Distributed Optimisation using the Mobile Agent Paradigm through an Adaptable Ontology: Multi-operator Services Research and Composition

Hayfa Zgaya and Slim Hammadi
LAGIS UMR 8146 – Ecole Centrale De Lille
France

1. Introduction

Giving transport customers relevant, interactive and instantaneous information during their travels, represents a real challenge according to the exponential growth of services available on large distributed networks. Unfortunately, distributed applications through wide networks are not easy to realize because of the limited aspect of bandwidth that remains restricted and also because of a high incidence of network errors (bottleneck, failure, crash…). Our goal is to properly access and share distributed data located in an Extended Transport Multimodal Network (ETMN). In this context, mobile technology (Pharm & Karmouch, 1998; Theilmann & Rothermel, 1999) can complement artificial intelligence because it can reduce considerably network traffic (Carzaniga et al., 1997). Giving the mobility character to a software agent will allow him to migrate towards any node on the network that can receive mobile entities. Nodes to be visited by a Mobile Agent (MA) correspond to his route called Workplan. Many researchers have long discussed the benefits of the MA paradigm and conclude that it might be efficient in some cases (Picco & Baldi, 1997; Buse et al., 2003). In a recent work (Zgaya & Hammadi, 2006b), we demonstrated that using the MA paradigm in a Transport Multimodal Information System (TMIS) to collect needed data, is widely beneficial than using classical paradigms such as the Client Server (CS) one, if we use an optimization approach. The verification was successful thanks to a two-level optimization approach (Zgaya et al., 2005a, 2005b) that optimises, using metaheuristic, the total number of mobile entities and their different Workplans through the ETMN. However, some network errors (bottleneck, failure, crash…) can occur during the moving of MAs through the network nodes. In our work, we define a MA negotiation process in order to reassign non-attributed services, to available network nodes. Therefore we designed a flexible transport ontology that allows an easy handling of the terms and messages for negotiating. The remainder of this chapter is organized as follows: the problem complexity and the correspondent general formulation are presented in the next section. The global architecture of the Multi-Agent System (MAS) is proposed in section 3 and the optimisation approach in section 4. The proposed negotiation protocol is specified in section 5, followed by the used flexible transport ontology in section 6. Simulations are given in section 7 and finally the conclusion and prospects are addressed in last section.

Source: Multiagent Systems, Book edited by: Salman Ahmed and Mohd Noh Karsiti,
ISBN 978-3-902613-51-6, pp. 426, February 2009, I-Tech, Vienna, Austria

2. Problem formulation

The main concern of a TMIS is to satisfy users, respecting the delays of the responses (due dates) and minimizing their costs; this is a two-step optimization problem: firstly the assignment of an effective set of MAs to all existent network nodes. This assignment builds initial Workplans of the MAs in order to explore, in an optimal manner, the ETMN entirely. The second step corresponds to the best assignment of a sub-set of the ETMN nodes to identified tasks, deducing final Workplans. The selected sub-set of nodes corresponds to the possible providers to the identified tasks. A single identified task corresponds to an independent recognized sub-request which belongs to one or several requests formulated simultaneously by one or different customers through different devices (laptop, PDA…). More precisely, a single task can correspond to a transport service (sub-route, well-known geographical zone…) or to a related service (cultural event, weather forecast…). After the decomposition process, information providers (distant nodes), which propose services to the correspondent identified tasks, are recognized (fig. 1). Finally, nodes must be assigned to tasks in order to satisfy all connected users. A user is satisfied if his request was answered

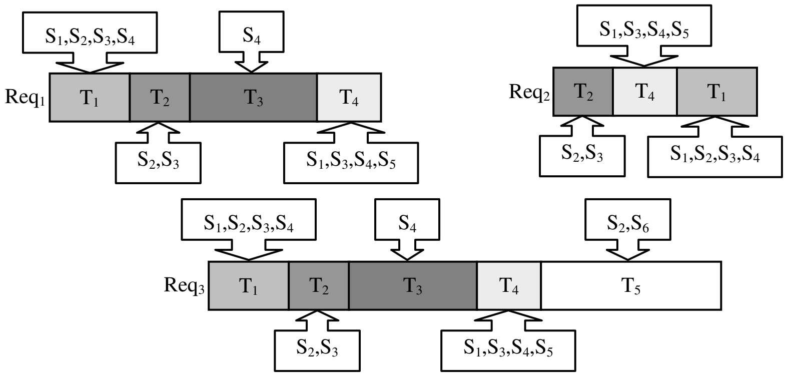

Fig. 1. Nodes identification

rapidly with a reasonable cost. This problem is called the Distributed Tasks Assignment Problem (DisTAP) and defined by:

- R requests, waiting for responses at the same instant *t*. The set of these requests is noted by R_t,
- The set of independent I tasks, representing all available services on the ETMN, is noted by $T=\{T_1,\ldots, T_I\}$,
- Each request $req_w \in R_t$ $(1 \leq w \leq R)$ is decomposed into a set of independent tasks, noted by $I_{t,w}=\{T_{r_1},\ldots,T_{r_{nj}}\}$ $(1 \leq n_j \leq I$ and $I_{t,w} \subseteq T)$,
- The set of independent I′ tasks (I′≤I), composing globally R_t, is noted by I'_t ($I'_t \subseteq T$ and $\bigcup_{w=1}^{R} I_{t,w} = I'_t$),
- Each request req_w has a due date d_w initially known, an ending date D_w and a total cost C_w,
- The realization of each task $T_i \in T$ requires a resource, or node, selected from a set of J registered nodes in the ETMN, noted by $S=\{S_1,\ldots, S_J\}$,
- The set of J′ nodes (J′≤J), selected from S to perform I'_t, is noted by S′ (S′⊆S),
- There is a predefined set of processing time; for a given node S_j and a given task T_i, the processing time of T_i using the resources of S_j, is defined and noted by $P_{i,j}$,

- There is a predefined set of information cost; for a given node S_j and a given task T_i, the cost of the information to collect from S_j, corresponding to the service referenced by T_i, is defined and noted by $Co_{i,j}$,
- The size of the collected data to ensure a service is defined; for a given node S_j and a given task T_i, the data size is defined and noted by $Q_{i,j}$,
- We have partial flexibility; the realisation of each task T_i requires a node selected from a set of nodes, which propose the same service performing the task T_i, with different cost, processing time and data size.

The three characteristics described above, namely ($P_{i,j}$;$Co_{i,j}$;$Q_{i,j}$), represent successively the first, second and last term of each element of what we call a service table (table 1).

	S_1	S_2	S_3	…	S_J
T_1	(0;0;0)	(0.2;5;3)	(0.4;3;3)		(0.2;5;3)
T_2	(0.2;4;5)	(0.1;5;2)	(0.4;5;1)		(0.3;8;3)
T_3	(0.1;0;3)	(0;0;0)	(0.2;0;3)		(0.4;2;2)
T_4	(0.3;2;1)	(0.3;1;1)	(0;0;0)		(0,0,0)
…					
T_I	(0.2;3;1)	(0.1;1;3)	(0.4;5;2)		(0.4;5;3)

Table 1. Example of a service table

We notice that if a provider does not offer a response to a task (partial flexibility); the correspondent term in the table above is (0,0,0). Otherwise we have $P_{i,j}\neq0$, $Co_{i,j}\neq0$ and $Q_{i,j}\neq0$. It is also possible to have $P_{i,j}\neq0$, $Q_{i,j}\neq0$ and $Co_{i,j}=0$ for a free information in the case of a promotional operation. In order to situate the complexity of our problem, an analogy was performed (table 2) between the problem described above (DisTAP) and the well-known Flexible Job Shop Problem (FJSP).

FJSP	DisTAP
N jobs	R requests
M machines	J servers
n_j non preemptable ordered operations /job_j	n_j non preemptable ordered tasks /$request_j$
The problem: to organize the execution of N jobs on M machines	The problem: to organize the execution of R requests on S servers
The execution of each operation i of a job j ($O_{i,j}$) requires one resource or machine selected from a set of available machines	The execution of each task i (T_i) of a request j (req_i) requires one resource or server selected from a set of available servers (similarities of requests)
The assignment of the operation $O_{i,j}$ to the machine M_k entails the occupation of this machine during a processing time called $d_{i,j,k}$	The assignment of the task T_i to the server S_k entails the occupation of this server during a processing time called $P_{i,k}$
At a given time, a machine can only execute one operation: it becomes available to other operations one the operation that is currently assigned to is completed (resource constraints).	At a given time, a server can only execute one task: it becomes available to other tasks one the task that is currently assigned to is completed (resource constraints).

Table 2. Analogy between FJSP and our problem

In DisTAP, we manage the similarities of requests in order to avoid the same data research. Besides, we have to assign the servers to tasks as well as to assign MAs to remote nodes (servers), taking into account the network state. Therefore our problem presents more difficulty than the FJSP which has been shown to be NP-hard. In addition, the distributed character of our system and the requirement to cooperate different autonomous static and mobile entities, confirm the choice of a multi-agent architecture for our system.

3. The multi-agent system

To resolve the problem described previously, we propose a system based on the coordination of five kinds of software agents (fig. 2):

- Interface Agent (IA): this agent interacts with the user of the system allowing him to choose an appropriate form of response to his demand, so this agent manages the request and then displays the correspondent result. Therefore, when a user accesses to the TMIS, an agent IA deals with the formulation of his request and then sends it to an available identifier agent. This one relates to the same platform to which several users can be simultaneously connected, thus he can receive several requests formulated at the same time,
- Identifier agent (IdA): this agent manages the decomposition of the requests that were formulated through a same short period of time ε* (ε-simultaneous requests). The decomposition process generates a set of sub-requests corresponding, for example, to sub-routes or to well-known geographical zones. Sub-requests are elementary independent tasks to be performed by the available set of distributed nodes (information providers) through the ETMN. Initially, each node must login to the system registering all proposed services knowing that a service corresponds to the response to a defined task with fixed cost, processing time and data size. Therefore, an agent IdA decomposes the set of existing simultaneous requests into a set of independent tasks, recognizing possible similarities in order to avoid a redundant search. The decomposition process occurs during the identification of the information providers. Finally, the agent IdA transmits cyclically all generated data to available scheduler agents. These ones must optimize the selection of providers, taking into account some system constraints,
- Scheduler Agent (SA): several nodes may propose the same service with different cost, processing time and data size. The agent SA has to assign nodes to tasks, minimizing total cost and total processing time to respect due dates (data constraint). Selected set of nodes corresponds to the sequence of nodes which build the Workplans (routes) of the collector agents. An agent SA has firstly to optimize the number of collector agents before assigning nodes to tasks.
- Intelligent Collector agent (ICA): an agent ICA is a mobile software agent who can move intelligently from a node to another through a network in order to collect needed data and finally returns to his home node, noted by H. This special kind of agent is composed of data, code and a state and has an intelligent behaviour. Collected data should not exceed a capacity threshold in order to avoid the overloading, so the agent SA has to take into account this aspect when assigning nodes to tasks. When they come back to the system, the agents ICA must transmit collected data to the available fusion agents,
- Fusion Agent (FA): the agents FA have to fusion correctly collected data in order to compose responses to the simultaneous requests. The fusion procedure needs information on behalf of IdA and SA agents and progresses according to the collected data availability. Each new answer component must be complementary to the already merged ones.

* Fixed by the programmer

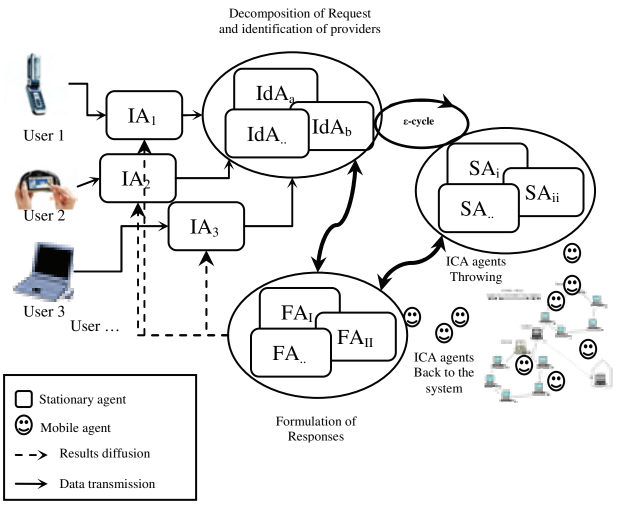

Fig. 2. System architecture

The needed data, required to satisfy the demands of the customers, are distributed through the ETMN and their collect corresponds to the jobs of ICA agents. Consequently, the SA agents have to optimize the assignments of the nodes to the identified tasks, minimizing total cost and response time. To this problem, we propose a two-level optimization solution (Zgaya et al., 2005a, 2005b) corresponding to the complex behaviour of the SA agents.

4. Scheduler Agent behaviour

The SA agent has two different basic behaviours: firstly the generation of an effective number of ICA agents in order to explore the ETMN entirely. This behaviour starts each time the network state varies considerably, in order to prepare the initial Workplans of ICA agents. Thus, we assume the existence of a network module that provides information to the system about the latest variations. The second behaviour is an ε-cyclic one, supervising the reception of the required services and their possible information providers, identified by the agents IdA. This second behaviour optimizes the assignments of nodes to the tasks in order to deduce the final Workplans of ICA agents from initial ones. The SA agents have to interact in order to share information and negotiate the different part of assignment for a global optimisation. For this problem, we just underline that we propose a solution using the formation of coalitions approach but this is not the topic of this chapter. For example, in the case of possible overlapping of the simultaneous requests, concerning SA agents have to gather, forming coalitions, in order to share the assignments about the different identified similarities. Hence, we focus here on the individual behaviour of a SA agent apart from his interaction with the other agents. We describe the two individual behaviours mentioned above in what follows.

4.1 Generation of the initial workplans

The Cost-Effective Mobile Agent Planning (CE-MAP) Algorithms, suggested by (Baek et al., 2001), are the most appropriate to our problematic. In fact, a proposed dynamic algorithm, called BYKY2, optimizes the number of MAs minimizing the total execution time, taking into account the network state. In a previous work (Zgaya, 2005a), we adopted the same approach but we considered the transported data and then the state variation of mobile entities. The MA Workplan problem is described in what follows, assuming some hypothesis:

- Collecting data on a visited node requires a processing time. We suppose that the size of the data to collect from a network node is equal to the average of the total data size on this node,
- Initially, we assume that an ICA agent is not totally empty because it contains an initial quantity of data Q_0,
- We suppose that minimal latency between each pair of nodes in the network is available tanks to an existent network monitoring module,
- Information can have a multimedia aspect, so we assume that the transmission of a quantity of data from a node to another depends on current latency.

4.1.1 Description

The MA Workplan problem can be described as follows: ICA agents are created and initially launched from an originally node (Home node). The other network nodes represent available information providers where an ICA agent can move to collect data corresponding to the claimed services. These ones are expressed in term of independent tasks. A same service can be proposed by different nodes with different cost, processing time and with different formats. We call a response time to a service on a node, the processing time of the correspondent task to this service on this node. So the response time on the Home node is null. Latencies are known and may affect navigation time of ICA agents. Our goal is to minimize the number of ICA agents and their navigation time in order to explore all the ETMN, taking into account network state. Initially, we introduce some definitions using variables described in table 3.

Variable	Description
m	Number of ICA agents
$ICA_1,\ldots,ICA_m$	Identifiers of ICA agents
H	Home node
$Wk_{i,p}$	Nodes sequence representing the Workplan of an ICA_i agent: ($S_{i_1}, \ldots, S_{i_p}$) with $1 \le p \le J$
$T(Wk_{i,p})$	Routing time for $Wk_{i,p}$
$Qte_{k,u}$	The size of the transported data by the agent ICA_k until the node S_u included
$Tr(Qte_{k,u},S_u,S_v)$	Transmission time for $Qte_{k,u}$ from node S_u to node S_v
CT_j	Processing time on node S_j for the data quantity Qt_j
Qt_j	Data quantity on node S_j
$d(S_i,S_j)$	Data transfer rate between nodes S_i and S_j

Table 3. Notations

Definition 1: CT_j (Processing time on node S_j) corresponds to needed computing time on the node S_j to extract the data quantity Qt_j.
Definition 2: Qt_j (Data quantity on node S_j) it is the average data size to extract from S_j.
T_i represents a task corresponding to a service proposed by S_j. Therefore:

$$Qt_j = \frac{\sum_{i=1}^{I} a_{i,j} Q_{i,j}}{\sum_{i=1}^{I} a_{i,j}} \qquad (1)$$

$$CT_j = \frac{\sum_{i=1}^{I} a_{i,j} P_{i,j}}{\sum_{i=1}^{I} a_{i,j}} \tag{2}$$

Where $a_{i,j}$ is a Boolean value as follows: $a_{i,j}=1$ if the node S_j proposes a service for the task T_i and $a_{i,j}=0$ otherwise (according to the given service table). Moreover, we remind that $P_{i,j}$ corresponds to the processing time of a given task T_i on the node S_j (section 2).

Definition 3: $Qte_{k,u}$ (Data quantity transported until S_u by ICA_k) corresponds to the size of the collected data by the agent ICA_k during his route, until the node S_u included.

$Qte_{k,u}$ is calculated by:

$$Qte_{k,u} = Q_0 + \sum_{r=1}^{u} Qt_{k_r} \tag{3}$$

We remind that Q_0 corresponds o the initial quantity of data within an ICA agent (parag.4.1).

Definition 4: $Tr(Qte_{k,u},S_u,S_v)$ (Transmission time) needed time for the agent ICA_k to migrate from S_u to S_v transporting the data quantity $Qte_{k,u}$.

$Tr(Qte_{k,u},S_u,S_v)$ is computed like this:

$$Tr(Qte_{k,u}, S_u, S_v) = \frac{Qte_{k,u}}{d(S_u, S_v)} \tag{4}$$

Definition 5: $T(Wk_{k,p})$ (Routing time for $Wk_{k,p}$) needed time for the agent ICA_k to visit the sequence of network nodes $(S_{k_1},\ldots,S_{k_p})$ with $1 \le p \le J$.

$T(Wk_{k,p})$ is computed like this:

$$T(Wk_{k,p}) = T_{go}(k, p=1) + T_{travel}(k,p) + T_{return}(k,p) \tag{5}$$

	$Wk_{k,p}$	T_{go}	T_{return}	T_{travel}
p=1	(S_{k_1})	$Tr(Q_0,H,S_{k_1})$	$Tr(Qte_{k,p},S_{k_p},H)$	CT_{k_1}
1<p≤J	$(S_{k_1},\ldots,S_{k_p})$			$X_{k,p}$

Table 4. Routing Time

With:

$$X_{k,p} = \sum_{i=1}^{p} CT_{k_i} + \sum_{i=1}^{p-1} Tr(Qte_{k,i}, S_{k_i}, S_{k_{i+1}}) \tag{6}$$

4.1.2 Proposed workplan schemes

To propose a cost-effective Workplan MA scheme, we assume that a monitoring module exists in the system providing information about the network status (latency, bandwidth, traffic, bottleneck, failure…). Therefore, we can get data transfer rate values among all pairs of nodes through the ETMN. The goal is to find an effective set of ICA agents minimizing their navigation time, in order to explore all the ETMN nodes, taking into account network state. It is clear that sending an ICA agent to each node gives us the best total computing time because, in this case, agents are launched simultaneously into each network node. Therefore, we keep this best total computation time to build nodes partitions, minimizing the number of ICA agents. Consequently, we just care about the data size and the processing time in the service table (section 2), ignoring the data cost. As described previously, $d(S_i,S_j)$ namely data transfer rate among two network nodes S_i and S_j, is available. We give here a brief description of the algorithm detailed in (Zgaya et al., 2005a):

The initial Workplan algorithm description
- ***Step 1***: Sort the nodes in decreasing order according to their correspondent routing time $T(Wk_i=S_i)$ $\forall 1 \le i \le J$. Set the threshold δ which is the routing time of the first node in the sorted list: $$\delta = \max_{1\le i\le J}(T(Wk_i = S_i)) \quad (7)$$
- ***Step 2***: Partition the given network into several parts by gathering nodes so that the routing time of each part does not exceed the threshold δ.

This proposed dynamic algorithm tries to find the next node to visit from the current position where the agent resides. In other words, this algorithm looks for the next node for a part calculating, each time, the new routing time. A node is selected if the new routing time does not exceed the threshold δ. Otherwise, a Workplan is ready to be assigned to an ICA agent and the algorithm ends if each available node belongs to a Workplan. The algorithm distributes all available nodes to a set of *m* ICA agents in order to explore the network entirely. Each built route corresponds to the initial Workplan of the correspondent ICA agent. Then, final Workplans will be deduced from initial ones thanks to our evolutionary approach described in next section. This will be done by selecting a subset S′ from S (the total number of available nodes in the ETMN) in order to optimise the management of the data flow through the network. Thus, some nodes will not be selected from S what can decrease the total number *m* of ICA agents. This will be happen when all the nodes composing the initial Workplan of an agent ICA are not selected. Let *m*′ be the new number of ICA agents. We have also $J'=|S'|$ the new number of nodes so $m' \le m$, $J' \le J$ and $S' \subseteq S$. Thanks to the generated final Workplans, required data will be collected in an effective manner, in order to reach as soon as possible and with reasonable costs, the best schedule of the simultaneous requests.

4.2 Composition of services using an evolutionary approach

The Evolutionary Algorithms (EA), inspired from genetic algorithms, added a new aspect to the field of artificial intelligence. These algorithms use various computational models of evolutionary processes to solve problems on a computer. EA are stochastic search methods that mimic the metaphor of natural biological evolution; they operate on a population of potential solutions applying the survival principle of the fittest results, in order to produce successively better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the population, then breeding them together using genetic operators such as crossover and mutation. Compared to traditional optimization methods such as gradient descent, EA are robust and global search technique. For this reason, the scheduling community has been quick to realize the potential of EA. In this section, we use an evolutionary approach to resolve our assignment problem. Therefore, we use some aspects that must be clarified:

- A specific genetic representation (or encoding) appropriate to the problem, to determine feasible solutions of the scheduling optimization problem,
- Original genetic operators that alter the composition of children during the reproduction. As it was mentioned previously, a task (sub-request) must be managed by only one provider selected from the set of nodes that propose the correspondent service. Therefore, we choose to correct generated solutions in order to respect this constraint. Consequently, each crossover or mutation operation must be followed by a correction process,
- Parents are selected randomly from current population to crossover/mutation with some probability p_c/p_m ($0 < p_c, p_m < 1$). We believe that this technique gives more chance to weak individuals to survey,
- A non-elitist replacement technique is adopted to generate the new population from the previous one,
- The evaluation functions estimate a possible solution according to two criteria: the cost and the delay.

4.2.1 The representation of an evolutionary solution

The research and the composition of distributed transport services are generated thanks to an evolutionary algorithm, managed by the active SA agents in the system. We notice that the selection of an appropriate representational scheme of a solution is fundamental to the success of EA applications. Therefore, in a previous work (Zgaya et al., 2005b), we designed an efficient coding (possible solution) for the chromosome respecting our problem constraints. Thus, we propose a flexible representation of the chromosome called Flexible Tasks Assignment Representation (FeTAR). The chromosome is represented by a matrix CH(I′×J′) where rows represent independent tasks (the services), composing globally simultaneous requests and columns represent identified nodes (the providers). Each element of the matrix specifies the assignment of a node S_{c_j} (1≤j≤J′) to the task T_{c_i} (1≤i≤I′) as follows:

Value of CH[i,j]	Condition
1	S_{c_j} is assigned to T_{c_i}
*	S_{c_j} may be assigned to T_{c_i}
X	S_{c_j} cannot be assigned to T_{c_i}

We notice that each task must be performed by a single node, selected from the available set of nodes that propose the service corresponding to a response to the concerned task. Indeed, the assignment and the scheduling of all distributed nodes to the required services, represent the optimisation of the services composition that provide transport customers effective responses to their requests.

4.2.2 The genetic operators

(a) The crossover algorithm

Crossover involves combining elements from two parent chromosomes into one or more child chromosomes. The role of the crossover is to generate a better solution by exchanging information contained in the current good one. The following algorithm specifies the crossover operator:

```
***CrossFeTAR*** Algorithm
The creation of $C_1$ (resp. $C_2$) representing the child 1 (resp. child 2) is given by:
-   Step 1: Choose randomly two parents and one node; suppose that $P_1$, $P_2$ and $S_{c_j}$ ($1 \le j \le J'$) are
    randomly selected,
-   Step 2: Tasks assignment of $S_{c_j}$ in $C_1$ (resp. $C_2$) must correspond to the same assignment of
    $S_{c_j}$ in $P_1$ (resp. $P_2$),
-   Step3:
    k :=1 ;
    while ($k \le J'$) and ($k \ne j$) {
      Tasks assignment of $S_{c_k}$ in $C_1$ (resp. $C_2$) corresponds to the same assignment of
      $S_{c_k}$ in $P_2$ (resp. $P_1$);
      k := k + 1 ;
    }
-   Step 4: if ($C_1$ (resp. $C_2$) is not a feasible solution)
            Correct randomly $C_1$ (resp. $C_2$);
```

We notice that sometimes, a generated solution resulting from a crossover process is not feasible. If it is the case, we propose a correction process that changes, in a random way, a non-feasible solution to a feasible one knowing that a feasible solution is a FeTAR instance that assigns each task composing it, only once. The algorithm ***CorrectFeTAR*** illustrates this correction process as follows:

```
CorrectFeTAR Algorithm
The correction of the FeTAR instance CH is given by:
for (i:=1; i ≤I'; i:=i+1){
    initialize to zero the vectors IndexAssigned[] and IndexNotAssigned[] of dimensions J';
    k1:=0; k2:=0;
    for (j:=1; j ≤J'; j:=j+1){
       if( $CH[c_i, c_j]$=1) {
         k1:=k1+1;
         IndexAssigned[k1]:=j;
       }else if( $CH[c_i, c_j]$=*) {
         k2:=k2+1;
         IndexNotAssigned[k2]:=j;
       }
    }
    if(k1=0) {
       Draw randomly an index p with 1≤p≤k2 ; s:=IndexNotAssigned[p];
       $CH[c_i, c_s]$:=1;
    }else if(k1>1) {
            Draw randomly an index p with 1≤p≤k1 ; s:=IndexAssigned[p];
            for (x:=1; x ≤J'; x:=x+1){
            if( $CH[c_i, c_x]$=1 et x≠s)
                $CH[c_i, c_x]$:=*;
          }
}
```

For example, we suppose that a Crossover process generated, from two FeTAR instances parents P_1 and P_2, two new FeTAR instances childs C_1 and C_2 like so:

P_1	S_{12}	S_6	S_3	S_{24}
T_1	X	*	*	1
T_5	*	*	1	*
T_3	1	X	*	*
T_9	*	*	1	X
T_2	*	*	1	*

P_2	S_{12}	S_6	S_3	S_{24}
T_1	X	*	1	*
T_5	*	*	1	*
T_3	*	X	*	1
T_9	*	*	1	X
T_2	*	*	1	*

C_1	S_{12}	S_6	S_3	S_{24}
T_1	X	*	*	*
T_5	*	*	1	*
T_3	*	X	*	1
T_9	*	*	1	X
T_2	*	*	1	*

C_2	S_{12}	S_6	S_3	S_{24}
T_1	X	*	1	1
T_5	*	*	1	*
T_3	1	X	*	*
T_9	*	*	1	X
T_2	*	*	1	*

Both C_1 and C_2 are not feasible solutions because C_1 does not assign the task T_1 and C_2 assigns the task T_1 more than one time (T_1 is assigned twice by S_3 and S_{24}). After correction, C_1 will be randomly C_{1x}, C_{1y} or C_{1z} and C_2 will be randomly C_{2x} or C_{2y} like so:

C_{1x}	S_{12}	S_6	S_3	S_{24}
T_1	X	1	*	*
T_5	*	*	1	*
T_3	*	X	*	1
T_9	*	*	1	X
T_2	*	*	1	*

C_{1y}	S_{12}	S_6	S_3	S_{24}
T_1	X	*	*	1
T_5	*	*	1	*
T_3	*	X	*	1
T_9	*	*	1	X
T_2	*	*	1	*

C_{1z}	S_{12}	S_6	S_3	S_{24}
T_1	X	*	1	*
T_5	*	*	1	*
T_3	*	X	*	1
T_9	*	*	1	X
T_2	*	*	1	*

C_{2x}	S_{12}	S_6	S_3	S_{24}
T_1	X	*	*	1
T_5	*	*	1	*
T_3	1	X	*	*
T_9	*	*	1	X
T_2	*	*	1	*

C_{2y}	S_{12}	S_6	S_3	S_{24}
T_1	X	*	1	*
T_5	*	*	1	*
T_3	1	X	*	*
T_9	*	*	1	X
T_2	*	*	1	*

(b) **The mutation algorithm**

Mutation represents another important genetic operator. Although mutation is important, it is secondary to crossover. It introduces some extra variability into the population and typically works with a single chromosome to create a new modified one. The mutation algorithm is presented as follows:

```
MuteFeTAR Algorithm
-   Step 1: Choose randomly one chromosome CH, one task $T_{c_i}$ (1≤i≤I') and one node
    $S_{c_j}$ (1≤j≤J');,
-   Step 2:
  if(CH[i,j]= *){
            Find $j_1$ with 1≤$j_1$≤J' and CH[i,$j_1$]=1 ;
            CH[i,$j_1$] := * ;
            CH[i,j] := 1 ;
} else
      if(CH[i,j] = 1 and ∃ $j_1$ / 1 ≤ $j_1$ ≤ J' and CH[i,$j_1$]=*){
                                        CH[i,$j_1$] := 1 ;
                                        CH[i,j] := * ;
}
```

For example, if the chromosome C_{1x} undergoes a mutation process, muted C_{1x} may be C'_{1x} like so:

C'_{1x}	S_{12}	S_6	S_3	S_{24}
T_1	X	1	*	*
T_5	*	*	1	*
T_3	*	X	*	1
T_9	*	*	1	X
T_2	1	*	*	*

With the mutation point (T_2,S_3).

4.2.3 Evaluation functions

At each iteration, individuals (chromosomes) in the current population are evaluated according to the same measure of fitness. There are a number of characteristics of the evaluation function that enhance or hinder the evaluation of a program performance. In our case, the fitness function intends to maximize the number of satisfied transport travellers, minimizing response delay and total cost. In other words, a chromosome is firstly evaluated according to the number of responses respecting due dates, then according to the average of total costs. Thus, a chromosome has to express ending responses date and the information cost (Zgaya et al., 2005b). The first evaluation function, called Fitness_1, computes the ending dates of all the requests according to the generated FeTAR solution, in order to deduce the number of satisfied users in term of response time. Then the second evaluation function, called Fitness_2, computes the total cost of each request. As we previously mentioned, a request req_w (1≤w≤R) is decomposed into $I_{t,w}$ tasks and the algorithm Fitness_1 computes the total processing time D_w for each req_w. This time does not include only the effective processing time on the nodes because we have to take into account the routing time of ICA agents. For that, we assume that, the ending date D_w (EndReq[w]) corresponding to the total execution time of a request req_w, includes also some value noted by γ which is the average navigation time of ICA agents (Zgaya et al.,2008). Besides, the total cost C_w (EndReq[w]) is computed for each request req_w by the algorithm Fitness_2.

Fitness_1 **Algorithm**

```
- Step 1: Initialisation ($1 \le k \le m$)
  - Initialize to Ø each set of tasks $U_k$ which should be performed by each $ICA_k$
  - Initialize to $\gamma$ the total time EndU[k] to perform each set of tasks $U_k$.
- Step 2: Compute the set of tasks $U_k$ performed by each $ICA_k$ and the total time
  to perform them as follows:
        for (i:=1; i ≤ I'; i:=i+1) {
          Find k and j while $ICA_k$ performs $T_{c_i}$ on $S_{c_j}$;
          $U_k = U_k \cup \{T_{c_i}\}$ ;
          EndU[k] := EndU[k] + $P_{c_i,c_j}$ ;
        }
- Step 3: Compute ending time for each request i: EndReq[w] with $1 \le w \le R$, by
  looking for each task composing this request. An ending time of a request is
  the maximum necessary time for all the agents ICA responsible for all the
  tasks composing this request, in order to carry out their Workplan. Ending
  time for each request is computed as follows:
  for (w:=1; w≤R; w:=w+1){
   Initialize to false TreatedICA[k] for each $ICA_k$ ($1 \le k \le m$);
        EndReq[w] = 0;
        for (j :=1; j ≤I'; j:=j+1){
          if ($T_{c_j} \in req_w$) {
            k:=1;
            while ((k ≤ m) and ($T_{c_j} \notin U_k$)) k:=k+1;
                if (not TreatedICA[k]) {
                  EndReq[w]:=max(EndReq[w], EndU[k]);
                  TreatedICA[k] = true;
            }
          }
        }
  }
```

Fitness_2 **Algorithm**

```
The total cost for a request is the total cost of all independent tasks composing
this request. It is calculated as follows:
 Step 1: Initialisation ($1 \le w \le R$)
Initialize to 0 CostRe[w] for each $req_w$
 Step 2:
   for (w:=1; w≤R; w:=w+1){
for each $T_{c_i} \in I_{t,w}$ {
 Find j/ $S_{c_j}$ assigns $T_{c_i}$ in the FeTAR instance CH;
  CostRe[w]= CostRe[w] + $Co_{c_i,c_j}$ ;
}
   }
```

The evaluation of a chromosome is illustrated by a vector, which express, for each request w (req_w), its required total time for the execution (D_w) and also its total cost (C_w). From the generated vector, we deduce the average total cost C_{av} and the maximum ending date D_{max} of all the requests managed by the chromosome.

4.2.4 Best solution

To determinate the best solutions, we adopted an elitist approach (Zgaya & Hammadi, 2008) using an external storage to memorise the most adapted individuals during the search. The evolutionary adopted approach is shown in fig. 3. During the evaluation process to crossover and mutation, best solutions are saved in external archives. Knowing that $d_{\max} = \max(d_w)_{1 \le w \le R}$, we discern two archive sets:

- Main solutions archive (M) representing best solutions which respect all due dates. In other words, a chromosome $CH \in M$ if and only if $\forall w$ ($1 \le w \le R$), $D_w \le d_{max}$. This archive set is decomposed into two sub-archives :
  - dominant solutions archive (d)
  - ε-dominant solutions archive (ε-d)

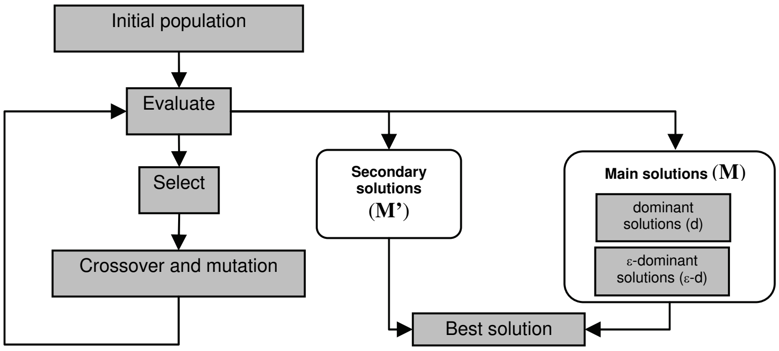

Fig. 3. Evolutionary approach

Secondary solutions archive (M') representing best solutions which exceed at least one due date. In other words, a chromosome $CH \in M'$ if and only if $\exists w$ ($1 \le w \le R$) and $d_{max} < D_w$.

We discerned two archive sets according to delay criterion satisfaction because we assume that if a response exceeds its due date, the user is not satisfied. That's why we consider that the first criterion has more priority that the second one. Consequently archive sets are firstly sorted according to delay criterion, then according to cost criterion. We notice by f1 the response delay function evaluated by Fitness_1 algorithm and by f2 the cost function evaluated by Fitness_2 algorithm. Considering two different solutions CH and CH', if $CH \in M$ and $CH' \in M'$ then CH dominates CH'. Otherwise, CH dominates CH' if and only if f1(CH)=f1(CH') and f2(CH)<f2(CH') . Each archive set has a maximum number size equals to the population size. If the number of individuals in an archive exceeds this fixed size, a crowding process must occur to decide which solutions must kept in the archive. The non-selected solutions are deleted; and the others contribute to the next selection procedure; archive members can then transmit their characteristics to offspring populations. M and M' archive sets represent generated solutions having minimum f1 and f2 values, so if a chromosome CH of the offspring dominates any archive member CH', the archive member

is deleted and the offspring is accepted. Fig. 4 represents the Pareto-optimal fronts with $\varepsilon_1=\varepsilon_2=\varepsilon=0.75$ (Zitzler & Thiele, 1998). We use a population size of N=100 with a crossover probability $p_c=0.8$ and mutation probability $p_m=0.2$.

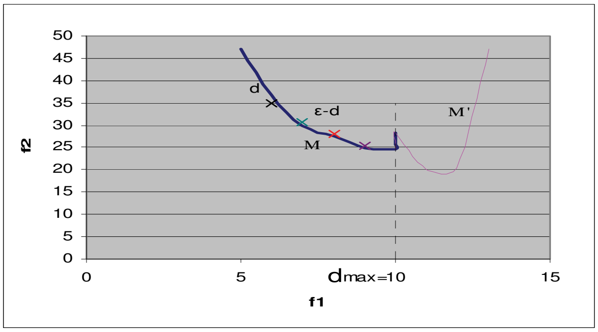

Fig. 4. Optimal fronts

4.2.5 Final workplans generation

According to the generated FeTAR instance CH, the equations (1) and (2) become:

$$Qt_{c_j} = \sum_{i=1}^{I'} (a_{c_i,c_j} \times Q_{c_i,c_j}) \quad (1')$$

$$CT_{c_j} = \sum_{i=1}^{I'} (a_{c_i,c_j} \times P_{c_i,c_j}) \quad (2')$$

With a_{c_i,c_j} a Boolean value as follows: if CH[c_i,c_j]=1 ($1 \le i \le I'$ and $1 \le j \le J'$) then $a_{c_i,c_j}=1$ else $a_{c_i,c_j}=0$. Routing time of the final Workplans are deduced using equations (1'), (2'), (3), (4), (5) and (6) according to table 4 (parag. 4.1.1).
Until this stage, the MA paradigm integrates a perfect structure without any incidence. But this is an improbable scenario because, in reality, errors may often occur (crash, bottleneck, failure…), especially in huge systems distributed through large networks. If it is the case, ICA agents have to interact with SA agents in order to negotiate the reassignments of potential cancelled services, keeping the whole robustness of the system. Consequently, we developed a negotiation protocol, described in next section.

5. The negotiation process

Some perturbations can occur through the network when ICA agents are following their correspondent final Workplans, according to the generated FeTAR instance. In this case, the ICA agents have to avoid unavailable nodes in their remained final Workplans. In addition, they have to change their itineraries in order to take into account the cancelled tasks that still need assignments because of the perturbation. Therefore, a new assignment process has to occur to find suitable new available providers. To do this, we have to benefit of active ICA agents who are still travelling through the network and to launch new ones otherwise. So ICA agents have to interact with SA agents in order to find suitable solution to the current situation. Thus, we propose a negotiation process inspired from the well-known contract net protocol (Smith, 1980) between ICA agents representing the participants of the negotiation and SA agents who are the initiators. In our proposed solution, we allow a partial agreement of the proposed contract (a FeTAR instance) from each ICA agent, to be confirmed partially or totally by the initiator of the negotiation (SA agent). A renegotiation process is necessary while there are still tasks that need reassignment. The purpose of this solution is to allow the ICA agents to cooperate and coordinate their actions in order to find globally near-optimal robust schedules according to their priorities, preferences and constraints, which depend on their current positions in their correspondent Workplans. Through the negotiation process tours, SA agents must assure reasonable total cost and time. In what follow, we describe in detail the proposed protocol. Firstly, we present a brief description of the initiators and participants of the negociation process.

5.1 Initiators and participants

An initiator of a negotiation is a SA agent who never knows the exact position of each travelling ICA agent. However, he knows all initial Workplans schemes and the final assignments of the servers (final effective Workplans). SA agent does not need to wait for all answers to take a decision, since he can accept a subset of responses to take pressing sub-decisions; urgent actions must be taken according to the current positions of ICA agents. Consequently, SA agent can take decisions every short period of time. In that case, he must update the set of services that need to be reassigned by providers through the confirmation step. After that, SA agent has to propose a new contract according to the updated services set and to the different capabilities of the participants of the negotiation. We suppose that errors on the network are identified before that an ICA agent leaves one functioning node toward a crashed one. A participant of a negotiation is an autonomous ICA agent who never knows anything about the other participants of the same negotiation process. Obviously, he knows his own initial Workplan scheme and his final assignments of servers (final effective Workplan). In addition, each ICA agent has his own priorities, preferences and constraints that are dynamic, depending on the network state and on his current position in the already defined final Workplan. Constraints of an ICA agent express tasks that he can't perform or servers he can't visit because they might cause problems (overloading, time consuming, high latency…). Priorities express servers where the ICA agent prefers visit because they are already programmed in his remained final Workplan. Finally, preferences express servers that are already programmed in the remained initial Workplan and not in the final one. Each time an ICA agent receives a new contract, he analyzes it to make a decision (refusal or total/partial acceptance).

5.2 The protocol

A protocol defines the language used by agents to exchange information. The proposed negotiation protocol (fig. 5) is characterized by successive messages exchanges between initiators corresponding to the agents who initiate a negotiation (SA agents) and participants of the negotiation (ICA agents). We designed our protocol so that a negotiation process can occur between several initiators and participants; it can be, for example, the case of simultaneous requests overlapping. Presently, we describe a negotiation protocol between a unique initiator and several participants. Negotiation always begins with the creation of a contract by the initiator agent, proposing it to active participants. The first contract corresponds to final Workplans that were already optimized thanks to our two-level optimization approach (Zgaya et al. 2005a, 2005b). A renegotiation means a round of modification request for a contract that "a part" has not been accepted the round before. In what follows, we show the adopted form for a communication before detailing the different exchanged messages between initiators and participants.

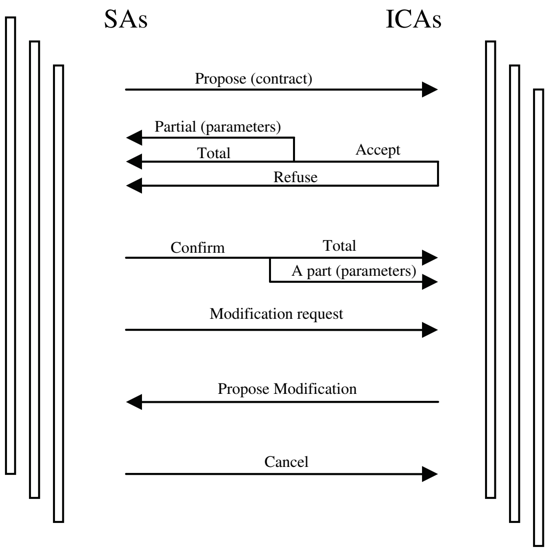

Fig. 5. The protocol

5.2.1 The agent message

We adopt the following structure for an agent message exchange:

<sender, receivers, service, perform, content, content-lang, ontology, f>

With:

- sender: the sender of the message,
- receiver: the list of receivers, they represent the recipients of the message,
- service: the "yellow-pages" service proposed by the receiver of the message,
- perform: the performative, which expresses the communicative intention,
- content: the information included in the message,
- content-lang: the content language, which represents the used syntax to express the content,
- the ontology: the vocabulary of the symbols used in the content and their meaning, used ontologies will be expressed in next section,
- f = <f1, f2, f3, f4, f5> represents some fields used to control several concurrent conversations and also to specify timeouts for receiving a reply. For the present, we don't assign this field but we just explain it for a best comprehension of message exchanges:
  - f1: reply-to *A*: the recipient of the message reply is the agent A,