THE AGENT ROUTERING PROCESS OF A DYNAMIC DISTRIBUTED DECISION SUPPORT SYSTEM

G. Stegmayer[(1)], M. L. Taverna[(1)], O. Chiotti[(1,2)], M. R. Galli[(1,2)]

[(1)] GIDSATD – UTN – FRSF – Lavaise 610 – 3000 SANTA FE – Argentina

[(2)] INGAR – CONICET – Avellaneda 3657 – 3000 SANTA FE – Argentina

mrgalli@ceride.gov.ar, chiotti@ceride.gov.ar

Abstract. When certain enterprise domain needs to make an unusual decision, it will probably require new kind of information. If the interaction model among domains cannot be defined because the decision scenarios are unpredictable, the system itself must search the required information analyzing where it is available, or can be generated. To do that, it is necessary to design a mechanism able to decide where to look for the information.We have developed a distributed DSS able to work in a dynamic way. This system is based on the use of mobile agents, which receive the user's queries and visit the DSS domains gathering the required information. One of the main elements to the efficient operation of this system is the Router Agent. Its responsibility is to give an adequate route to the mobile agents. In this paper we propose a structure for the Router Agent. It includes a Knowledge Base where the information managed by each domain is represented, and a mechanism to compare the query content with the domain representation. We also present an interactive strategy to obtain the initial data to be stored in the Router Knowledge Base. The AHP method is used to support the task of evaluating the relative importance among terms representing each domain.

Keywords. Distributed System, Mobile Agent, Router Agent, DSS, Information Retrieval, AHP.

1. INTRODUCTION

The new organizations of productive systems are based essentially on horizontal structures, with few hierarchical levels and an important delegation of decision along the whole production chain, which no longer involves the company but the whole supply chain. This atomization of the decision process requires an appropriate system that supports each decision point of the organization, able to coordinate the whole decision process.

The decision points of an organization are generally located on different functional units geographically disperse, use decision models and techniques specific to each decision type and they usually share only some information derived from precise decisions. Therefore, a "global" decision support system should be designed as a set of geographically distributed subsystems (Domains), operating with the smallest level of possible joining, which we will call Global Decision Support System (GDSS).

Nagel et al. (1999), have designed a distributed architecture for a GDSS based on the federation concepts. This is an event driven mechanism where each domain must specify both its responsibilities for publishing the information it generates and its subscription to the information produced by other domains. In this way, by means of this contract-based mechanism, the architecture only establishes information dependence among system components that implicates both weakness coupling and domain autonomy conservation.

The interaction model of this distributed architecture is flexible in the sense that at design time it can be adapted to particular decision-processes requirements of a particular industrial organization, and it is a robust interaction mechanism among domains with a fully defined data structure. But this interaction model does not allow the domains to ask for and to receive new information types. Then, the domains cannot support any particular requirements that have not been foreseen at design time.

When a domain needs additional information to which it is not subscribed, it is necessary to determine if any other domain is responsible to publish this information. Otherwise, it must identify which domain is able to provide the required information. In this case, the GDSS must have the same level of intelligence to facilitate the search in a dynamic way.

That is, there exist two types of decisions that should be contemplated for designing a GDSS: static decision processes that define predictable scenarios, and dynamic decision processes that define non predictable scenarios.

A predictable scenario is one that can be previously specified to design the interaction model, while a non predictable scenario is one that is not foreseen at design time. On that basis, Caliusco et al. (2000) proposed to develop an architecture for the GDSS with two mechanisms of interaction among domains: a contract-based mechanism, which supports the static decision process, and a mobile agent-based mechanism that allows domains carrying out searching and gathering of information not included in these contracts. In this paper, we will only consider the second mechanism.

2. MOBILE AGENT-BASED MECHANISM TO SUPPORT DYNAMIC DECISION PROCESSES

When the interaction model among domains cannot be defined because the decision scenarios are non predictable, the system itself must know where to search for the required information analyzing where these data is available, or can be generated. To do that, it is necessary to design a search mechanism able to decide where to look for the answer. In this context, the mobile agent technology appears as the most suitable for this case (Chess et al., 1998; Rus et al., 1998; White, 1997).

An agent can be described as a software component that performs a specific task autonomously. Two classes of agents can be identified: static and mobile agents. While static agents are software components that stay in their respective computers using the network only to transfer information, mobile agents are software components that can migrate between different locations of a network in order to perform their tasks locally. Since mobile agents maintain an internal execution state and/or data state, they are able to perform different parts of their tasks sequentially on different locations into a distributed system. This concept of remote programming avoids the need for transferring a large amount of data across a network. At the same time, an agent can be regarded as intelligent or not, depending on whether it is able to learn or not.

The mechanism proposed to support the dynamic decision process is based on the use of mobile agents to carry out searching and gathering of information. Within the architecture itself, we can find a series of fundamental elements, among which there are static, mobile and intelligent agents. These components are namely: Collecting Agents, the Representative Agent, the Collecting-Agents Server, the Gatekeepers and the Router Agent (Caliusco et al., 2000). Now, we will provide basic notions about each of them. We present this architecture in Figure 1.

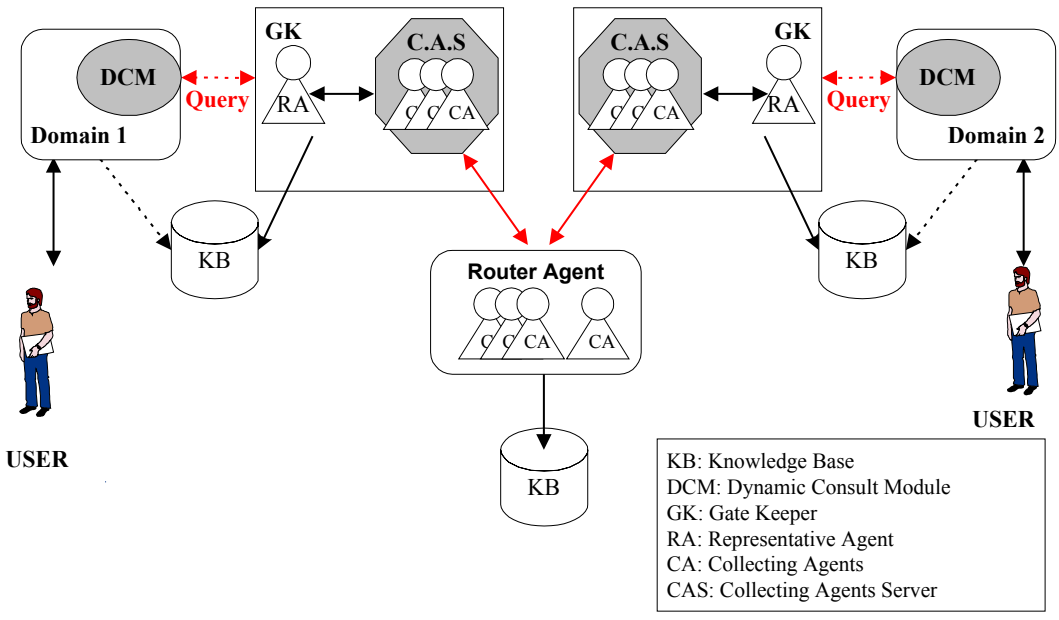

Figure 1. Mobile Agents to support the dynamic decision process

Collecting Agents (CAs): The function of the CAs is to receive the user's query and to obtain a list of domains that can provide an answer, according to the query. Then, they must visit each of those domains looking for the information, and once the collection phase is finished, they must return to the original domain to give the user the obtained information.

Representative Agent (RA): All domains have a RA that represents them. This is a static agent located in the domain Gatekeeper, which must be always up and running. Its functions are the following:

a) To send the consults coming from domain users to the collecting agents, so that the last ones can search for the information.

b) To represent the domain when a collecting agent belonging to another domain arrives asking for information.

c) To transmit the previous queries answers to the collecting agents.

To sum up, the domain representative agent acts as a mediator between the domain and the collecting agents belonging to either its own or another domain.

Collecting Agents Server (CAS): Currently, the mobile-agents technology is based on server programs that serve as a habitat for the agents. These servers are the mediators between the agents and the system resources. They must provide protection for the local system against the coming agents so as to avoid the harmful action of any evil agent. Since the proposed architecture is based on mobile agents, it must necessarily have server programs that are capable of handling all the functions associated to such agents and of protecting machines where the agents live together. The Collecting Agents Server, is the element that plays this role in this architecture and it is located inside the Gatekeeper just as the RA, which was described in the previous paragraph. Every interaction between an external program and the collecting agents takes place through the CAS.

Gatekeeper (GK): It is an abstraction of the set of programs that must be always up and running; no matter if the domain they belong to is active. In our case, the domain RA, the CAS and a third element must remain active. This third element has not been mentioned yet and is the Knowledge Base (KB) of the domain. These three components constitute what we call Gatekeeper and in fact, none of them is compelled to reside in the same machine where the others reside.

Router Agent: This is an intelligent static agent independent of the domains and the Gatekeepers. Its function is to receive the queries of the incoming CAs and, according to the information in its Knowledge Base (KB), to give the CAs a route with the domains that can provide them with an answer. Also, assuming that it must deal with mobile agents, it must have the capability of providing such agents with a safe execution environment.

This mobile agents-based mechanism operates in the follow way:

When a decision-making user needs specific information that he/she is not actually receiving, he/she gets into the Dynamic Consults Module. There, with the help of an assistant, he/she expresses the information he/she would like to receive.

Once the query is made, the assistant passes it to the domain RA, which will change the query into a format accepted by the mobile agents and will send it to the Collecting

Agents Server (CAS). At this moment, the CAS gives the query to a domain CA which goes to a special agent server: the Router Agent. This one receives the CA that has just left its domain, reads the formulated query and according to the information kept in its KB about the domains, gives the CA a list of domains (route) that would provide the searched information.

This route can contain zero to n domains. If the searching route generated by the Router Agent does not contain domains where to search for the information, it is implied that the Router Agent does not have any piece of information in its KB that can associate the required information with one or more of the existing domains. The lack of this association does not mean that the information being searched does not exist or cannot be generated. In this case, the CA can send a request to the Router asking for the list of all known domains and can visit them searching for the information.

If the agent is working in sequential mode, it will simply go through the scales of its route as they appear. One by one, it will visit all the domains making its query. Once it has finished and is about to go back to the original domain, it will visit the Router Agent to inform it the obtained results. This allows the Router Agent to have new information for bringing up to date its KB for future queries. Once made all this, the CA will go back to its domain, will deliver the information and will destroy itself.

If the CA is not working in sequential mode but in parallel mode, after the domains list is obtained, the agent will generate clones of it, one for each of the domains to be visited (except for the one it will visit). Thus, all the clones and the original agent will be sent to their respective destinations. Each of them will travel to a domain, will present the query to the respective domain Representative Agent, will collect the results and will go back to the Router. Once all the agents, the original one and the clones, meet in the Router, the original agent asks its clones, one by one, for their information to be structured. After the clones give the original agent its information, this one tells the Router the result of its incursion in the assigned domain. After that, each clone destroys itself. Once the original agent is ready to go back, it returns to the original domain.

The RA centralizes this Domain Information Retrieval (DIR) process because only its KB has knowledge about each domain belonging to the GDSS. The system performance is improved by avoiding its overload by agents, that is, the domains are not constantly interrupted with many queries they are probably not able to answer. The centralized structure used by the DIR process enables an updated KB. The Router Agent receives the information collected by the CAs and, using a learning mechanism, it corrects and updates its knowledge about the domains.

The Router Agent DIR process allows interpreting the queries and deciding which domain would answer them. This mechanism is different to those published for IR from big document collections (Baeza-Yates and Ribeiro-Neto, 1999), because in a dynamic GDSS there are not texts to be gathered.

The goal of this work is to analyze the Router Agent of this system, and particularly to describe both its mechanism of query understand and its strategy to identify domains able to give an answer.

3. THE ROUTER AGENT STRUCTURE

As we have shown above, the Router Agent has two well defined main functions:

a) To act as an agents-server,

b) To manage the knowledge about the information handled by each domain composing the GDSS.

For that reason, we define its structure with a two-component model (Figure 2):

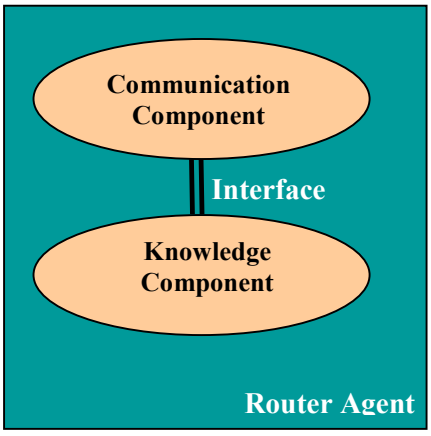

Figure 2. The Router Agent Structure

The *Communication Component* is responsible for communicating with the agents that arrive at the Router Agent. That is, it receives their queries and sends them to the knowledge component for their prosecution. After receiving the answer from this, the communication component sends it to the agents that have arrived with the query. And finally, this component receives the information obtained from the visited domains to update the Knowledge Base, before the collecting agent goes back to the original domain.

The *Knowledge Component* is responsible for two tasks:

a) Interpreting the query transported by the collecting agents, and inferring a probable set of domains able to answer the query. We will call this task Domain Information Retrieval into the GDSS.

b) Maintaining and up dating their KB by means of a learning process that uses the information, carried out by the collecting agents after finishing their travel.

An *interface* between the two components is responsible for the communication between them, that is, for transferring queries, answers and "feedback".

In this paper we will concentrate our attention on the mechanism by which the Knowledge Component carries out one of the assigned functions: to determine the set of domains that can give an answer to the users need for information. That is, how it

interprets the content of the query, in which way the information managed by the different domains is stored at the KB and how it accesses to this information. The learning mechanism will be discussed in a future work.

4. INFORMATION RETRIEVAL INTO A GDSS

An Information Retrieval system (IRS) is a tool that allows representing and storing information to be automatically retrieved at the moment required by the user. The most developed is the Information Retrieval (IR) area concerned with selecting from a collection of documents those that are relevant for the information needed by the user. Several works have been published at the last ten years (Mladenic, 1999; Baeza-Yates and Ribeiro-Neto, 1999)

The problem accomplished in this work is something different: the goal is the identification of the place where the information is located, but there are no documents to analyze. Anyway we will do a brief revision of the IRS published, because they are the base for the IRS proposed by us to implement into a GDSS.

In classical IR, the documents, stored in a collection, are analyzed to obtain a contents representation. This process is called "indexation". The IRS extracts index terms from the text of each document to build a document representation consisting of a set of weighted terms.

A user has access to the system sending a query. Starting from this, the system identifies the entire document relevant for this query. The IRS extracts index terms from the text of natural language query to build a query representation consisting also of a set of terms. Document representations are then matched to the query representation. Documents are ranked according to how well their representation matches the query representation (Crestani, 2000).

There are several models of information retrieval, and the result of the search depends on these models (Crestani ans Pasi, 1999).

In the *Boolean Model*, documents are represented as sets of index terms. A query is constructed by the user choosing the terms and connecting them by Boolean operators (e.g., AND, OR, NOT). A document is considered relevant and retrieved if it satisfies the logical formula represented by the query.

In the *Vector Space Model* each document is represented by a set of index terms with an associated numerical value, which represents the degree of significance of the term with respect to the content of the document. A query is transformed into a vector of keywords. Then a similarity measure between document and query representation is used to evaluate the document's relevance regarding the query.

In the *Probabilistic Model* documents are ranked by their evaluated probability of relevance for a user's need of information.

The Vector Space model and the Probabilistic model are the most important models in IR. Both of them require automatically extracting terms from documents and queries or manually assigning terms to them. This way of representing documents and queries has a very serious side effect known as term mismatch problem: a relevant document will not be retrieved in response to a query if the document and query representations do not share at least one term. That is so because users and authors of indexed documents often

use different terms to refer to the same concepts (Crestani, 2000). To solve this problem a number of approaches have been proposed. They can be classified in three groups:

Dimensionality Reduction: These approaches consist of reducing the number of possible ways in which a concept can be expressed, that is, reducing the vocabulary used to represent concepts.

Query Expansion: It consists of automatically or semi-automatically adding terms to the query by selecting from the term space those that are the most similar to the ones originally used by the user. Terms added to the query should be weighted in such a way that their importance in the context of the query will not modify the original concept expressed by the user.

Logical Imaging: It uses term-term semantic similarity to direct the transfer of indexing weights at retrieval time from the terms non present in the document to terms that are present.

The aforementioned IR systems allow us to solve the problem of looking for documents published in big collections as INTERNET, according to the necessities that a user has formulated through a query in natural language.

The analysis of the text contents of these documents allows for the extraction of a set of terms that represent them. The importance of these terms regarding the description of the document (weight) is deduced by starting from the number of times that each term is repeated in the text and its position inside the document (title, abstract, etc.). Also, the similarity between terms can be obtained comparing documents belonging to the same context.

In a GDSS, there exists a finite set of domains, each one managing some specific information of the enterprise.

A user, in the presence of a necessity of information, carries out a query in natural language, which should be answered from a domain that he/she ignores.

The main difference lies on the strategy we will use to represent the domains, since in our problem there are no texts to analyze. Therefore, it is necessary to have the intervention of an expert who defines the initial set of keywords that will identify the domain, and the relative importance of each of them to define its context.

For this purpose, we propose to use an adaptation of the Analytical Hierarchy Process (AHP) to support the expert in the task of defining the weight of the indexing terms.

4.1. Domain Representation

The strategy we propose for obtaining the initial representation of each domain comprises the following five steps:

1. Define a set of keywords that allows identifying the information generated and manipulated in the domain.
2. Establish a weight associated to each term, which defines its importance with respect to the description of the domain content.
3. Make a semantic expansion of each keyword defined in (1).

4. Establish a similarity degree of each term determined in (3) with respect to the expanded keyword.

5. Establish a weight associated to each term of the expanded set of keywords.

The first four steps require the intervention of an expert. Each one of these items will be developed in the following way.

4.1.1. Keywords representing the domain. An expert of each domain *d* must identify a set of terms K_d that are usually employed in the domain , which allows us to identify the manipulated information.

$K_d = \{t_i / t_i$ is a keyword defined to represent the domain $d\}$

It is important that these terms $t_i \in K_d$ do not represent related concepts. This allows the simplification of the posterior task of evaluating the relative importance of each term for the domain.

Let us consider, for example, the domain "forecasting" of a particular GDSS. An expert has selected as keywords the terms {*prediction, time, model, graphic*}, but not {*forecast, prediction, prognosticated*}, because the last ones are related with the same concept.

4.1.2. The indexing weights assigned to each keyword of the domain. To carry out this task we propose the use of the Analytical Hierarchy Process (AHP).

AHP is a decision tool for dealing with complex, unstructured and multiattribute decisions, developed by Thomas L. Saaty (1980). The methodology is based on the principle that for making a decision, the experience and the knowledge of people are as valuable as the used data. AHP offers a structure to include a classification of trials and approaches in an intuitive and consistent way.

AHP uses pairwise comparisons of attributes in the decision making process. This process is called the intensity of the reasons affecting the decision. This comparison technique is useful in this paper to find the weight factor of each indexing terms of the domain. The expert assigns an importance intensity number from Table 1 that represents, in a 1-9 scale, the relative importance of each term $t_i \in K_d$ with respect to other term $t_j \in K_d$ to describe the domain.

If *n* terms have been selected by the expert to describe the domain, an $n \times n$ matrix A is derived.

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}$$

where a_{ij} is the intensity importance of the term t_j over the term t_i.

If the expert criterions are consistent: $a_{ij} = 1$ iff $t_i = t_j$; $a_{ij} = 1 / a_{ji}$; and $a_{ij} = a_{ik}\, a_{kj}$. Saaty has suggested a strategy to evaluate the consistency of the pairwise matrix and an acceptability criterion that we will not develop in this paper.

To get a raking of priorities from a pairwise matrix A, according to Saaty, the best approach is the eigenvector solution. A short computational way to obtain the eigenvector suggested by him is the following:

a) Raise the pairwise matrix to powers that are successively squared each time.

b) Sum the n elements of each row.

c) Normalize the result by dividing each row sum by the row totals. The result is the eigenvector.

d) Repeat this process until the eigenvector solution does not change from the previous iteration.

The elements of the eigenvector $\boldsymbol{w_d}$ found when this iterative process has finished represent the indexing weights w_i^d assigned to each keyword t_i of the domain d.

Table 1: Intensity importance of factors in the pairwise comparison process

Intensity of importance on an absolute scale	***Definition***	***Explanation***
1	Equal importance	Two terms contribute equally to the objective.
3	Moderate importance of one over the other	Experience and judgement weakly favour one over the other.
5	Essential or strong importance	Experience and judgement strongly favour one over the other.
7	Very strong importance	A term is strongly favouring and its dominance demonstrated in practice.
9	Extreme importance	The evidence favouring one term over the other is of the highest possible order of affirmation.
2,4,6,8	Intermediate values between the two adjacent judgements, when compromise is required.	

4.1.3. Semantic expansion. Not everyone uses the same terms to express the same concept. For that reason, a relevant domain could not match with a query because they have no keywords in common. That is an obstacle to produce a good ranking of domain associated to the query.

To take into account this problem, we suggest to make a semantic expansion of the keywords defined by the expert. That is, to associate a set of words T^i related to its concept to each keyword $t_i \in K_d$.

$T^i = \{t_j / t_j$ is a word related to the t_i concept$\}$ includes the term t_i.

This task must be also done interactively by the expert. Considering the same example of the forecasting domain, the terms *forecast* and *prognosticated* should be included in the semantic expansion of the keyword *prediction.*

4.1.4. Similarity Measure. Let be the keyword $t_i \in K_d$, which has been expanded by a set of terms T^i. We need a measure of similarity that enables us to evaluate a real value for each term $t_j \in T^i$. This value estimates how semantically close the term t_j is to t_i.

Most similarity measures used in IR attempt to estimate the semantic similarity between terms by looking at their pattern of occurrence in documents. Two terms are considered semantically similar if they tend to co-occur in the same context. That is not applicable to the problem addressed in this paper. Again AHP will be a useful tool to establish the similarity between terms.

This similarity measure should have the following properties (Crestani, 2000):

$$0 \le \mathrm{Sim}(t_i, t_j) \le 1 \quad \forall\, t_j \in T^i,\ t_i \in K_d$$

$\mathrm{Sim}(t_i, t_j) = 1$ iff $t_j = t_i$

$\mathrm{Sim}(t_i, t_j) \approx 1$ if t_j and t_i can be used to express the same concepts.

$\mathrm{Sim}(t_i, t_j) \approx 0$ if t_j and t_i are not semantically close.

Then we need to compute the similarity vector $\boldsymbol{s}^i = \{\mathrm{Sim}(t_i, t_j)\}$ with m elements (where m is the number of terms that expand the keyword t_i , including t_i).

The AHP allows us to compute a vector $\boldsymbol{v}^i = \{v_{ij}\}$ that ranks the terms $t_j \in T^i$ according to their capacity for representing the same concept as t_i.

The first stage to derive this vector consists of defining an $m \times m$ pairwise matrix P^i, in which the element p^i_{jk} is the importance intensity number, in a 1-9 scale (the same as Table 1), which represents the relative importance of each term t_j with respect to other term t_k to describe the same concept as t_i.

Then, the eigenvector $\boldsymbol{v}^i$, associated to the matrix P^i, is computed following the strategy above described to find the weight factor of keywords of the domain.

Finally, starting the eigenvector $\boldsymbol{v}^i$, we can obtain the similarity vector $\boldsymbol{s}^i$ dividing each element v_{ij} by v_{ii}:

$$Sim(t_i, t_j) = \frac{v_{ij}}{v_{ii}} \qquad \forall\ j \in T^i\ ,\quad i \in K_d$$

The similarity measure obtained in this way verifies the properties previously required.

4.1.5. Weight assigned to each term of the extended domain representation. Since the semantic expansion has been done and the similarity measures computed, it is

necessary to obtain the weights we_k^d associated to each term t_k of the expanded set of keywords $EK_d = \{t_k / t_k \in T^j$ and $t_j \in K_d\}$. For that purpose, we propose to compute:

$$we_k^d = Sim(t_k, t_j)\, w_j^d \qquad \forall\ k \in EK_d\ ,\ j \in K_d$$

The set of terms EK_d and their associated weights we_k^d constitute the initial elements of the Router KB. According to the GDSS work, a learning mechanism, not included in this paper, will up date this KB.

4.2. Query Representation

When a user from a domain d' belonging to the GDSS requires the same information, he/she must build a query that will be transported to the Router by a collecting agent. This query will have a two-part structure:

The TITLE: it consists of an explicative phrase with an upper bound of the number of strings.

The EXPLANATION: in this section the user explains in detail which information he/she requires. No bound over the extension is imposed to this part.

To analyze the possible domains that can give an answer to the query, the Router Agent only takes into account the field TITLE. The other field will be only used by the domain that should answer the query.

The set of keywords K_q that represent the query q must be identified. For this purpose, the system breaks the Title into its constituent works. That is, the keywords $t_q \in K_q$ are the terms that remain after the stop words (i.e. articles, propositions, auxiliary verbs, etc.) are eliminated.

4.3. Domain Information Retrieval Process

Since the KB of the Router has the initial information about each domain of the GDSS, that is, the indexing weight we_i^d of each keyword $t_i \in EK_d$, the Router can be able to identify the set of domains associated to each query that comes from the user.

The domain information retrieval process accomplished by the Router is schematically showed in Figure 3.

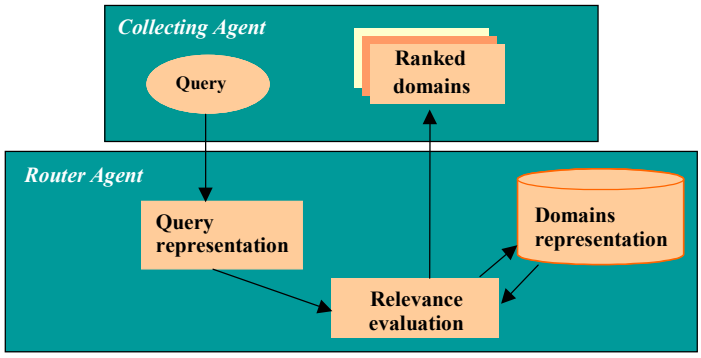

Figure 3. Schematic representation of the domain information retrieval process

When a query comes to the Router, the system determines a query representation in the way explained in the previous paragraph. Then, it uses some relevance evaluation algorithm to compare the domains representation with the query representation. Finally, it lists the domains that best match the query.

In classic IR, the Retrieval Status Value (RSV) of documents is used to estimate the relevance of a document with respect to a query. Many models have been proposed to compute the RSV. The most used are those that consider the frequency of the terms representing the query *q* in the contents of the document *d*. We can not use those models because we have domains represented by a set of keywords. Then we propose to use an adaptation of the coordination level matching model, which simply counts the number of terms that are present in both document and query (Crestani, 2000).

$$RSV(d,q) = \sum_{t_i \in K_q} we_i^d$$

where RSV(d,q) is the retrieval status value of the domain *d* with respect to the query *q*.

The RSV(d,q) is the sum of the indexing weight of the total keywords of the query, which are included in the expanded representation of the domain *d*. If any term does not belong to the expanded representation of the domain, its weight is null.

The list of the RSV(d,q) for each domain is the information that the Router Agent transfers to the collecting agent for deciding the route to follow to get the answer.

5. CONCLUSIONS

In a previous work we have describe a GDSS based in the use of mobile agents (Caliusco et al., 2000). This system is able to support non-habitual decision processes. The Router Agent is the intelligent component that allows achieving an efficient operation of the query mechanism, avoiding the overload of the system by mobile agents, and the constant interruption to the domains with queries that they are not probably able to answer. This agent is responsible of giving a route to the collecting agents, which must visit the system domains searching for an answer to queries formulated by a user. In this paper we have presented a strategy to obtain a ranked list of domains able to answer these queries.

To perform this domain information retrieval process, the Router Agent must have a KB and a mechanism of matching the query contents to the domain contents. We have proposed a structure to represent the information managed by each domain. The strategy to obtain the initial data to be stored in the Router KB requires the intervention of a domain's expert. The AHP method is used to support the task of evaluating the relative importance among the keywords.

We proposed an interactive mechanism to obtain an adequate initial representation of the domain contents into the KB. But this initial information must be increased according to the dynamic query mechanism work. To this end, the knowledge component of the Router Agent must also to account with a learning mechanism. That is been studied now and will be presented in a future work.

REFERENCES

Baeza-Yates, R. and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Wokingham, UK, 1999.

Caliusco, M., L. Cabral, H. Nissio, P. Villareal, M. Galli, M. Taverna and O. Chiotti. *Distributed Architecture to Support Dynamic Decision Processes*, Proceeding XXVI Conferencia Latinoamericana de Informática , Mexico, septiembre, 2000.

Chess, D., B. Grosof, C. Harrison, D. Levine, C. Parris and G. Tsudik, *Itinerant Agents for Mobile Computing,* in Michael Huhns' & Munidar Singh's book, *Reading in Agents, 1998*.

Cretani, F., *Exploiting the Similarity of Non-Matching Terms at Retrieval Time,* Information Retrieval, **2**, 25-45, 2000.

Crestani, F. and G. Pasi. Soft Information Retrieval: Applications of Fuzzy Set Theory and Neural Networks. In: N. Kasabov and R. Kozma, editors, Neuro-Fuzzy Techniques for Intelligent Information Systems, pages 287-315, Physica Verlag (Springer Verlag), Heidelberg, Germany, 1999.

Mladenic, D. *Text-Learning and Related Intelligent Agents: A Survey.* IEEE Intelligent Systems, July/August, 44-54, 1999.

Nagel, J., P.Rossi, A.Toffolo, P.Villarreal, and O. Chiotti *Distributed Architecture of a Global DSS Generator for Industrial Companies*, Proceeding 25th International Conference on Computers and Industrial Engineering, Lousiana, USA, 1999.

Rus,D., R. Gray and D. Kots, Department of Computer Science, Dartmouth College, *Transportable Information Agents,* in Michael Huhns & Munidar Singh book, *Reading in Agents,* 1998

Saaty, T. L. *The Analytical Hierarchy Process.*New York: McGraw-Hill, 1980.

White, J.E., General Magic Inc., 1997 , *Mobile Agents*, in Jeffrey Bradshaw's book *Software Agents,* AAAI Press/The MIT Press 1997.

Acknowledgments

We are grateful to acknowledge financial support from "Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)" and "Universidad Tecnológica Nacional (UTN), Facultad Regional Santa Fe".