechanism. The clonation process has two main activities: the initialization of the clones and the gathering for their later elimination.

The QCA decides to create a clone when, during a negotiation with an DRA, it receives a too long answer time as counterproposal. In this time The clone takes charge of waiting for an answer. Simultaneously the original QCA leaves the domain and goes to its next destination.

When a clone is created, it location is registered by the QCA. A clone has the same capacities that an original QCA, but it does not carry out the same activities. The assigned initial goal is to conclude the negotiation begun by the original QCA. Then, it has to wait for an answer from the DRA and to respond the messages that it receive from the original QCA.

When a clone receives a positive answer from the DRA (performing the protocol *GetQueryAnswer*) it has to inform this event to the original QCA. For that matter, it solicits the QCAA the location of the original QCA. Then, if the QCA decides to conclude the search, it communicates this decision to all its clones that have to cancel their wait (performing the *OrderQueryCancelation* protocol) and go back to it. The original QCA extracts of each clone that arrives the gathered feedbacks and then eliminates it. Once all clons have been collected, the QCA informs the result to the ISLA (performing the *InformResult* protocol) and ends the query (performing the *EndConsult* activity).

5.2 Answer Time Negotiation

As has been described in section 4, the *GiveQuery* protocol includes the *AnswerTimeNegotiation* subprotocol. In the prototype, the negotiation process between the QCA and the visited DRA, has been implemented fulfilling the following stages:

- If the DRA is not active, the negotiation fails.
- If the DRA is active, the QCA sends it the query and it proposes the DRA a time to answer. The DRA can respond it or not:
  - If the DRA does not respond, the negotiation fails.
  - If the DRA responds proposing an answer time different to the one proposed, then:
    - The QCA can accept the proposed time and to stay waiting for an answer. Successful negotiation.
    - The QCA can accept the proposed time and to leave a clone waiting for an answer. Successful negotiation.

6. Conclusions

This work shows that using the business processes concept it is easy to define the roles and protocols that a system has to perform to reach its goal. Then, based on these roles and protocols the architecture of the multi-agent system can be defined. Furthermore, it shows that through the specification of the quality attributes of the system and the execution environment of each agent type it is possible to decide when is better to use a mobile agent than using a static one.

We also showed that these approaches allowed to specify the functionality of each agent with an appropriate level of details, in such a way, that it has not be troublesome to define its architecture. Particularly, from the required activities and protocols defined to the QCA has been easy to see that the BDI (Beliefs, Desires and Intentions) model was the more appropriate one.

Finally, the implementation of the clonation mechanism in the QCA prototype, allowed to reduce the search time, as it allows the parallel execution of a search.

References

[1] Brézillon P & Pomerol J-Ch Contextual knowledge sharing and cooperation in intelligent assistant systems. Le Travail Humain, Vol. 62, No. 3, (1999), pp. 223-246.

[2] Cabral, L., Caliusco, M., Nissio, H., Villarreal, P., Galli, M.R., Taverna, M. and Chiotti, O.,"Use of Agents Technology to Support Distributed Decision Processes", *First World Conference on Production and Operations Management*, POM. Sevilla, Spain. August, (2000).

[3] Chess, D., Harrison, C.G., Kershenbaum, A.: Mobile agents: Are they a good idea? . In Mobile Object Systems: Towards the Programmable Internetwork, ed. by J. Vitek and C. Tschudin. Berlin, Germany: Lecture Notes in Computer Science (1997) 25-47.

[4] Dieng, R. Knowledge Management and the Internet. *IEEE Intelligent System.* (May/J, 2000), pp. 14-17

[5] Kolp, M., Mylopoulos, J.: Software Architectures as Organizational Structures. In Proceedings ASERC Workshop on "The Role of Software Architectures in the Construction, Evolution, and Reuse of Software Systems", Edmonton, Canada (2001).

[6] Mozzati, C., Taverna, M.L., Caliusco, M.L.,Chiotti,O. And Galli, M.R. "Agent for Information Source Location in a Dynamic DSS. *Actas del VI Congreso Argentino de Ciencias de la Computación (CACIC'2002)*. Buenos Aires, 15 al 18 de octubre de 2002.

[7] Nonaka, I. A Dynamic Theory of Organisational K Creation. Org. Science. 5, 1, (1994), pp. 14-37.

[8] Lange, D.B. IBM Tokio Research Laboratory (1997),"Java Aglet Application Programming Interface (J-AAPI). White Paper, http://www.trl.ibm.co.jp/aglets/JAAPI-whitepaper.html

[9] O'Malley, S.A., Self, A.L., DeLoach, S.A.: Comparing performance of static versus mobile multiagent systems. Proc. of the National Aerospace and Electronics Conference (2000).

[10] Rabarijaona, A; Dieng, R.; Corby, O.; Ouaddari, R. Building and searching an XML-based corporate memory. *IEEE Intelligent System.* (May/June, 2000), pp. 56-63.

[11] Rao, A.S. and Georgeff, M.P., "Modeling Rational Agents within a BDI-Architecture. *Tech. Report 64*,Australian Artificial Intelligence Institute, Melbourne, Australia, Febrruary, 1991.

[12] Schwartz, D. and Dov Te'eni, D. G. Tying Knowledge to action with kMail. *IEEE Intelligent System.* (May/June, 2000), pp. 33-39

[13] Stegmayer, G., Taverna, M, Chiotti, O. and Galli, M., The Agent Routering Process of Dynamic Distributed DSS, *Journal of Computer Science & Technology*, Vol. 5, No. 2, (2001), pp. 30-43

[14] Van Heijst, G.; Van der Spek and Kruzinga, E. Organizing corporate memories. Proc. 10th Banff Workshop on Knowledge Acquicition for Knowledge-Based Systems 1996.

[15] Villarreal,P., M. Alesso, S. Rocco, M.R. Galli, O. Chiotti. "Approaches for the Análisis and Design of Multi-Agent Systems". *Actas del 31 Jornadas Argentinas de Informática e Investigación Operativa (31 JAIIO).* Santa Fe, 9 al 13 de septiembre de 2002

[16] Villarreal,P., M.L. Caliusco, M.R. Galli, O. Chiotti. "Multi-agent Distributed Knowledge Management System". *Proceeding infoUYClei 2002*. ISBN 9974-7704-1-6. Montevideo (Uruguay), noviembre, 2002

[17] Wallis, C. Consciousness, context and know-how. http://www.arts.uwaterloo.ca , 2000.

[18] White, J: Mobile Agents. In Software Agents, ed. by J. Bradshaw, AAAI Press (1997) 437-472.

[19] Wooldridge, M., Jennings, N.: Intelligent agents: Theory and Practice. The Knowledge Engineering Review, 10(2) (1995) 115-152.

[20] Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems. Vol. 3(3), 2000.