Multi-Agents based Dynamic Request Placement Strategies in Fully Distributed Information Systems

Abstract

In this paper we propose several strategies for dynamic query placement in order to reduce the response time of services on the Internet. These strategies are based on the cooperation of multi-agents, which are organized as a community of cognitive agents, and a community of reactive agents. Reactive agents propose services. Cognitive agents have the responsability of placing the queries that they receive from users. To take such a decision, they are able to cooperate and to learn about the state of reactive agents. We also develop a model of knowledge, which enable cognitive agents to characterize the quality of service of the reactive agents on the Internet. When a query is submitted to a cognitive agent, it uses its knowledge on the state of the system, possibly by cooperating with other cognitive agents. In order to facilitate the composition of complex services, we also define a new sub-contractor agent type.

Keywords: Load balancing, agents, cooperation, document query

1 Introduction

The fast development of a more and more distributed and more and more multi-media information on the Internet, is one of the major evolutions of present computer technology. Though the network bandwidth increases, specially on backbones, the increase in the volume of documents and in the number of users result in variable quantities of service and in the very mediocre access time.

Improving response times implies the use of multiple techniques, such as image compression, document filtering, cooperative retrieval, optimized documents placement, and task allocation. Within the U-Doc [**?**] a french project which is the implementation of a collection of assistance tools for hyper-documents retrieval on the Internet, we have studied the optimized query placement in order to reduce the response times.

Task allocation and load balancing was widely studies in the literature [**?**, **?**, **?**, **?**, **?**] in the context of distributed systems. The purpose is to optimize the use of resources in order to improve the throughput of the distributed system, and thus to reduce the response time. There exist static and dynamic techniques to implement load sharing in distributed systems. In the Internet context, static solutions which are mainly based on operations research results [**?**, **?**], are not applicable as they rely on a previous knowledge of the state of the system and of the applications. Dynamic solutions [**?**, **?**, **?**, **?**] try to avoid this limit. However, they guess the availability of several parameters (number of processors, length o their waiting for processing tasks, etc), allowing to determine the state of the system. But when accessing a document or submitting a query on the Internet, the geographical location (quite commonly located in a different continent) of the target site may make very imprecise (useless) the evaluation of these parameters. The response time and transfer throughput are the only available informations. Therefore other approaches are needed.

In this paper we propose several strategies for dynamic query placement, in order to reduce the response time of services on the Internet. They are based on the use of multi-agents [**?**, **?**], organized into a community of cognitive agents and a community of reactive agents. Reactive agents are merely the final servers (available services in the system such searchers, bibliographic databases, movie databases, etc.). Cognitive agents are able to cooperate and to learn about the state of reactive agents. They have the responsibility of placing the queries which they receive. We also develop a model of knowledge, which enables cognitive agents to characterize the *quality* of service of the reactive agents on the Internet, and to learn informations on the system state through their experience. Thus when a query is submitted to a cognitive agent, it uses its knowledge on the system state to place the query. If its knowledge is not sufficient to take this decision, it enquires a defined group of cognitive agents trying to complete its knowledge. If the knowledge obtained does not enable it to decide about the allocation, it start a request for biding negotiation process [**?**]. Finally, in order to enable composition of complex services with large added value, we also define a new type of sub-contractor agent.

The paper is organized as follows: in paragraph 2 we describe the context of this work. In paragraph 3 we introduce the agents model, and the way in which the cognitive agents represent and learn their knowledge. Strategies for dynamic query placement are described in paragraph 4. We introduce sub-contractor agent in paragraph 5. As a conclusion, we compare our approach with other ones which are also based on agents.

2 Context and Problems

We studied this problem in the context of the U-Doc project, which objective is the implementation of a collection of assistance tools to facilitate document access on the Internet, and the implementation of their administration. We first briefly describe the U-Doc architecture [**?**] then our framework.

2.1 The U-Doc Architecture

The **U-Doc** architecture is depicted in Figure **??**. The client access request arrive to the extern interface of **U-Doc** (mailer or DQBE). After have been formated, the request it is delivered successively to: 1. the *Concepts Manager and thesaurus* module which divides the request in more precise and domain Dependant ones (e.g requests about colors, sounds, geography, etc.), 2. the *Indexers* module which search the documents in the local documents database, 3. the *profiler* to extract from inmediat request the long term profile, 4. the *Interrogateur* delivers the clients' inmediat request, and the permanent request produces by the *Profiler* to external *Searchers* (lycos, Yaoo, etc.) and evaluates the abstracts and titles gotten, 5. the *Glaneur* to search the selected documents. Then these documents are delivered to the *Storage system*, and finally to the client.

The *Storage system* keeps the documents, abstracts and annotations in a cache memory and in the tertiary memory. The *Thesaurus* manages a corpus of the reference documents (for instance articles of a review previously chose to describe the concerned area), and learns the correlations among the concepts in that corpus.

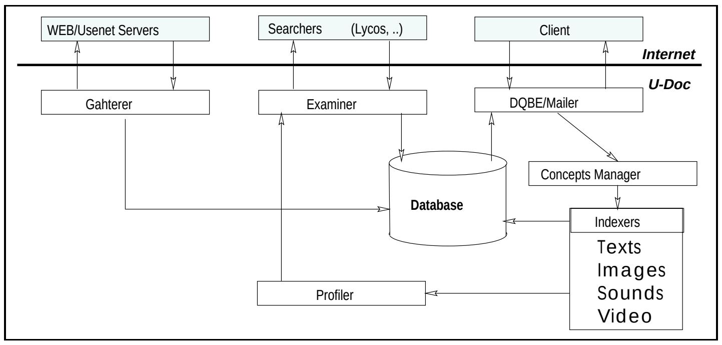


Figure 1: Architecture of U-Doc.

2.2 Problem and our Framework

A document retrieval query rely on localization, format modification, translation, document selection operators, which can be based on indices, and on information extraction (data meaning) on normalized documents such as SGML ones. In our architecture a retrieval query is decomposed by the DBQE module into a set of document retrieval operators, represented by a data flow graph such as the one in Figure **??** We suppose that techniques for query decomposition are known, as they are not the subject of this paper. Following, based on a placement strategy aiming response time optimization, these operators are placed on specialized servers such as the ones which propose services for document searching. On the Internet,

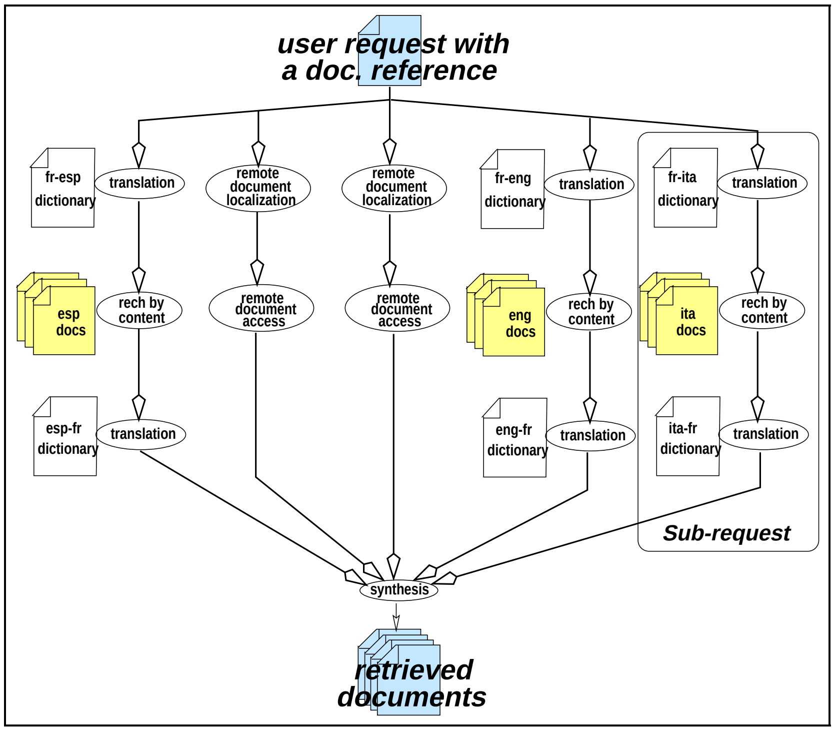


Figure 2: Decomposition of a request by the DQBE/Mailer.

there are many such search services, such as Lycos, Yahoo, etc. A document can also be replicated on several sites, such as proxies or mirrors. Presently, dynamic query placement in U-Doc takes place in two places : firstly in the examiner for the choice of a search server on a previously defined list, when the query comes from an user of the profiler, secondly, at the gatherer level, for the choice of a document when this document exists in several servers. In both cases, it may be useless the measure of classical parameters helping the allocation problem. Therefore, we do not have a classical task allocation problem in the usual sense, as we cannot decide process allocation in the remote sites.

One solution in our Internet framework of added value services will be submit the same query to all servers and choose the one which delivers the fastest answer. This is an easy but quite expensive solution. The problem is thus to define dynamic placement strategies on the various subtasks of an R document retrieval query, in order to reduce the response time. We propose an heuristic solution, based on past experience in terms of response times, and possibly on the experience of other sites.

2.3 The Problem

Our main objective in the **U-Doc** project is have short response times. To fulfill this objective we must solve the problem of how to choose the searcher and the document server (if there exist the choice) to get shortest response time. Upon the previous explanation of the **U-Doc** architecture's behaviour, the allocation techniques are necessitated by the *Interrogateur* to select a *Searcher* from a set previously established, and also to the *Glaneur* to select one document server from the list obtained by the selected *searcher*.

The conditions of our system are the actual ones, that is, Internet available *searchers* (Lycos, Yaoo, etc.) do not give information about its states useful to calculate the response time of a request (e.g. number of waiting for processing tasks, processing power of the server, size of the waiting tasks. etc.). Then because these conditions and our aim to make transparent this problem to user, two solutions are possible. The first is to chose the servers (to search and to retrieve the document) randomly, the second is select the servers taking into account the experience obtained of previous relationships. Because we are using agents having the learning capability our choice is the second. One of the strategies proposed is a combination of both, however.

2.4 Model

Though our approach was developed for the U-Doc distributed information system, the proposed mechanisms are general ones and can be applied in other context than our particular. Figure **??** depicts the distributed system that we take as framework and we are using to simulate the performance of the algorithms. The system is constituted of three sets distributed among a set of sites (computers) connected by a network: a set of *cognitive agent* denoted by $C_A = \{C_{A1}, C_{A2}, C_{A3}, \ldots, C_{A\alpha}\}$. Another of *reactive agent* denoted by the set $R_A = \{R_{A1}, R_{A2}, R_{A3}, \ldots, R_{A\beta}\}$, and a set of *u*sers, denoted by $U = \{U_1, U_2, \ldots\}$. A site lodge zero or one user, zero or one *cognitive agent* and zero or more *reactive agent* .

The set of *reactive agent* represent the available services in the system. The same service can be delivered by different *reactive agents*. *Cognitive agents* receive the tasks delivered by users, are able to communicate with each other by sending messages over the communication network, learn about the system state and have the skill to allocate a task based in its knowledge about the system (servers). Finally, users or clients deliver theirs tasks $T_\gamma = \{t_1, t_1, \ldots, t_\kappa\}$ to the *cognitive agents* via an interface.

In the **U-Doc** architecture, the set of *reactive agents* represent the set of searchers availables by Internet (Lycos, Yaoo, etc.). A delivered task is either a search request that should be addressed to one of the searchers or a request of access to get an specific document that must be delivered to one of the document servers.

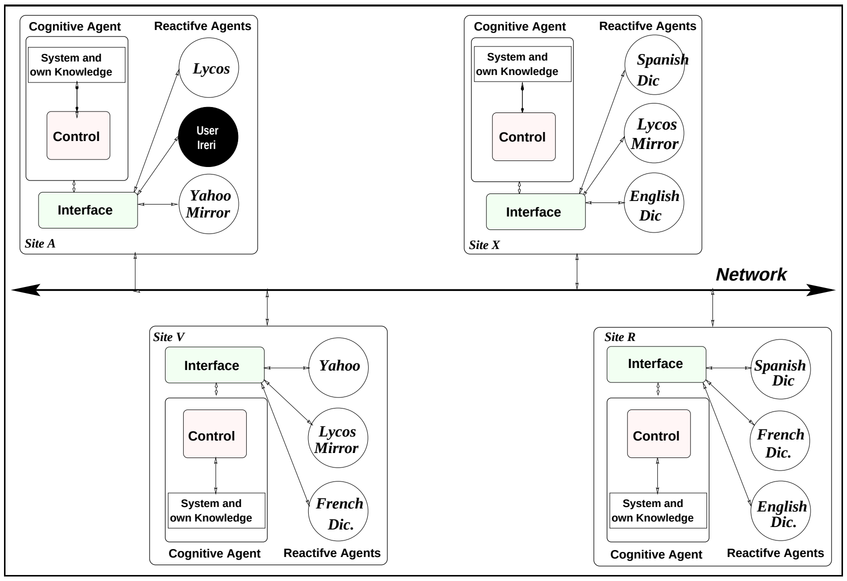
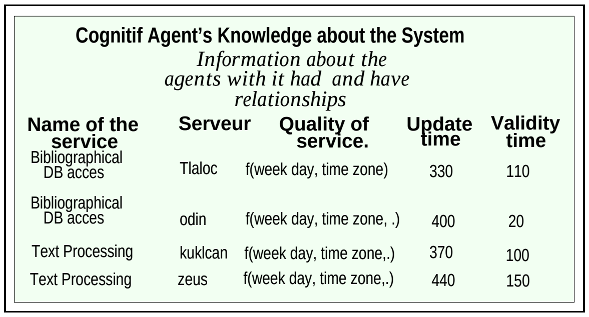


Figure 3: The system is organized in two communities: one of Reactive Agents delivering the services available over the system the other of Cognitive Agents keeping the global state of the system. The major objective of the Cognitive Agents community is helping to distribute in a fair way the load among the first community.

A *cognitive agent* as depicted in Figure **??** in its structure contains a knowledge and a control element. Control element implements the location policy (transfer policy is *every time a task is received, decide about its transfer*, and selection policy is *the arriving task must be allocated*). To decide about the allocation of a task the *cognitive agent* necessitates information about the state of the servers offering the required service. As established in **??** on our framework, even if *reactive agents* are able to deliver useful information to decide about an allocation, communication delays make useless they. We are using the learning capability of our *cognitive agents* to alleviate this problem Figure **??** shows the information learned by *cognitive agents*.



Cognitif Agent's Knowledge about the System

Information about the agents with it had and have relationships

Name of the service	Serveur	Quality of service.	Update time	Validity time
Bibliographical DB acces	Tlaloc	f(week day, time zone)	330	110
Bibliographical DB acces	odin	f(week day, time zone, .)	400	20
Text Processing	kuklcan	f(week day, time zone,.)	370	100
Text Processing	zeus	f(week day, time zone,.)	440	150

Figure 4: Knowledge a cognitive agent has about the system state.

2.5 Learning and Knowledge

Cognitive agents are able to learn about the evolution of reactive agents' states. For this purpose, they memorize for each agent, the next information that will be helpful to "predict" their service quality. We denote q_{xi} the quality of service x on the *reactive agent* i.

- q_{xi}: quality of service.
- Th: throughput of a query on the network.
- T: response time of a reactive agent to a query.
- x: the service proposed by the reactive agent.
- D: an array containing the days of the week.
- H: an array containing *cognitive agent* local time.
- V: validity duration of the knowledge.
- μ: a coefficient between 0 and 1, which represents the cognitive agent capability to remember the past

In our Internet context the quality of a service of a *reactive agent* is function of the response time, of the throughput, of the *reactive agent* local time (a server is more charged during the work times than at night) and of the day of the week (week ends are les loaded than other days). Thus a *cognitive agent* learns the behaviour of a *reactive agent* for each hour of the day and for each day of the week. To do this it use the $q_{xi} = Th/T$ formula to measure this quality when it sends it a task. The response time is an indicator of the *reactive agent* 's load, while the throughput is an indicator of the network load.

It is necessary to take into account the changes of behaviour of a *reactive agent* however. In our case, every time an answer of a *reactive agent* is received, its quality service for that day and time is modified as follows:

$$q_x = \mu * qx + (1 - \mu)q$$

Where μ is comprised between 0 and 1, and represents the agent capability to remember the past. The choice of the best μ coefficient is determined by our simulation results.

However if the informations have not been updated before some delay V, they are considered out-of-date. In that case, the cognitive agent must start a new learning phase on the whole relevant knowledge.

3 Dynamic Requests Placement Strategies

Two criteria appear to be essential in the dynamic request placement: first *work distribution* implies that an application must be widely distributed in order to use in the best way the available services, and the *locality criterion* that aims reducing the overheads due to communications by dispatching the application only over a neighborhood. These two criteria are easily expressed by an economic equation [**?**, **?**], but one can see in a straightforward way that these requirements are opposite. The strategies presented are dynamic and non pre-emptives. They are based on: the behaviour of *cognitive agent* which collaborate to achieve a common goal; *reactive agent* which execute a task; a bidding protocol [**?**] used by *cognitive agent* to get information about the system state and finally on the capacity of learning of the cognitive agent. The use of a multi-agents approach allows us to deal with the trade-off problem in a dynamic way.

3.1 Strategy of Placement Based on Service Negotiation

Initially cognitive agents have no knowledge on the state of the system due to lack of experience. Then to determine which reactive agent to choose, and to enrich at the same time its knowledge, it starts a process of negotiation similar to that of bidding found in free markets. Three phases in a such process are identified: first a *request-for-bidding* is launched beside all reactive agents proposing the service; second, an *evaluation* of reactive agents' replies is executed; and third the *contract attribution* phase determines the reactive agent on which the request is placed. If this negotiation mechanism is general and simple, it necessitates a lot messages. In a context of a large system as Internet, the cost associated to such a communication can be prohibitory and most important, the servers currently are not able to reply to a request for bid (today there is a great effort to establish the actual minimal information needed in today distributed systems, some formalisms like KQML [**?**] and kif [**?**] are been studied). Therefore, we propose below strategies based on this simple biding negotiation protocol, but using the learning capacity of *cognitive agent*, and different organizations of the communities of *reactive agent* and *cognitive agent*, to minimize its number of times it must be executed.

Let sum-up the strategic we are using and the semantic of some parameters the *cognitive agent* use to decide upon the allocation of a task.

the knowledge of *cognitive agent* is for each *reactive agent* serving a type of service. this information is organized in an array having for each day a set of checking points meaning the behaviour of the *reactive agent* learned of its experiences. By *useful information* we mean: that the information is fresh enough, and based on enough of experiences to be confident.

Strategy 1: when the agents have an individual organization

We present here the behaviour of the system when no organization exists between the two communities of agents. When a cognitive agent receives a task, his knowledge of the system's state should be used for the task allocation. If the knowledge is insufficient, the *cognitive agent* selects randomly a *reactive agent* submits it the task and learns its behaviour. Upon the activity of the user delivering tasks to the *cognitive agent* the knowledge it has will became useful to take a decision easily.

Another approach that we take is create a watching *reactive agent* for service. When a *cognitive agent* address its request to this *reactive agent* when it needs information. In this case the behaviour of the *cognitive agent* is almost that described by the bidding negotiation described previously, excepting that the use of knowledge of the agent limit its execution. This *cognitive agent* behaviour is depicted by the procedure 1. Every time the *cognitive agent* receives a reply it updates its partial knowledge about the system.

Procedure 1 `Cognitive Agent allocation`

```
Case event of {
    Task T :
        For each subtask $t \in T$ do
            If local information is enough to allocate t
                allocate(t);
            Else { * start the negotiation to get the service *
                RFB(service);
                evaluate(offers);
                allocate(t);
                update knowledge;
            }
    Load :  ···
        ⋮
}
```

Example 1.

To show the behaviour of the algorithm we consider the next example having two services $S = \{s1, s2\}$ offered by reactive agents placed on sites $A,\ B,\ C$ and D. The service s_1 is offered by reactive agents of the sites A and B while the service s_2 by sites C and D. Submitted requests are $R1,\ R2,\ R3$ and $R4$ that make call respectively to services $\{s1, s2\}$, $\{s1\}$, $\{s1,\ s2\}$ and $\{s1\}$.

Table **??** illustrates the chronograme of a request placement following the strategy based on service negotiation. Initially sites do not work. Similarly cognitive agents have no knowledge on the state of the system. When a cognitive agent has to place the request R_1 that makes call to services s1 and s_2, the former makes a call for proposals beside four sites $A,\ B,\ C$ and D. As all sites propose the same quality of service, the cognitive agent places without preference the request R_1 on A and C, it also modifies correspondingly its knowledge on the quality of service of the two sites. During the arrival of request R_2, the cognitive agent, consulting its knowledge, allocates this request on the site B that is becoming the site proposing a best quality of service for s_1. The execution of the totality of requests necessitates 6 broadcast and 24 point-to-point communications.

Cooperative placement strategies

Cooperative strategies are based on the organizations of *reactive agent* and *cognitive agent* in groups. This organization has as objective reducing:

- the number of messages exchanged between agents.
- the quantity of information to manage at the level of each agent.

Table 1: Chronogram of the execution of the sequence R_1, R_2, R_3, R_4, R_1 and R_5

temp	Site/Req.	s_1/A	s_1/B	s_2/C	s_2/D	X' Knowledge	Y' Knowledge	Z' Knowledge
t1	X/R_1	1_{R_1}		1_{R_1}		$s_1/A = s_1/B = 0$ $s_2/C = r_2/D = 0$		
t2	Y/R_2	1_{R_1}	1_{R_2}	1_{R_1}		Valid Inf.	$s_1/A = 1$ $s_1/B = 0$	
t3	X/R_3	$2_{R_1,R_3}$	1_{R_2}	1_{R_1}	1_{R_3}	Valid inf.	Valid inf.	
t4	X/R_1✞ Y/R_2 ✞	1_{R_3}	0	0	1_{R_1}	not valid inf.	not valid inf.	
t5	Z/R_4	1_{R_3}	1_{R_4}			not valid inf.	not valid inf.	$s_1/A = 1$ $s_1/B = 0$
t6	X/R_1 X/R_5	$2_{R_1,R_3}$	$2_{R_4,R_5}$	$2_{R_1,R_5}$		$s_1/A = s_1/B = 1$ $s_2/C = 0; s_2/D = 1$		

- the overhead associated to the placement algorithm.

The idea to organize process in groups has been implemented in different systems such as Amoeba [**?**], PVM [**?**], etc. and has proven to be a powerful mechanism to reduce the complexity in distribution costs. organizing the agents in groups can be in function of the geographical distribution of the agents, then we speak of *geographical clustering*, or in function of agents' characteristics for instance: their service, their homogeneity, etc. We speak then of *virtual clustering*. Each of these ways to organize the agents communities has its advantages and drawbacks. The actual problem is that for each application choose the best organization.

Strategy when AR are organized in groups of collaborators

The principle of grouping elements is carry out a global load sharing by means of the load sharing among the groups. When reactive agents are organized in virtual groups by service type. The outcome we look for is reducing the number of messages needed to carry out a global load sharing. Some other advantages of executing load sharing in groups are reduce the information a *cognitive agent* must manage, reduce the overhead of the load sharing algorithm, and establishing a boundary helping the satisfaction of the *locality criterion*.

Organizing in this way our communities necessitates a management for each group. In this case, our approach associates an administrator to each group (type) of reactive agents. Cognitive agents can address it to obtain useful to allocation information. The administrator update its informations about the elements of the group by sending them periodically "probe" requests.

The algorithm an *cognitive agent* execute when it receives a task is depicted in the Procedure 2. In the first instance it will verify if the information it has is enough to allocate the task, other ways if it knows the manager necessitated it will request him the missing information. Otherwise it start the biding protocol. The drawback of this algorithm is that a manager by service is necessitated.

Procedure 2 `Cognitive Agent allocation`

```
Case event of {
    Task T :
        For each subtask t ∈ T do
            If local information is useful to allocate the subtask
                allocate(t);
            Else
                If the manager of the required service is known
                    negotiate with the manager the service;
                    allocate(t);
                    update knowledge;
                Else { * start the negotiation to get the service *
                    RFB(service);
                    evaluate(offers);
                    allocate(t);
                    update knowledge; * about the manager and services *
                }
    Load : ···
        ⋮
}
```

Example 2.

We resume the same series of requests $R_1,\ R_2,\ R_3$ and R_4 as well as services s_1 and s_2 used in the example 1. There exists 5 sites $A,\ B,\ C,\ D$, and E. The service s_1 is offered by reactive agents on sites A and B while service s_2 is offered by reactive site agents $C,\ D$ and E. Reactive agents are regrouped according to their type of service, and therefore there exists a manager for service s_1 and another for service s_2. We also add another aspect concerning the duration of validity of the information that is supposed of 2 units of time. Request $R_1,\ R_3$ and A_5 are launched by cognitive agent on the site X, the request R_2 by cognitive agent of the site Y and the request R_4 by cognitive agent Z.

An execution of the different requests is illustrated in Table **??**. To simplify our example, the quality for each service is represented by a simple value. At time t=1, cognitive agent of site X starts request R_1 that calls services s_1 and s_2. By asking service managers Ms_1 and Ms_2, it learns that sites A and B propose the same quality of service for s_1 while C and D propose the same for s_2. Without preference, R_1 is placed on sites A and C. At time t=4, the duration of the validity of knowledge being fixed at 2, knowledge acquired by cognitive agent of site X is expired. At time $t = 5$, request R_4 that simply uses service s_1 is started by Z. At this time the knowledge of cognitive agent of site Z have allowed to favorably place the request on site B. The placement of these requests on the different reactive agents necessitate 4 broadcast, 3 multi-casts and 29 point-to-point communications.

Table 2: Chronogram of the execution of the sequence $R_1, R_2, R_3, R_4, R_1\ and\ R_5$

temp	Site/Req.	s_1/A	s_1/B	s_2/C	s_2/D	s_2/E Knowledge	X' Knowledge	Y' Knowledge	Z'
t1	X/R_1	1_{R_1}		1_{R_1}			$s_1/A = s_1/B = 0$ $s_2/C = r_2/D = 0$		
t2	Y/R_2	1_{R_1}	1_{R_2}	1_{R_1}			Valid Inf.	$s_1/A = 1$ $s_1/B = 0$	
t3	X/R_3	$2_{R_1,R_3}$	1_{R_2}	1_{R_1}	1_{R_3}		Valid inf.	Valid inf.	
t4	X/R_1✞ Y/R_2 ✞	1_{R_3}	0	0	1_{R_1}		not valid inf.	not valid inf.	
t5	Z/R_4	1_{R_3}	1_{R_4}				not valid inf.	not valid inf. $s_1/B = 0$	$s_1/A = 1$
t6	X/R_1 X/R_5	$2_{R_1,R_3}$	$2_{R_4,R_5}$	2_{R_1}		1_{R_5} $s_2/C = 0; s_2/D = 1$	$s_1/A = s_1/B = 1$		

Strategy Organizing Cognitive Agents in Virtual Groups

To allow knowledge sharing between cognitive agents, we organize them in groups. Each *cognitive agent* knows the elements of its group. Thus, when a cognitive agent does not know how to place a request due to lack of knowledge, it address a request to others cognitive agent in the same group to enrich its knowledge. If after such knowledge enrichment, it still does not know how to place the request, it initiates a negotiation process.

Procedure 3 `Cognitive Agent allocation`

```
Case event of {
    Task T : *task allocation request*
        For each subtask $t \in T$ do
            If local information is useful to place the subtask
                allocate(t);
            Else { *the agent tries to obtain information from its friend*
                enrich-knowledge(t, Friends);
                If the enriched information is useful to allocate t
                    allocate(t);
                Else { *start the negotiation to get the service *
                    RFB(service);
                    evaluate(offers);
                    allocate(t);
                    update knowledge; * about the manager and services *
                }
        Load :  * a reply containing information about a reactive agent*
            ⋮
    }
```

Strategy 3: offering complex services by sub-contractor agents

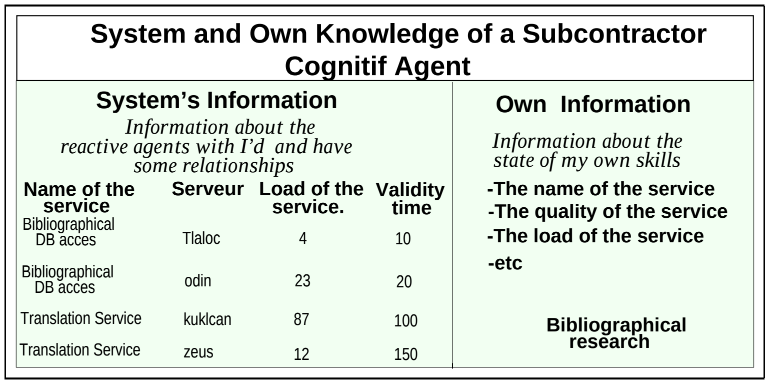
In multi-media information systems, to be able to offer complex added value services, it is necessary to allow the composition of services, that is,give the possibility to build a service from other less complex ones offered by agents. An instance of a complex service is the bibliographical research that is composed from a translator, a localization server, a gatherer and a format translator. We introduce a new type of agents we name *sub-contractor* who offers complex services. *sub-contractor* agents have a behavior of reactive agent because they offer a service (even if complex, is a service), but also the behaviour of a cognitive agent since they construct their complex services from their knowledge on the other reactive agents or sub-contractors (Figure **??** shows some of the information an *sub-contractor* keeps about the system). Sharing the knowledge among *sub-contractors* is useful, because not all the *reactive agent* have the same validity time of informations. Then sharing information is interesting mainly to compleat them when is necessary to take a decision about the allocation. The results obtained in this way have as characteristics:

- the time of response of a *subcontractor cognitive agent* is cheaper because the communication is at group level and not at system level (a mutlticast replace a broadcast).
- the quality computed can not be the best on the system.

Interactions between the different agents are illustrated by the Figure **??**. The algorithm used by sub-contractor agents to place requests is briefly developed below. The algorithm implemented is shown in procedure 4.

Procedure 4 `Sub-contractor Cognitive Agent`

```
Case event of {
```



System and Own Knowledge of a Subcontractor Cognitif Agent

System's Information

Information about the reactive agents with I'd and have some relationships

Name of the service	Serveur	Load of the service.	Validity time
Bibliographical DB acces	Tlaloc	4	10
Bibliographical DB acces	odin	23	20
Translation Service	kuklcan	87	100
Translation Service	zeus	12	150

Own Information

Information about the state of my own skills

-The name of the service
-The quality of the service
-The load of the service
-etc

Bibliographical research

Figure 5: A sub-contractor cognitive agent keeps information letting him to delivers the service of planning a complex task. In this illustration the name of the service is bibliographical search, necessitating the services of bibliographic database and a text processing.

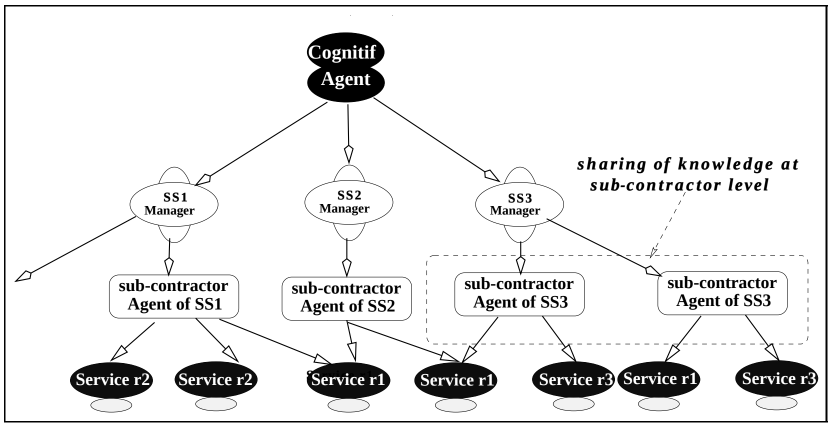

Cognitif Agent
sharing of knowledge at sub-contractor level
SS1 Manager
SS2 Manager
SS3 Manager
sub-contractor Agent of SS1
sub-contractor Agent of SS2
sub-contractor Agent of SS3
sub-contractor Agent of SS3
Service r2
Service r2
Service r1
Service r1
Service r3
Service r1
Service r3

Figure 6: Interactions among the three agents communities.

```
RFB :
    For each subtask $t \in T$ do
        If own information is useful to allocate or subcontract the subtask
            compute bid;
            return(bid);
        Else {
            ask for useful information to its collaborators
            If received information is enough
                compute(bid)
                return(bid);
            Else
                For each missing service{
                    start a negotiation
                    If all services are obtained
                        compute(bid);
                        update information
                        return(bid)
                    Else
                        delivers a diagnostic message
                }
            }
        **Task :**  *allocate(task)*
            ⋮
    }
```

Example 3

We consider three services $s_1,\ s_2$ and s_3, s_1 offered by reactive agents of five sites $A,\ B,\ C,\ D$ and E, s_1 by A and B, s_2 by C and E, and s_3 by F. We consider also three sub-contractor agent types SS_1, SS_2 and SS_3, each one proposing an added value service composed from s_1, s_2 and s_3. The sub-contractor type SS_1 using s_1 and s_2 is available on sites $A,\ B$ and E; The sub-contractor type SS_2 using s_2 is available on sites C and D while the sub-contractor type SS_3 using s_1 and s_3 is available on site C. The validity duration of the information is a function of the service, and is 1 for services s_1 and s_2, and 3 for service s_2. Three complex requests $R,\ S$ and T are considered, R accessing sub-contractors types SS_1 and SS_2, S accessing SS_2 and SS_3 while T accesses SS_2.

An execution of these requests is illustrated by Table **??** that shows the evolution of knowledge obtained on the different sites. The evolution of the quality of service is illustrated by Figure **??**. At time t=1, the request R making call to sub-contractors types SS_1 and SS_2 is launched by a cognitive agent on site X. The former, having no knowledge, makes a request-for-bids (RFB) beside sub-contractors types SS_1 and SS_2 on all sites. Sub-contractors, due to lack of knowledge, also start a call for proposals beside their reactive agents so as to complete their knowledge. Knowledge thus reverberated and obtained by the cognitive agent on site X allows it to place the request on sites A and C without preference. Sites A and C reverberate the work on services s_1, s_2. At time t=2, request S launched by a cognitive agent on site Y also execute a **RFB** beside sub-contractors types SS_1 and SS_2. These sub-contractors types, having required knowledge, no longer need to get information beside their service suppliers. Request S is placed on sites C and D. At time t=3, request R is ended while T is launched by a cognitive agent on site Z. Not having knowledge, the cognitive agent of site Z makes a call for proposal beside sub-contractor SS_2 whose knowledge have expired. This results in that sub-contractors SS_2 on site C and D inform mutually to complete their knowledge. And finally the two sub-contractors have to inform beside reactive service supplier agents in order to refresh their knowledge, and the request Z is placed on site D.

Table 3: Chronogram of the execution of the sequence R_1, R_2, R_3, R_4, R_1 and R_5

time	Request/site	SS_1/A	SS_1/B	SS_1/E	SS_2/C	SS_2/D	SS_3/C	Knowledge of X	Knowledge of Y	Knowledge of Z
t1	R/X started/X	$s_1/A = 0$ $s_1/B = 0$ $s2/C = 0$ $s2/D = 0$ R	$s_1/A = 0$ $s_1/B = 0$ $s2/C = 0$ $s2/D = 0$	$s_1/A = 0$ $s_1/B = 0$ $s2/C = 0$ $s2/D = 0$	$s_1/A = 0$ $s_1/B = 0$ R	$s_1/A = 0$ $s_1/B = 0$		$SS_1/A = 0$ $SS_1/B = 0$ $SS_1/E = 0$ $SS_2/C = 0$ $SS_2/C = 0$		
t2	S/Y	idem	idem	idem	idem	$s_1/A = 0$ $s_1/B = 0$ S	S	idem	$SS_2/C = 1$ $SS_2/D = 0$ $SS_3/A = 2$	
t3	R✞/X T/Z	$s_1/A = 0$ $s_1/B = 0$ $s_2/C = 0$ $s_2/D = 0$	$s_1/A = 0$ $s_1/B = 0$ $s_2/C = 0$ $s_2/D = 0$	$s_1/A = 0$ $s_1/B = 0$ $s_2/C = 0$ $s_2/D = 0$	ask to D $s_1/A = 1$ $s_1/B = 1$	ask to C $s_1/A = 1$ $s_1/B = 1$ T	$s_1/A = 1$ $s_1/B = 1$ $s_3/B = 0$			$SS_2/C = 1$ $SS_2/D = 1$
t4	R/X	$s_1/A = 2$ $s_1/B = 1$	$s_1/A = 2$ $s_1/B = 1$		R			inv. inf after RFB $SS_1/A = 2$ $SS_1/B = 2$ and from D $SS_2/C = 0$ $SS_2/D = 1$	inv. inf	

4 Conclusion

While the problem of dynamic placement and load balancing is rarely addressed in database and information systems research [**?**, **?**], the approach based on learning and agents cooperation is a recent one. Schaerf et Al [**?**] have studied the interaction of various parameters and their effect on the system efficiency. In Arcadia [**?**], dynamic placement is mainly based on the cooperation of two agent types, ”system agents”, and ”application agents”. Both approaches usually assume some control of the system, in that sense that it is possible to migrate a process on another site to balance the load. They also rely on several indicators such as the number of messages received on a site. This makes them inapplicable in the context of Internet. The strategies for dynamic query placement which we have developed in this paper are part of the U-Doc project, and assume a worldwide distribution of the information system at Internet scale. The rely on the approach of multiple agents organized in a community of cognitive agents and a community of reactive agents. In order to optimize dynamic query placement, we mainly use the knowledge a *reactive agent* obtains during its experiences and also allowing their cooperation to share informations. The response time and the throughput are the only parameters that the cognitive agents use to build their knowledge through their experiences. In order to enable complex services with large added value, subcontractor agents are also proposed. They implement service composition becoming usual, as shows our own project which can be viewed as a *sub-contractor* agent composed of bibliographic database servers and searcher servers. Our approach alleviate the impossibility of actual servers of reply a request for bid, take advantage of learning of agents to reduce the biding negotiation process. We have developed these strategies of allocation taking into a count the actual conditions of our project, and simulated them to uplight their performance. The results of our simulations are positives and will be validated actually in our U-Doc project.