KNOWLEDGE COMPONENT OF A MULTIAGENT DISTRIBUTED DECISION SUPPORT SYSTEM

Georgina Stegmayer - María Laura Caliusco
GIDSATD-UTN – Facultad Regional Santa Fe – Argentina

{gstegmay, mcaliusc}@frsf.utn.edu.ar

Omar Chiotti - María Rosa Galli
GIDSATD-UTN – Facultad Regional Santa Fe – Argentina
INGAR – CONICET – SANTA FE – Argentina
{chiotti, mrgalli}@ceride.gov.ar

ABSTRACT

We have developed a distributed DSS capable to working in a dynamic way. That is, when a domain of an organization needs a new kind of information, the system looks for this information. This system is based on the usage of mobile agents, which receive the user's queries and visit the appropriate DSS domains to gather the required information. The system itself must analyze where this information can be generated. To make this decision there is an intelligent agent (the Router) with a knowledge base (KB) where the information managed by each domain is represented. In this work, we present a strategy to obtain the initial data to be stored in the KB, a knowledge retrieval mechanism from the KB, and a learning mechanism so that the KB and the DSS operation can be continually improved. The proposed learning process is an interpretative case-based reasoning, which uses a set of rules to analyze the results of the information retrieval process and modifies the content of the

router KB. Some examples are presented to illustrate the learning mechanism.

Keywords: Dynamic DSS, Case-Based Reasoning, Learning, Intelligent Agent, Knowledge Retrieval.

1. INTRODUCTION

Currently, most organizations are horizontally structured, delegating decisions to their different sections. For example, in an industrial organization, enterprise domains such as Product Design, Planning, Control, Scheduling, Forecasting and Sales, are required to cooperate so as to achieve their common goal (see Figure 1). These domains constitute different decision points and are generally located on different functional units geographically disperse, use specific models and techniques to each decision type, and need to share knowledge and information. Therefore, a Decision Support System (DSS) that involves the whole organization should be designed as a set of geographically distributed subsystems (Domains),

Figure 1. Decision Support System Domains

operating with the smallest possible level of dependence, which will be called distributed Decision Support System.

Nagel et al. (1999) have designed a distributed architecture for that distributed DSS based on the *federation* concept [14]. This is a contract-based mechanism where each domain must specify both its responsibility for publishing the information it generates and its subscription to the information generated by other domains. This is an appropriate mechanism for situations in which the information different domains should share can be predicted on design time. But the enterprise domains change in time and they depend on human behavior, which sometimes may be unpredictable. In these cases, the system must be able to search the ad-hoc required information analyzing where it is available or can be generated. The mobile agent technology appears as the most suitable one for this case ([18], [9]).

Caliusco et al. (2000) proposed to develop a distributed architecture for a DSS with two mechanisms of interaction among domains: a contract-based mechanism, which supports the static (predictable on design time) decision

processes, and a mobile agent-based mechanism to support dynamic (non predictable on design time) decision processes [3].

This dynamic distributed DSS should be able to interpret a decision-maker's query (for example, "where can I find the graphics for the last year product sales?") and to distinguish the domain that is able to provide the information. In the current version of this system, only a domain user answers the information requirement. The possibility of this system gathering information from a distributed data base will be included in future works. The multiagent system proposed in [3] has a static agent that fulfills these functions: the Router agent. This agent is the intelligent component that allows achieving an efficient operation of the query mechanism. This prevents the system from being overloaded by mobile agents that look for data independently, and avoids the constant interruption to the domains with queries they might not be able to answer. These properties have not been taken into account in any other multiagent systems published up to now.

Several aspects must be considered when building a multi-agent DSS: agent organization, knowledge of the agents, communication language and protocol, query processing and learning capabilities [10]. To address social aspects of the agents we used software engineering techniques, like requirement analysis, use cases and interaction diagrams. Furthermore, we used knowledge engineering methodologies to

solve knowledge acquisition, modeling and reuse problems. Specifically, strategies of information retrieval (IR) and cases based reasoning (CBR) have been adjusted to the special features of the considered problem ([1], [11]).

Stegmayer et al. (2000) have proposed a knowledge retrieval mechanism for the Router agent, adjusting the existing methodologies for information retrieval from big document collections [21]. It is based on a vectorial representation of the available information in each domain. An expert, who in this context is defined as a person that knows the knowledge managed at the domain, should originally dictate the keywords for this representation. For that purpose they proposed a weighting strategy for the keywords that feed the knowledge base of the Router agent, where this domain knowledge will be represented.

The initial contents of this KB could be insufficient or have certain incompatibilities as a consequence of being loaded by experts in each particular domain and probably with partial knowledge of all the other domains. So, the Router agent should have a learning mechanism to improve the domains representation in the KB according to the dynamic DSS work.

In this paper we present the learning mechanism for the Router agent using case-based reasoning, which allows maintaining and updating the KB of the Router agent.

The remainder of this paper is structured as follows. In Section 2, we present a

brief discussion on related works. Section 3 presents the components that integrate the multiagent system and their interaction scheme. This section goal is to show the environment where the Router agent works. The role of the router agent and its knowledge recovery strategy to establish priorities when traveling to look for the answers are described in Sections 4 and 5. The learning mechanism used to update the Router agent KB is detailed in Section 6, together with some simple examples illustrating the procedure. Finally, some conclusions and future works are presented in Section 7.

2. RELATED WORKS

Several multiagent systems have been developed to integrate the activities concerning the modeling of production processes, especially in manufacturing enterprises [20]. These systems make it possible to dynamically integrate new subsystems or to remove them, without stopping or reinitializing the working environment. But in these systems the communication structure has been designed on the basis of predictable scenarios. That is, precise knowledge exists as regards which sources provide each information type.

To detect which domain can answer to non-habitual queries, that is to say, to identify new information sources, it is necessary to know the topics that each area manages and to identify what topic is referred to in the user's query. A wide range of researchers are involved in the intensive development of methods for

using machine learning techniques on text databases, which are called text learning and combine research in machine learning with information retrieval [16].

Burke et al. (1995) proposed the FAQFinder agent. This agent accesses distributed text information sources and helps users to find answers to their questions in databases such as FAQ files. It matches questions from relevant FAQ files against user questions and returns the five best matching questions together with their answers [2].

Dale and DeRoure (1997) presented an architecture that supports mobile agents, which carry out distributed information management tasks to help the user to integrate and manage their data through Internet protocols and between Internet systems such as WWW servers [6]. But in that work it is considered that whenever the information is required, mobile agents will visit all domains to gather the data from their databases. The possibility of access to different users of the network is not considered.

The mentioned works look for information that should be stored; they do not contact users that can provide answers according to the knowledge they possess. Krulwich and Burkey (1996), presented the ContactFinder agent [12]. This agent searches the bulletin board looking for contact people and their topic areas, storing the found information in its database. Then, it searches new messages looking for questions, extracting the topic area of the found question and finding a contact person in its database for that topic area.

Vivacqua (1999) analyzed the problem of finding people who can help solving problems by using software agents to assist the search for expertise and mediate the information exchange process [22]. The agent, called Expert Finder, does not provide answers to problems; it guides people to contact other people who can do it. To create the user's profile, the system accesses to the user's files and it analyzes their contents. This system has been applied to contact experts in Java programming.

Although our work consists of getting users in contact with the appropriate knowledge, its object is to determine the profile of each domain-knowledge keeping each user's privacy, especially for security reasons.

3. MULTIAGENT ARCHITECTURE TO SUPPORT DYNAMIC DECISION PROCESSES

Considering that the information used by company departments (called Domains from now on) changes in time because of the enterprise dynamics, the DSS has to be flexible in order to satisfy the information needs in a dynamic environment. For that purpose, it was necessary to design a searching mechanism that analyzes where the information is available or can be generated. Firstly, we describe a searching mechanism architecture based on an agent approach, and then, we discuss some considerations about the architecture.

Figure 2. Mobile Agents to support the dynamic distributed decision process

3.1 Multi-agent architecture

To develop a searching mechanism we have designed a multi-agent architecture [3] (see Figure 2) based on modular organization, in which each component performs a specific task. These components are mobile and static agents. On the one hand, mobile agents search the domain where the information needed for a domain user is available or can be generated. On the other hand, static agents act as a representative of one domain. Furthermore, there is a static agent that centralizes the searching mechanism. The centralized approach has been considered to avoid system overload by agents, that is, the domains are not constantly interrupted with many queries they might not be able to answer. Following, we describe the main features of each agent.

1. *Collecting Agents (CAs)*: The role of CAs is to receive the user's query, to

visit the domains that could provide the answer and return to the original domain to give the obtained information. These agents are mobile and hybrid (reactive and proactive). First of all, they have to migrate to different domains in order to obtain the information requested by a user. Second, they have to be reactive when receiving the user's query and they have to plan their search.

2. *Domain Representative Agent (DRA)*: The representative agent of a domain acts as a mediator between a domain and the CAs belonging either to its own or to another domain. This is a static and reactive agent whose function consists of sending the users queries to the CAs, so that the latter can search for the information. It also transmits the answers to previous queries of the incoming agents and represents the domain when a CA that belongs to another

domain arrives asking for information.

3. *Router Agent (RA)*: It receives the queries of the incoming CAs and, according to the information in its KB, gives the CAs a route listing the domains that can provide them with an answer. Furthermore, assuming that it must deal with mobile agents, it must have the capability of providing such agents with a safe execution environment. This is a static and hybrid agent.

Furthermore, the architecture is composed by:

4. *Collecting Agents Server (CAS)*: It is the mediator between the CAs and the system resources. It must provide protection for the local system against the incoming agents so as to avoid the harmful action of any "spy" agent. Every interaction between an external program and the CAs takes place through the CAS. This component is needed to implement the mobile agent architecture.
5. *Gatekeeper (GK)*: It is an abstraction of the set of programs that must be always up and running; no matter whether the domain they belong to is active or not. The *Representative Agent*, the *Collecting Agent Server* and a *Knowledge Base* are the three components that constitute the Gatekeeper of each domain.

The proposed multi-agent architecture is an asynchronous system, in which a set of agents acts autonomously to one another.

When a decision-maker needs specific information, he/she expresses the information he/she would like to receive with the aid of an assistant. Then, the assistant passes it to the DRA, which will change the query into a format accepted by the mobile agents and will send it to the CAS. The CAS gives the query to a domain CA and it goes to a special agent server: the RA. It receives the CA that has just left its domain, reads the formulated query and according to the information kept in its KB, it gives the CA a list of domains that would provide the needed information. Then, the CA visits these domains with its query. Once it has found the answer, and before coming back to the original domain, it visits the RA again to inform it of the obtained results, allowing it to have new information to update its KB for future queries. Finally, the CA goes back to its domain, delivers the information and destroys itself.

3.2 Mobile Agent Mechanism

Agent based systems naturally support the representation of information-driven business processes. Agent based systems also offer high levels of flexibility and robustness in dynamic or unpredictable environments by virtue of their intrinsic autonomy. For example, agents can be provided with objectives and strategies; abstractions not easily supported by classical object models. Agent key tenets are abstraction, interoperability and indirect management and they represent the next evolution in integration since they can interoperate at many different

levels (with the user desktop, with information and legacy systems in both local and distributed manner, and with other agent-based systems) [6]. The characteristics of mobility in agents allow the metaphor to address aspects of distribution, especially those that involve large networks of machines that are spread over wide areas. To connect different machines, we would have used a traditional client-server architecture that uses remote procedure call (RPC). But, in contrast to this mechanism, mobile agents transport data and procedure to the client, which can lead to less traffic [8].

4. THE KNOWLEDGE COMPONENT OF THE ROUTER AGENT

The RA is responsible for making the information sources identification an efficient process. Its structure has been defined with a two-component model [21]:

- a) The *Communication Component* has the function of communicating with the CAs.
- b) The *Knowledge Component* is responsible for two tasks:
 - i) Interpreting the query transported by the CAs, and inferring a probable set of domains able to answer the consult. We will call this task knowledge retrieval into the distributed DSS.
 - ii) Maintaining and updating its KB by means of a learning process.

In this paper we will concentrate our attention on the Knowledge Component. First of all, we briefly describe the

information retrieval mechanism, which is explained in detail in [21] and then, we develop the learning mechanism, which allows updating the router KB starting from the experience obtained with the system operation.

5. KNOWLEDGE RETRIEVAL INTO A DISTRIBUTED DSS

An Information Retrieval (IR) System is a tool that allows representing and storing information to be automatically retrieved when required. There are several models of IR, and the result of the search depends on these models ([1],[5]). The most developed Information Retrieval area is the one concerned with the selection of documents published in big collections as INTERNET. The analysis of the text contents of these documents allows 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 [4].

The problem accomplished in this work is something different. Neither texts from a documental base nor data from a database should be recovered. The goal is to find the domain from which a certain user can answer a specific query. Then, a knowledge base is required, which should contain information about the type of knowledge managed in each enterprise domain, so that once a query is formulated, it can determine which part of the system could provide the answer. We call this process "Knowledge

Retrieval” (KR). Therefore, an expert is needed to define the initial set of keywords that will identify the domain knowledge and the relative importance of each keyword to define its context. For this purpose, we proposed using the Analytical Hierarchy Process (AHP) to support the expert in defining the weight of the indexing terms. Next, we present a brief description of the proposed strategy for the domains representation in the router KB and the mechanism for the knowledge retrieval.

5.1. Domain Representation

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

5.1.1. 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 identifying the manipulated information.

$K_d = \{t_i / t_i \text{ is a keyword that represents domain } d\}$

5.1.2. The Indexing Weights Assigned to each Keyword of the Domain.

Since keywords $t_i \in K_d$ have been identified, it is necessary to assign the indexing weights w_i^d for each them. To carry out this task we have proposed to use the AHP technique [19]. The domain expert does not know in advance the relative importance value of each keyword to explain the information managed by this domain. But he/she can estimate a relative pair-wise comparison

a_{ij} as an approximation to the ratio of w_i^d/w_j^d .

Each keyword t_i is compared to all other terms $t_j \in K_d$. Comparisons are made by firstly asking “Which of the two is more important to explain the domain content?” and secondly, “To what extent?” The strength of importance is expressed on a 1-9 ratio scale. The value 1 indicates that the both terms equally contribute to describe the domain content, while the value 9 indicates that one term is much more important than the other. For example, to represent the domain *Sales*, if we do the pair-wise comparison between the keywords *product* and *graphic*, we can assign the intensity importance 3 because the term *product* is moderately more important than *graphic*; and if we compare the keywords *time* and *customer*, as *time* is very strong less important than *customer* we can assign the intensity importance 1/7.

If n terms have been proposed to describe the domain, an $n \times n$ matrix $A=[a_{ij}]$ is derived, where a_{ij} is the intensity importance of the term t_j over the term t_i . If the expert criteria is consistent $a_{ij}=1$ if and only if $t_i = t_j$; $a_{ij} = 1/a_{ji}$ and $a_{ij} = a_{ik} a_{kj}$. Saaty in [19] describes a strategy to evaluate the consistency of the pair-wise matrix and an acceptability criterion. If not, the process must be done again until the matrix gets consistent. Here, we do not take into account the different

possible matrixes generated from different experts of the same domain. We consider only one matrix.

	<i>sales</i>	<i>product</i>	<i>item</i>	<i>time</i>	<i>customer</i>	<i>graphic</i>	<i>profits</i>	<i>estimate</i>
<i>sales</i>	1	2	7	7	2	4	3	9
<i>product</i>	1/2	1	6	6	1	3	3	9
<i>item</i>	1/7	1/6	1	1	1/7	1/2	1/6	4
<i>time</i>	1/7	1/6	1	1	1/7	1/2	1/6	4
<i>customer</i>	1/2	1	7	7	1	3	3	9
<i>graphic</i>	1/4	1/3	2	2	1/3	1	1/4	7
<i>profits</i>	1/3	1/3	6	6	1/3	4	1	7
<i>estimate</i>	1/9	1/9	1/4	1/4	1/9	1/7	1/7	1

Consistency: 0,05

Table 1. Sales domain pair-wise matrix

To get a ranking of importance weights from a pair-wise matrix A , the best approach is the eigenvector solution, following this procedure [19]:

- Raising the pair-wise matrix to powers that are successively squared each time.
- Summing the n element of each row.
- Normalizing by dividing each row sum by the row totals. The result is the eigenvector solution.
- Repeating this process until the eigenvector does not change from the previous iteration.

When this iterative process finishes, the eigenvector is found. The indexing weights w_i^d assigned to each keyword t_i of the domain d are obtained by dividing each element of the eigenvector by the element of maximum value. This normalization allows comparing different domains.

5.1.2.1. Example 1

To illustrate this procedure, this example

shows the representation of the “*Sales*” domain belonging to an industrial organization. Eight keywords have been identified for this domain. Using the AHP technique, the pair-wise matrix shown in Table 1 has been computed. These values have been accepted because the consistency index is low (less than 0.1 as suggested by Saaty [19]). Table 2 shows the indexing weights corresponding to each keyword.

Keyword	weight
<i>Sales</i>	1,00
<i>Product</i>	0,71
<i>Item</i>	0,12
<i>Time</i>	0,12
<i>Customer</i>	0,74
<i>Graphic</i>	0,23
<i>Profits</i>	0,48
<i>Estimate</i>	0,06

Table 2. Keywords indexing weights in Sales domain.

5.1.3. Semantic Expansion.

Not everybody uses the same terms to refer to the same concepts. That is an obstacle to produce a good ranking of

domains associated to a query. A relevant domain might not match a query because their representations have no keywords in common. To take this problem into account, we suggest making a semantic expansion of the keywords defined by the expert. That is, to associate each keyword $t_i \in K_d$, to a set of words T^i related to its concept.

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

For example, to the keyword *sales* in the *Sales* domain, the terms *sales*, *salesman*, *sell*, *saleswoman*, *salesroom*, could be a semantic expansion.

The expert must also do this task interactively.

5.1.4. 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 . This similarity measure should satisfy [4]:

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

Then we need to compute the similarity vector $s^i = \{\text{Sim}(t_i, t_j)\}$ 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 $v^i = \{v_{ij}\}$ that ranks the terms $t_j \in T^i$ by their capacity for representing the same concept as t_i .

The first stage to derive this vector consists of defining an $m \times m$ pair-wise

matrix P^i , in which the element p_{jk}^i is the importance intensity, which represents the relative importance of each term t_j in respect to another term t_k to describe the same concept as t_i .

Starting from eigenvector v^i , associated to the matrix P^i , we can obtain the elements of the similarity vector s^i :

$$\text{Sim}(t_i, t_j) = \frac{v_j}{v_{\#}} \quad \forall j \in T^i, i \in K_d \quad (2)$$

In Table 3 and 4 the pair-wise P^{sales} and the similarity vector, respectively, are showed.

5.1.5. Weight Assigned to each Term of the Extended Domain Representation.

Since the semantic expansion has been done and the similarity measures have been computed, it is necessary to obtain the weights we_k^d associated to each term t_k of the expanded set of keywords

$$EK_d = \{t_k / t_k \in T^i \text{ and } t_j \in K_d\}.$$

To this end, we propose to compute:

$$we_k^d = \text{Sim}(t_k, t_j) w_j^d \quad (3)$$

$$\forall k \in EK_d, j \in K_d$$

The set of terms EK_d and their associated weights we_k^d define the taxonomy of each domain at the KB and constitute the initial elements of the router KB. (In Table 5 the Sales domain taxonomy is showed). According to the distributed DSS work, a learning mechanism will update this KB.

This two-stage (keywords definition and semantic expansion) representation strategy of each domain makes the expert task easier, since he/she should work

	<i>sales</i>	<i>salesman</i>	<i>sell</i>	<i>saleswoman</i>	<i>salesroom</i>
<i>sales</i>	1	2	3	2	5
<i>salesman</i>	1/2	1	2	1	3
<i>sell</i>	1/3	1/2	1	1/2	3
<i>saleswoman</i>	1/2	1	2	1	3
<i>salesroom</i>	1/5	1/3	1/3	1/3	1

Consistency: 0,01

Table 3. Pair-wise matrix for *sales* semantic expansion.

keyword	Similarity
<i>sales</i>	1
<i>salesman</i>	0,55
<i>sell</i>	0,34
<i>saleswoman</i>	0,55
<i>salesroom</i>	0,17

Table 4. Similarity vector for *sales* semantic expansion.

with small groups of words to obtain an expanded set of keyword-weight couples.

5.2. Query Representation

When a user from a domain d belonging to the distributed DSS requires some information, he/she must build a query that will be taken to the RA by a CA. This query will have a two-part structure:

- An explanatory phrase with an upper bound number of words (Title) in natural language.
- A detailed explanation about the information the user requires (only searched in the Domain that answers the query).

The set of keywords K_q that represents the query q must be identified. For this

purpose, the system breaks the Title into its constituent words. That is, the keywords $t_q \in K_q$ are the terms that remain after stop-words (i.e. articles, prepositions, auxiliary verbs, etc.) are filtered.

5.3. Knowledge Retrieval Process

When a query comes to the RA, the system determines a query representation as explained in the previous paragraph. Then, it uses some relevance evaluation algorithms to compare the domains representation with the query representation. Finally, it lists the domains that best match the query (Figure 3).

The retrieval status value of the domain d in respect to the query q , $RSV(d,q)$, is used to estimate the relevance of a domain in respect to a query.

$$RSV(d,q) = \sum_{t_i \in K_q} w e_i^d \log \frac{N}{n_i} \quad (4)$$

where N is the total number of domains in the system and n_i is the number of domains that have the keyword t_i in their

Figure 3. Schematic representation of the knowledge retrieval process

taxonomy. The factor $(\log N/n_i)$ allows us to consider that the words that appear in many domains are not useful to distinguish among relevant and non-relevant domains.

The list of the $RSV(d,q)$ for each domain is the information the RA transfers to the CA to decide the sequence of domains to be visited in order to get the answer. This is called the Knowledge Retrieval (KR) process.

6. LEARNING PROCESS

In the router KB there is taxonomy for every enterprise domain, which is built on the basis of the subjective knowledge of the domain experts. This knowledge could become incomplete or inaccurate as a consequence of the dynamic DSS work. The RA should realize that situation and be able to keep the KB updated and corrected.

The KR process extracts information from the KB to calculate a ranking of domains able to answer a query.

Depending on the knowledge kept in the KB, the KR process will drive the query to the correct sources of answers or not.

The RA should have a mechanism for modifying its KB (if necessary) to reflect the dynamic behavior of the system, and indirectly improving the KR process to avoid mistakes in the search for information done by the CAs. To do so, this mechanism should receive some feedback about how well the KR process is working and should use a learning approach to keep the knowledge updated (Figure 4).

Figure 4 - Learning process for the Router Agent

Once the CA has both finished its journey and obtained an answer to its query, it should inform the RA about the success of its search before returning to the original domain. In other words, in what degree the route proposed by the RA has been the correct one. The learning process begins with this.

The explicit request for reusing knowledge and experience calls for the application of Case Based Reasoning

(CBR) [11]. It is a technique that considers learning as a process of accumulating experiences (named cases) in its memory and then remembering a small set of them to solve or interpret a new situation. This two different approaches lead to different types of CBR: problem solving and interpretive, respectively [14].

For the RA learning process we have defined an interpretative type Case Based Reasoning strategy because it is the one that best applies to our learning process. In the interpretive reasoning, cases are remembered to understand situations and to justify the election of an interpretation to begin reasoning with. Interpretive Process takes a situation as input and the output is a classification of it, with a justification of the solution [15]. The RA should select some cases and interpret them, reasoning what has happened in the cases and learning from their solutions.

It is necessary to highlight that in this paper CBR is used to update a KB and not to directly drive an information retrieval system [7].

According to this style of CBR, we propose using a Memory of Cases to store the results of searches carried out by the CAs that arrive to the RA. Using rules, the cases will be retrieved and classified into different categories of cases (or different types of updates to the KB). The rules will analyze each case stored in the Memory in order to decide if the RA could learn from some of them. The application of the rules will indicate to the RA what to infer from those cases.

As a result of those inferences, the KB will be modified.

The CA may get a complete or partial answer to the query from one or more system domains. With the aim of carrying out an initial analysis about the learning mechanism, we assume that only one domain answered the query completely. Future works will consider the possibility that more than one domain positively answers the query or part of it.

6.1. Cases Representation in the Cases Memory

The application of CBR requires a structured representation of knowledge in the form of cases. This makes further analysis and inference processes easier.

The information needed to store the cases is:

- a) a description of the situation in which the case occurred,
- b) the solution reached in the case,
- c) the consequences the solution of the case had on the environment.

For the RA, we use the frame shown in Figure 5 for the cases representation.

The RA learning should improve the KR process. The RA task is to classify the domains according to their capacity to answer a query made from another domain. This objective is not correctly fulfilled if the first domain in the list given to the CA is not the one that finally gives the answer.

CASE c	
Origin:	D_j
Quer	$t_1, t_2, \dots, t_i, \dots$
Ranking:	$RSV(D_1, q_c); \dots; RSV(D_k, q_c); \dots$
Answer:	D_m

where $j \neq k$ and $j \neq m$

Origin: domain that originated the query q_c .

Query: keywords of the query (all $t_i \in K_{q_c}$).

Ranking: ordered list of $RSV(d, q)$ transferred by the RA after the KR process.

Answer: domain that effectively answered the query q_c

Figure 5: Frame for the cases representation.

For that reason, the cases to be stored will be the ones corresponding to queries whose answers were not given by the domain that is best ranked by the RA.

Among the cases to be stored three situations can be distinguished:

- The domain D_m that answered the query q_c is in the ranking, but not in the first position, and the query keywords belong to the taxonomy of D_m in the KB. That is, they have indexing weights but, nevertheless, these values do not make it appear first in the list. Then, these weights turn out wrong or out of date, so they need to be corrected.
- The domain D_m that answered the query q_c is in the ranking, but not in the first position, and some query keyword do not belongs to the taxonomy of D_m in the KB. This could be a result of a RSV insufficient value. Then, those keywords should belong to the representation of

D_m . As a consequence, weights should be assigned to them.

- The domain D_m that answered the query q_c is not in the ranking because any query keywords belong to the D_m taxonomy in the KB. Then, those keywords should belong to the representation of D_m , and they should have assigned weights.

6.2. The Inference Mechanism

Starting from the stored cases, an inference mechanism based on IF-THEN rules will identify patterns of occurrence of keywords and will point out the action to be taken.

The rules classify the cases to evaluate when additional terms are needed in the taxonomy of a domain or changes are necessary in the weights assigned to certain keywords.

Every time a case enters the Cases Memory, the rules antecedents are evaluated. The rules whose antecedents are satisfied are executed.

The rules set defined at present is the following:

6.2.1. Rule 1

If all the query keywords belong to the taxonomy of the domain D_m that gives the answer, the weights assigned to these keywords should be modified in such a way that D_m is the best ranked for this query (See Appendix A).

Also, as these terms are already in the KB, they probably have a semantic expansion, whose words should also

modify their weights in the same proportion determined by the previously calculated similarity measure. (See Example 1).

Rule 1

Set

q_c : the query corresponding case c

D_m : answering domain.

D_1 : first ranked domain

$K_{qc} = \{t_i / t_i \text{ is a keyword of the query } q_c\}$

$EK_{Dm} = \{t_k / t_k \in T^i \text{ and } t_i \in K_{Dm}\}$

IF

$K_{qc} \subset EK_{Dm}$

THEN

$$we_i^{Dm} \Big|_{new} = \frac{RSV(D_1, q_c)}{RSV(D_m, q_c)} we_i^{Dm} \Big|_{current} \quad \text{for all } t_i \in K_{qc}$$

$$we_k^{Dm} \Big|_{new} = we_i^{Dm} \Big|_{new} Sim(t_i, t_k) \quad \text{for all } t_k \in T^i$$

time those n queries enter the system (see Appendix B).

Rule 2

Set

α, n : parameters to define

D_m : answering domain.

$C_{Dm} = \{c_j / c_j \text{ is a case with } D_m \text{ as the answering domain}\}$

$EK_{Dm} = \{t_k / t_k \in T^i \text{ and } t_i \in K_{Dm}\}$

$Y_{Dm} = \{t_i / t_i \text{ is a query keyword common to } n \text{ cases } c_j \in C_{Dm} \text{ and } t_i \notin EK_{Dm}\}$

IF

$\#Y_{Dm} = \alpha$

THEN

$$EK_{DM} \Big|_{new} = EK_{DM} \Big|_{current} \cup Y_{Dm}$$

$$we_i^{Dm} = \frac{1}{\alpha \log\left(\frac{N}{n_i + 1}\right)} \text{Max}_j \{RSV(D_1, q_{c_j}) - RSV(D_m, q_{c_j})\} \quad \text{for all } t_i \in Y_{Dm}$$

where n_i = number of domains that currently have t_i in their taxonomy.

6.2.2. Rule 2

When the same domain D_m answered for n cases (n is a previously specified number) and some terms that appear as keywords in the queries do not belong to the current taxonomy of D_m , these terms should be included in the D_m representation. The weight to be assigned to these keywords should be such that they allow D_m to be the best ranked every

The application of this rule is shown in Example 2.

6.2.3. Rule 3

Every time a new case is included in the Cases Memory, the system determines the rules whose antecedents are satisfied, and executes them. After executing some of the previous rules, it is necessary to compute again the rankings of the stored cases so as to eliminate those that have

been already corrected. This activity is performed by rule 3.

Rule 3
Set $C_{Dm}^1 = \{c_j / Dm \text{ answers the query } q_{ej} \text{ and } D_m = D_1\}$
IF $C_{Dm}^1 \neq \emptyset$
THEN Cases Memory = Cases Memory - C_{Dm}^1

This rule application is shown in the following examples.

6.3. Examples

To illustrate the learning mechanism carried out by the RA, we present a reduced example of a distributed DSS, using only three domains: Sales, Forecasting and Planning (N=3). The taxonomy of each domain is shown in Tables 5, 6 and 7, respectively. They are not the original representation that each domain expert has derived using the strategy proposed in Section 5.1 but the current representation of each domain after the system has begun to work.

Note that these representations are extremely reduced with the only purpose of simplifying the explanations.

The number of cases that should be in the Cases Memory should be previously specified so that rule 2 antecedents are satisfied. We will assume $n=2$ for these examples.

Keyword t_i	we_i^{sales}	Semantic expansion t_j	$Sim(t_i, t_j)$	we_j^{sales}
Sales	1,00	Salesman	0,55	0,55
		Sell	0,34	0,34
		Saleswoman	0,55	0,55
		Salesroom	0,17	0,17
Product	0,71			
Item	0,12			
Time	0,12			
Customer	0,74	consumer	1,0000	0,74
Graphic	0,23			
Profits	0,48			
Estimate	0,06			

Table 5. “SALES” domain taxonomy

Keyword t_i	we_i^{for}	Semantic expansion t_j	$Sim(t_i, t_j)$	we_j^{for}
Forecast	1,00	Prediction	1,00	1,00
Sales	0,21	Sale	1,00	0,21
Time	0,33			
Graphic	0,65			
Method	0,55	Exponential Smoothing	0,60	0,33
		Moving Average	0,60	0,33
Product	0,33			
Profits	0,06			

Table 7. “FORECASTING” domain taxonomy

In each of the following examples a different scenario is presented to illustrate the application of each rule.

This is only a simple example scenario, with few domains and keywords, since we do not have much space to develop a more complex example.

6.3.1. Example 1

Let's assume that in order to make an operative decision a user of the Planning domain needs certain information that does not habitually receive. For such a reason, he/she enters the dynamic DSS and formulates a query in natural language: "*Where can I find the graphics of the last year product sales?*" A CA takes this query to the RA. At this time, the called KR process begins. The keywords associated to this query are extracted. These keywords are: *graphic*, *sales* and *product*. Immediately, starting from the information stored in the router KB the RSV values are calculated [$RSV(\text{forecasting}, q_1) = 0,173$; $RSV(\text{sales}, q_1) = 0,166$]. (The keyword *sales* does not contribute to the value of any RSV because it belongs to all domains taxonomy, then $\log N/n_{\text{sales}} = 0$). According to these values, the domain presenting the greatest probability of answering the query would be Forecasting and then Sales, in this order.

This route is transferred to the CA, which goes to the Forecasting domain where the query is not answered. Then, it goes to the Sales domain where it gets an answer.

Next, the CA goes to the RA, completes the information corresponding to this search, and returns to the Planning domain to give it the requested information.

As the domain that answers does not coincide with the best ranked by the RA, the case is stored in the Cases Memory with the structure shown in Figure 6. Once the case enters the Cases Memory the rules are applied.

CASE 1	
Origin:	Planning
Query:	graphic, sales, product
Ranking:	Forecasting 0,173 Sales 0,166
Answer:	Sales

Figure 6: Cases representation for example 1.

All query keywords belong to the taxonomy of Sales domain, which answers the query. But "Sales" is not the best ranked. Then, the conditions to apply rule 1 are satisfied.

Therefore, the weights of the three terms: *graphic*, *sales* and *product* will be modified by the factor $f = 0,173/0,166 = 1,042$.

The new indexing weights associated to the keywords *graphic*, *sales* and *product* will be the following:

$$w_{\text{graphic}}^{\text{sales}} = 0,24$$

$$we_{sales}^{sales} = 1,04$$

$$we_{product}^{sales} = 0,74$$

But the terms *salesman*, *sell*, *saleswomen* and *salesroom* belongs to the semantic expansion of *sales*, then its weight must be modified too.

$$we_{salesman}^{sales} = Sim_{(sales, salesman)} we_{sales}^{sales}$$

$$we_{salesman}^{sales} = 0,57 ;$$

$$we_{sell}^{sales} = 0,35 ;$$

$$we_{saleswoman}^{sales} = 0,57 ;$$

$$we_{salesroom}^{sales} = 0,18$$

With these new weights, the RSV(forecasting, q_1) value becomes equal to the RSV(sales, q_1) value. Then rule 3 antecedents are satisfied so that case 1 is eliminated from the Cases Memory. For another similar searching, the Forecasting domain will be the best ranked.

6.3.2. Example 2

This example represents a scenario of application of rule 2. To this end we consider the domains taxonomy shown in Tables 5,6 and 7.

Let's assume that the user of the Planning domain requires information. To this end, he/she builds a query. When this query arrives to the RA, it is represented by the following keywords: *estimate*, *monthly* and *product*.

After the KR process, the RA specifies the following ordered list of RSV:

$$RSV(sales, q_2) = 0,154;$$

$$RSV(forecasting, q_2) = 0,058.$$

With this information, the CA first visits the Sales domain. As it does not get an answer, the CA goes to the Forecasting domain. This domain answers the query. Then, before returning to the original domain, the CA informs the RA about the results of the search.

As the domain that answers the query does not coincide with the best ranked by the RA, the case is stored in the Cases Memory with the structure shown in Figure 7.

CASE 2

Origin:	Planning
Query:	Estimate, monthly, product
Ranking:	Sales 0,154
	Forecasting 0,058
Answer:	Forecasting

Figure 7: Cases 2 representation for example 2.

The query keywords *estimate* and *monthly* do not belong to the Forecasting taxonomy, then, the antecedents required by rule 1 are not satisfied.

No case is stored in the Case Memory at this time. Therefore, the conditions for rule 2 execution are not verified and this case is stored in the Cases Memory.

In another opportunity, a user of the domain Planning requires the services of the dynamic DSS. This time, the representation of his/her query has the keywords: *estimate*, *profits* and *monthly*. And the ranking obtained by the KR process is:

$$RSV(sales, q_3) = 0,113;$$

$$RSV(forecasting, q_3) = 0,011.$$

Again, the domain that answers the query is not the best ranked. Therefore, the case is stored in the Cases Memory (Figure 8).

CASE 3		
Origin:	Planning	
Query:	estimate, profits, monthly	
Ranking:	Sales	0,113
	Forecasting	0,011
Answer:	Forecasting	

Figure 8: Cases 3 representation for example 2.

The query keywords *estimate* and *monthly* do not belong to the Forecasting taxonomy, then, the antecedents required by rule 1 are not satisfied.

This situation is repeated in another case (case 2) where the Forecasting domain also answered. Then, the antecedents for rule 2 are satisfied.

The application of this rule indicates that the keywords *estimate* and *monthly* should be included into the Forecasting representation with the following indexing weights:

$$we_{estimate}^{for} = 0,29$$

$$we_{monthly}^{for} = 0,11$$

With the modified KB, if a query similar to the q_2 is formulated, the KR process will obtain the ranking:

$$RSV(forecasting, q_2) = 0,162$$

$$RSV(sales, q_2) = 0,136$$

With these new weights, the $RSV(forecasting, q_2)$ and $RSV(forecasting, q_3)$ values become the best ranked. Then the rule 3 antecedents are satisfied, so cases 2 and 3 are deleted from the Cases Memory.

With these new values, the CAs will go over fewer domains to look for the answer to similar queries as q_2 and q_3 .

As it can be observed, the system performance will improve with these new values.

7. CONCLUSIONS

In this paper we have analyzed the mechanisms used by the intelligent component of a mobile agent-based DSS (the Router agent) to achieve an efficient operation. The RA avoids the system overload with mobile agents, and the constant interruption to domains with queries they might not be able to answer. To implement this mobile agent-based system we are using an evolutionary prototyping approach. At this moment, we have implemented a first prototype of the agent based architecture, using the "Aglets Software Development Kit" also known as "ASDK", created by IBM's laboratories [13].

The RA is responsible for giving a route to the collecting agents, which must visit the system domains searching for an answer to queries formulated by a user. To perform this, the RA has a KB where a representation of the information managed by each domain is stored. We have proposed an information retrieval

process that allows matching the query contents to the domain contents to identify a ranked list of domains to visit. To experimentally evaluate this mechanism, we have considered our college as an organization for analysis. The different departments (student, library, etc.) have been regarded as system domains. The great number of keywords defined for each domain, around 100 terms, confirmed the advantage of using a two-stage mechanism to assign the indexing weights. Using this strategy, the number of comparisons required to apply AHP to each domain diminished from approximately 4500 to about 10 sets of 45 comparisons each. The study of this particular example also allowed designing the Router KB, taking into account that it is necessary to have an accident base, a constructions base and a stop-words base, with the aim of improving the query-filtering process. This work is being prepared for its soon publication.

The centralized structure used by the information retrieval process enables an updated KB, using a learning mechanism. In this paper, we have presented a case-based reasoning strategy to analyze the quality of the information offered by the KR process and to suggest improvements. To this end, we have defined a set of rules that, in function of the errors detected in the route used by the CAs, suggest modifications of the data stored in the router KB.

There are a number of reasons for choosing CBR over conventional rule-based systems. The main requirement for

the system is that it should be able to support a variety of business domains that change in time. If we only apply a rule-based system, it is difficult to take into account the variety and complexity of the query-response pairs. CBR allows remembering previous situations and deciding an adequate KB modification according to accumulated experiences.

Although in this work we only present three rules, our goal is to expand this set of rules because it is necessary to obtain a more detailed classification of the cases in order to improve the KB updating strategy. Currently, we are studying modifications for the proposed rules with the purpose of considering the statistical occurrence of situations, the weights decrease when necessary, the systematic evaluation of the appropriate order of rules application, etc.

APPENDIX A.

Indexing weights modification in rule 1.

If all the query's keywords belong to the domain's taxonomy D_m that gives the answer, as this domain is not the best ranked, it should be assumed that its assigned weights are not the right ones. Therefore they should be modified in such a way that, for this query, $RSV(D_m, q_c)$ is as best ranked as $RSV(D_1, q_c)$.

$$RSV(D_m, q_c)|_{new} = RSV(D_1, q_c)|_{current} \quad (5)$$

By equation (4)

$$RSV(D_m, q_c)|_{new} = \sum w e_i^{D_m}|_{new} \log N/n_i \quad (6)$$

Assuming that the indexing weights of all the keywords modify f in same proportion:

$$we_i^{D_m}|_{new} = f we_i^{D_m}|_{current} \quad (7)$$

By substituting Eqs. (5) and (7) in (6):

$$RSV(D_l, q_c)|_{current} = f RSV(D_m, q_c)|_{current} \quad (8)$$

Finally, the new indexing weights of the keywords $t_i \in K_{qc}$ in the D_m representation are:

$$we_i^{D_m}|_{new} = \frac{RSV(D_l, q_c)}{RSV(D_m, q_c)} we_i^{D_m}|_{current} \quad (9)$$

APPENDIX B.

Indexing weights to be assigned to new keywords of the domain D_m (rule 2).

In n cases the domain D_m answers the query, but α query keywords do not belong to the domain D_m representation, these terms must be included into D_m taxonomy. The indexing weight of these terms must allow increasing the $RSV(D_m, q_c)$ until the maximum RSV of the best ranked domain in each case. Then,

$$RSV(D_m, q_c)|_{new} - RSV(D_m, q_c)|_{current} = \max_{j=1, \dots, n} \{RSV(D_1, q_j) - RSV(D_m, q_j)\} \quad (10)$$

As the α keywords $t_1, \dots, t_\alpha$ must be included into EK_{D_m}

$$RSV(D_m, q_c)|_{new} = RSV(D_m, q_c)|_{current} + \sum_{i=1}^{\alpha} we_i^{D_m} \log \frac{N}{n_i} + 1 \quad (11)$$

Let's note that when t_i is incorporated into D_m taxonomy, n_i must be increased in one unit.

Substituting (11) into (10),

$$\sum_{i=1}^{\alpha} we_i^{D_m} \log \frac{N}{n_i} + 1 = \max_{j=1, \dots, n} \{RSV(D_1, q_j) - RSV(D_m, q_j)\} \quad (12)$$

Assuming that the α keywords contribute uniformly to the RSV value.

$$\alpha \left(we_i^{D_m} \log \frac{N}{n_i} + 1 \right) = \max_{j=1, \dots, n} \{RSV(D_1, q_j) - RSV(D_m, q_j)\} \quad (13)$$

Then, the keyword t_i must be incorporated into D_m taxonomy with the indexing weight given by:

$$we_i^{D_m} = \frac{1}{\alpha \log \frac{N}{n_i} + 1} \max_{j=1, \dots, n} \{RSV(D_1, q_j) - RSV(D_m, q_j)\} \quad (14)$$

REFERENCES

- [1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Wokingham, UK, (1999).
- [2] R. Burke, K. Hammond and J. Kozlovsky, "Knowledge-Based Information Retrieval for Semi-Structured Text". *Working Notes from AAAI Fall Symp. AI Applications in Knowledge Navigation and Retrieval*, AAAI Press, Menlo Park, Calif. Pp19-24 (1995).

- [3] M. Caliusco, L. Cabral, H. Nissio, P. Villarreal, M. Galli, M. Taverna and O. Chiotti. "Distributed Architecture to Support Dynamic Decision Processes", *Proceeding XXVI CLEI, Mexico*, (September 2000).
- [4] F. Crestani, "Exploiting the Similarity of Non-Matching Terms at Retrieval Time". *Information Retrieval*, **2**, 25-45, (2000) .
- [5] F. Crestani, G. Pasi, "Soft Information Retrieval: Applications of Fuzzy Set Theory and Neural Networks". *Neuro-Fuzzy Techniques for Intelligent Information Systems*, Ed. N. Kasabov and R. Kozma, Physics Verlag (Springer Verlag), Heidelberg, Germany, 287 , (1999).
- [6] J. Dale and D. De Roure. "A Mobile Agent Architecture for Distributed Information Mangement". *Proceedings of the International Workshop on the Virtual Multicomputer*. March. (1997).
- [7] J.J. Daniels and E L N Rissland. "A case-based approach to intelligent information retrieval. *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*. Seattle, Washington, USA, pp 238-245 (1995).
- [8] S. Fünfroeken, F. Mattern, " Mobile Agents as an Architectural Concept for Internet-based Distributed Applications: The WASP Project Approach". In Steinmetz (Hg.) Proc. KiVS'99 ("Kommunikation in Verteilten Systemen), (1999).
- [9] N. Jennings. "An agent-based approach for building complex software systems". *Communications of the ACM*. Vol. 44, No. 4, pp 35-42 (2001)
- [10] C. Knoblock and J.L. Ambite. "Agents for information gathering", *IEEE Expert: Intelligent Systems and their Applications* , (September/October 1997).
- [11] J. Kolodner, *Cased Based Reasoning*. Morgan Kaufmann Publishers, USA, (1993).
- [12] B. Krulwich and C. Burkey, "The ContactFinder Agent: Answering Bulletin Board Questions with Referrals." *Proceedings of 13th Nat'l Conference. AI (AAAI 96)*, AAAI Press, Mento Park, Calif. Pp 10-15 (1996).
- [13] D.B. Lange, IBM Tokyo Research Laboratory, Feb. 1997, *Java Aglet Application Programming Interface (J-AAPI) White Paper - Draft 2* <http://www.trl.ibm.co.jp/aglets/JAAPIwhitepaper.htm>
- [14] D. Leake, " CBR in context: The present and Future". *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. (AAAI Press/MIT Press, 1996). 1-35
- [15] R. Lopez de Mantaras, E. Plaza; "Case-Based Reasoning: An overview." *AI Communications Journal* **10**, 1 ,21-29, (1997) .
- [16] D. Mladenic, "Text-Learning and Related Intelligent Agents: A Survey". *IEEE Intelligent Systems*. Pp 44-54 July/August (1999).
- [17] J. Nagel, 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).

- [18] D. Rus, R. Gray and D. Kots, *Transportable Information Agents*, in Michael Huhns & Munidar Singh book, *Reading in Agents*, (1998).
- [19] T. L Saaty. *The Analytical Hierarchy Process*. New York: McGraw-Hill, (1980).
- [20] W. Shen and D.H. Norrie. "Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey". *Knowledge and Information Systems, an International Journal*, 1(2), 129-156, (1999).
- [21] Stegmayer, G., M.L. Taverna, O. Chiotti and M.R. Galli. *Dynamic Communication Mechanism of a Distributed DSS: The Information Retrieval Process of the Router Agent*. Proceeding CACIC '2000, Usuhaia, Argentina, (2000).
- [22] A. Vivacqua, "Agents for Expertise Location" *Proceedings 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace*. Technical Report SS-99-03 - Stanford, CA, USA, March (1999).
- [23] M. Wooldridge, N.R. Jennings and D. Kinny. "A Methodology for Agent-Oriented Analysis and Design". *Proceeding of 3rd Int. Conference on Autonomous Agents (AGENTS-99)*, Seattle, W.A., 69-76, (1999).