Dynamic Resource Allocation in a MAS: A Case Study from the Industry

Daniela Briola, Viviana Mascardi, Maurizio Martelli
DISI, Università degli Studi di Genova
Via Dodecaneso 35, 16146, Genova, Italy
Daniela.Briola@unige.it
Viviana.Mascardi@unige.it
Maurizio.Martelli@unige.it

Riccardo Caccia, Carlo Milani
Ansaldo-STS, Italy
Caccia.Riccardo@ansaldo-sts.com
Milani.Carlo@ansaldo-sts.com

ABSTRACT

This paper describes the theoretic issues and the design of an implemented Multiagent System developed by DISI, the Computer Science Department of the University of Genova, and Ansaldo-STS, the Italian leader in design and construction of signalling and automation systems for conventional and high speed railway lines.
The problem discussed in this paper is a multiagent resource allocation problem where resources are modeled as nodes in a directed, non-planar graph that agents must traverse from one start point to one end point.
The goal of the multiagent system is to find a feasible allocation of resources to agents over time that emerges as the result of a sequence of local negotiation steps.
The multiagent system has been implemented using JADE and exploits the JADE Web Services Integration Gateway to access legacy applications developed by Ansaldo-STS. It has already been tested on real data and will be integrated into one of Ansaldo-STS's core business applications in a few months.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents, Multiagent systems

Keywords

Multiagent Resource Allocation, Industrial Application of MAS

1. INTRODUCTION

This paper describes the theoretic issues and the design of an implemented Multiagent System (MAS) developed by DISI and Ansaldo-STS. DISI (Dipartimento di Informatica e Scienze dell'Informazione) is the Computer Science Department of the University of Genova, IT. Ansaldo-STS is the Italian leader in design and construction of signalling and automation systems for conventional and high speed railway lines.

The real application for which the MAS has been developed is protected by a Non Disclosure Agreement. Thus, in this paper we provide a generalization of the problem that we addressed, we show how the complexity of this problem can be profitably faced following an agent-oriented approach, and we discuss the design of the developed MAS maintaining our description at the right level of abstraction. We provide some examples of the negotiation algorithm that we have developed and we discuss some tests we run.

The problem that the agents in our MAS solve is a classical Multiagent Resource Allocation (MARA) problem. As stated by [2],

> *Multiagent Resource Allocation is the process of distributing a number of items amongst a number of agents.*

The features of our MARA scenario are the following:

1. Resources are non-sharable, indivisible, and may be assigned to different agents in different and non intersecting time slots.
2. Agents have preferences on the bundle of resources they receive.
3. There is no single entity that decides on the final allocation of resources amongst agents, as in combinatorial auctions where the central entity is the auctioneer and the reporting of preferences takes the form of bidding. Instead, in our scenario allocations emerge as the result of a sequence of local negotiation steps.
4. The scenario is highly dynamic: resources may become unavailable because they break up, they may become available again because they have been fixed, and agents may enter and exit the system in any moment.
5. The objective of the negotiation is to find a feasible allocation. There are no requirements on the optimality of the solution which would contrast, in most cases, with the need of finding an allocation in a limited and pre-defined amount of time.

As demonstrated by hundreds of publications on MARA [2, 5, 6], many proposals already exist that might seem suitable to the features of our scenario. However, we were not able to find in the literature the right algorithm for solving exactly our problem. Many algorithms similar to the one discussed in this paper exist, but changing even only one assumption, requirement or constraint, leads to very different results. Thus, an algorithm "similar" but not identical to the one we needed, was not suitable for our purposes.

Also, we developed the MAS having a real application in mind. We could not neglect all the constraints on the software environment where our MAS would be integrated. Even if we had found

a description of the algorithm that would solve all our problems in some paper, we still had to cope with implementation issues, since, to the best of our knowledge, no negotiation algorithm similar to the one we needed is available to the research community as a source code.

For these reasons we decided to design and implement our own negotiation algorithm, even if we knew that its originality might be limited due to the large amount of literature on the topic.

At this point in time, the MAS implementation is almost complete; there are Interface Agents that correctly access services provided by legacy applications implemented by Ansaldo-STS and running outside the MAS, and the MAS has been successfully tested on real data. Some minor details of the behavior of the other types agents running in the MAS are still under refinement. The MAS has been implemented using JADE [1] and the legacy applications that is accesses expose their functionalities as Web Services integrated into JADE by means of the JADE Web Services Integration Gateway, WSIG [3]. In a few months the MAS will be integrated into the software module for which it has been developed. Then, the MAS will start working at a cracking pace.

The value of the research experience that we describe in this paper mainly consists in providing further support to what we stated in our WOA 2008 paper, talking about one previous (and definitely less challenging) joint project involving DISI and Ansaldo-STS [4]:

> *The role of academia in providing a good support during the design and implementation of MASs for real applications is a key factor in the take-off of the agent technology, and the joint DISI – Ansaldo-STS project discussed in this paper represents a success story in this direction.*

Thus, this paper reports another story on fruitful collaborations between university and industry.

The paper is organised in the following way: Section 2 provides a mathematical model of the problem; Section 3 describes how we reduced the mathematical problem to a MARA problem; Section 4 describes the allocation algorithm in detail; Section 5 illustrates how our negotiation algorithm works by means of two examples; Section 6 concludes and highlights some future directions for our work.

2. THE DYNAMIC RESOURCE ALLOCATION PROBLEM

The problem consists of

- A set of indivisible resources that must be assigned to different entities in different time slots (each resource can be used by only one entity in each time slot).
- A set of entities with different priorities, each needing to use some of the available resources for one or more time slots; entities have preferences over the set of resources they can obtain.
- A directed graph of dependencies among resources: an entity can start using resource R only if it used exactly one resource from $\{R_1, R_2, ..., R_n\}$ in the previous time slot (we represent these dependencies as arcs $R_1 \rightarrow R$, $R_2 \rightarrow R$, ..., $R_n \rightarrow R$ in the graph).
- A set of resources named "start points" that can be assigned to entities without requiring the prior usage of other resources (no arc enters in the corresponding node).
- A set of resources named "end points" that, once assigned to one entity, allow the entity to complete its job (no arc exits from the corresponding node).
- A set of couples of conflicting arcs in the graph of dependencies: an entity releasing R_1 for accessing R_2, where the usage of R_2 depends on the previous usage of R_1, might conflict with an entity releasing R_3 for accessing R_4. The two entities might indeed need to use the same transportation means for accessing R_2 from R_1 and R_4 from R_3 respectively, and the transportation means might be non sharable as well.
- A static allocation plan that assigns resources to entities for pre-defined time slots, in such a way that no conflicts arise.

In an ideal world where resources never go out of order and where any entity in the system can always access the resources assigned to it by the static allocation plan, no problems arise.

In the real world where entities happen to use resources for longer than planned and where resources can break up, a dynamic re-allocation of resources over time is often required. Thus, the solution of the real world problem is a dynamic re-allocation of the resources to the entities such that:

1. the re-allocation is feasible, namely free of conflicts; in our scenario, conflicts may arise both because two or more entities would want to access the same resource in the same time slot, and because two or more entities would want to use conflicting arcs in the same time slot;
2. the re-allocation task, whose output is either a feasible re-allocation or a message stating that no feasible re-allocations exist, is completed within a pre-defined amount of time;
3. each entity minimizes the changes between its new plan and its static allocation plan: the start and end point must always remain those stated in the static allocation plan, but the nodes in between may change, as well as the time slots during which resources are used;
4. each entity minimizes the delay in which it reaches the end point with respect to its static allocation plan;
5. the number of entities and resources involved in the re-allocation process is kept to the minimum;
6. in case no feasible re-allocation exists, the algorithm stops.

Goals 3 and 4 are local to the entities, whereas goal 5 is a global one. Achieving these three goals together is often impossible since the best re-allocation for one entity might cause the involvement of many other entities in the process, which contrasts with the last goal. Also, the re-allocation process must output either some conflict-free plan or a "no solution exists" message in a limited and pre-defined amount of time (goal 2), for functioning requirements of Ansaldo-STS's application. Thus, the plan resulting from the re-allocation problem, if any, may be sub-optimal.

2.1 A Model for the Problem

We modeled the problem as a directed and non-planar graph that entities must traverse from one start point to one end point. Nodes in the graph are labeled by resources[1] whereas arcs represent dependencies among them. We adopt a discrete and linear time model.

The fact that an entity stops in a node R for a given time slot TS corresponds to the usage of R during TS by the entity. Arcs $R_1 \rightarrow R$, $R_2 \rightarrow R$, ..., $R_n \rightarrow R$ in the graph mean that, in order for an entity to start using R at time T, exactly one resource among $R_1, R_2, ..., R_n$ had to be used by the same entity at time $T-1$[2].

When and entity traverses an arc $R_1 \rightarrow R_2$ it releases R_1 and accesses R_2. We assume that traversing an arc always requires one time unit. This is not true in the real application that we addressed, where the traversal time may vary, but this simplification made the problem easier to address. The usage of $R_1 \rightarrow R_2$ during TS makes $R_1 \rightarrow R_2$ and all the arcs that conflict with it, occupied for TS.

Entities enter the graph one after the other: in any time instant T, there is at most one entity showing up in one start point.

In order to keep our model as general as possible, we assume that there may be many different direct arcs connecting the same two nodes.

Entities need to traverse the graph from one start point to one end point. Entities can not change the start and end points established by their static allocation plan, but they are free both to find another path connecting these points and to modify the time slots during which they use some resources, if the original path stated by the static allocation plan is no longer available. The reasons that may cause an entity to change its original static allocation plan are:

1. a delay in entering the start node with respect to the original plan;
2. a node in the original path that is either no longer available because the corresponding resource is out of order, or not free in the needed time slot because another entity is using it;
3. an arc that the entity should traverse for moving from the current node to another node, and that is not free in the needed time slot.

Given this model, the problem to solve can be stated as:

> *For each entity that enters the graph from a start point either confirm the validity of the plan stated in the static allocation plan, or, if some unexpected event occurred that makes the original plan no longer applicable, find a new plan for reaching an end point of the graph. The new plan should minimize the delay in which the entity exits the graph and the number of required changes with respect to the original plan, as well as the number of entities involved in the re-allocation process.*

[1]In the sequel, we will sometimes use "node" and "resource" interchangeably, identifying a node by the resource that labels it.

[2]The resource in the set $R_1, R_2, ..., R_n$ might have been used by the entity for more than one time slot, namely from $T-n$ to $T-1$, $n > 1$. We are not interested in what happened before $T-1$. The important thing is that at time $T-1$, the entity used exactly one resource in the set.

3. REDUCING THE ALLOCATION PROBLEM TO NEGOTIATION IN A MAS

The problem described in Section 2 involves heterogeneous, autonomous, self-interested and distributed entities each with a partial knowledge of the environment where they live. Entities compete for obtaining the control of the resources needed for completing their job (i.e., exiting the graph), but they also need to cooperate in order to find a new feasible allocation that respects the rules imposed by the system (i.e., keeping the number of entities involved in the re-allocation as small as possible). These features make the dynamic resource allocation problem extremely suitable for being faced with an agent-oriented approach. Thus, we designed three different types of agents each with its own capabilities and view of the system:

- **Resource Agents (RA)**: each node in the graph is managed by one RAs in the MAS that knows the state of the node and the state of the arcs entering in it.
- **User Agents (UA)**: entities traversing the graph become UAs in the MAS.
- **Interface Agents (IA)**: agents responsible for the interactions with the environment outside the MAS are named IA.

The graph becomes the environment where agents live. There is no central control of the state of the graph, which is indeed spread all over the RAs.

3.1 Resource Agents

Each node in the graph is managed by one RA. RAs do not take decisions about which UA will obtain the control of the node but keep track of the node's state (free/occupied). RA also manage the allocation of arcs entering the node.

UAs interact with RAs for knowing whether the node is free or occupied in a given time slot. RAs answer the question and, in case the node is not free, tell which UA occupies the node for the given time slot, its priority, and when the node will become free again.

RAs also manage the allocation of the arcs incoming into the node they control. The RA controlling node N has the list of all its neighbors, namely those RAs controlling nodes N_{from} such that an arc $N_{from} \rightarrow N$ exists. For each arc $N_{from} \rightarrow N$, the RA also possesses the list of arcs $A_1, ..., A_k$ that conflict with $N_{from} \rightarrow N$[3].

When the RA receives a reservation request for node N and chooses the free arc $N_{from} \rightarrow N$ to reach it, it updates its reservation table by marking the arc as occupied, answers the request, and informs all the RAs that may be interested by this reservation, namely those

[3]Note that two arcs may conflict even if they have no common nodes; conflicts between arc may be represented as intersecting arcs within the graph which is, in fact, non-planar.

controlling arcs $A_1, ..., A_k$, about the new state of the arc. These RAs need to know that arcs conflicting with $N_{from} \rightarrow N$ can not be used for the specified time slot.

In this way, the neighbors of RA have up-to-date information about the state of the arcs that might cause conflicts with their own arcs, and will be able to provide conflict-free answers to successive reservation requests.

3.2 User Agents

Each UA has an original plan stated in the static allocation plan and consisting of the list of nodes to traverse together with arrival and departure time for each of them. As soon as an UA enters the graph, no matter if it is on time or late, it always tries to get a reservation for the nodes in its original path. UAs do not try to reserve a specific arc to reach a node: they ask RAs to reserve the most suitable arc for them.

UAs do not know the topology of the graph; they may interact with the Path Agent introduced later on in this section to obtain information on the paths that connect the start point where they enter the graph with the end point they must reach to exit the graph.

UAs communicate with RAs to reserve resources. Only in one case UAs may communicate with each other. Every UA has a priority that it uses to reserve a resource or even to steal a resource to another UA. In case of theft of a resource, the UA victim of the theft may directly interact with the thief as discussed later on.

Since each UA has the unique goal of getting out of the graph, it continues to look for a path in it until it obtains the reservation for all the nodes in the path. If it looses the reservation of one of these nodes, for example because a UA with higher priority stole the node to it, it will start the negotiation again until it will succeed in reserving all the nodes in one path.

3.3 Interface Agents

IAs act as an interface between the MAS and the external environment. There are three types of IAs:

- The **Path Agent (PA)** provides an interface between agents in the MAS (in particular, UAs) and the *Path Finder Service* offered by a software module external to the MAS. The Path Finder Service exploits its knowledge about the graph topology and geometry. Given two nodes, it returns a list of selected paths connecting them ordered from the best one to the worst one. The strategy for selecting and ordering the paths depends on the application. In Ansaldo's application, it depends on the number of nodes in the path and on geometrical constraints. If a UAs wants to pass through a particular node that we name "parking node", it asks the PA to look for a path satisfying this requirement. The PA uses this additional parameter to query the Path Finder Service and to obtain the list of all the paths that include the parking node.

- The **RA Manager** reads the structure of the graph from a configuration file that includes real data and creates the RAs corresponding to the nodes, equipped with all the information they need.

- The **UA Manager** creates the UAs that enter the graph according to a configuration file and taking the real data on the agents' delay into account.

4. THE NEGOTIATION ALGORITHM

4.1 Reservation of Resources

The negotiation algorithm is aimed at accommodating the conflicting goals of UAs which try to get a reservation for the resources they need to traverse the graph. RAs keep track of reservations and help UAs to find an agreement by informing them about the reservations of other UAs and the state of the graph. Depending on the situation, a UA can make two types of reservation:

- **Disputable**: this reservation can be stolen by other UAs and can be accepted by the RA only if the resource it controls is free.

- **Non Disputable**: this reservation can not be stolen by other UAs and must be accepted by the RA even if the corresponding resource is already occupied; the RA must cancel the previous reservation, unless they are "non disputable" too. The last situation, which would raise many problems in deciding which non disputable reservation should be disputed, never occurs in the real Ansaldo-STS application.

When a UA wants to reserve a resource R that labels node N for a given amount of time, it sends a request to the RA in charge of R and indicates the arrival and departure times in R, its priority, and the node N_{from} in the graph from which it wants to reach N. The RA accesses its reservation table and may answer:

- "ok" if R free in TS;

- "occupied by UA X" if R is occupied by UA X in TS; in this case, it also sends information about X's priority and about the time when R will become free again;

- "not available" if it is not able to find a free arc from N_{from} to N or if the resource itself is not available (for example because it went out of order).

If R is free in TS, then the UA gets an "ok" message from the RA and has to confirm its reservation within a specified amount of time (every request of availability has a time out).

4.2 Behavior of UAs

When a UA is created by the UA Manager, it immediately tries to reserve its original path as specified in the static allocation plan. This original path, consisting of an ordered list of nodes to reach, together with the arrival and departure time for each of them, becomes the UA's "current path". To reserve a path, the UA sends one request to each RA in charge for the resources labeling the nodes in the path. These requests are stored in the UA's knowledge base with "waiting" state and remain valid only for a specified amount of time ("time out"). The UA waits for the answers from the RAs only for this amount of time. The requests with an elapsed time out change their state from "waiting" to "not accepted".

The UA processes all the answers it receives until:

1. All the expected answers arrived before the time out expiration and they are all "ok". In this case, the UA confirms all the reservation requests.
2. All the answers arrived before the time out expiration but at least one of them was marked as "not accepted" (namely, it was either "occupied" or "not available"). The UA looks for another path, as discussed in the sequel.
3. The time out elapsed before all the expected answers arrived. The situation is the same as in the previous case: reservation requests that were not served in time are marked as "not accepted" and the UA looks for another path.

Every UA has a private knowledge on the maximum delay that it can undergo. The "occupied" answers from RAs include the time instant when the resource will be free again. When all the answers from the RAs have been received and at least one of them is not "ok", the UA computes the delay it would gain by maintaining the current path and delaying the arrival and departure times, and behaves according to the following rules:

- The computed delay is acceptable: the path is maintained, the arrival and departure times are changed using the information in the RAs' answers, and new requests are sent to the RAs with new time slots, re-starting the process of reserving a path (which is the same path as before, but with different arrival and departure times in the nodes).
- The computed delay is not acceptable: the UA asks the list of the paths from its start node to its end node to the PA.
  - If the list of paths returned by the PA is not empty, the UA selects the first path in the list, calculates the arrival and departure times to/from each node in the path[4] and sends the reservation requests for this new path, restarting the reservation process. Once obtained all the answers from the RAs, the UA decides whether to confirm the reservation or not:
    * the path is acceptable (either all the reservations were successful, or some of the reservations must be delayed because the nodes are not free in the requested time slot, but they can be reserved later without causing too much delay): the requests are changed with the new time slots (if some delay must be taken into account) and sent again; when the RA will send a confirmation of their availability, the UA will confirm their reservation;
    * the path is not acceptable: the reservations are not confirmed are will expire. The UA records the delay associated with this unacceptable path, and tries to reserve the next path returned by the Path Agent. This process can be iterated for P times, where P is a configurable threshold.
    - if, after iterating the reservation for all the first P paths (or for all the returned paths, if they are less than P), no path turns out to give an acceptable delay, the UA reserves the best one among them.
  - If the list of paths returned by the PA is empty or all the paths in the list include at least one node that is out of order, the UA reserves the current path specifying that the reservation is "non disputable". In this case the RAs involved in the reservation must answer with an "ok" message (even if the resource was already occupied; the behavior of the RA is discussed later) and when the UA confirms all the requests, it specifies that the confirmation is, again, "non disputable".

When a RA answers to a request with an "occupied" message, it also includes the priority of the UA that holds the reservation.

- If the priority of the UA sending the request ("*sender*" in the sequel) is greater than the priority of the UA currently holding the resource ("*holder*"), the sender can steal the reservation to the holder. The sender confirms the reservation to the RA and specifies that it won the implicit contest with the holder, succeeding in getting the resource.
- If the priority of the sender is equal to the one of the holder, the sender can try to steal the resource to it: in this case the UA that loses the resource is informed by the RA and can act in two ways:
  - it looks for another path and finds one with an acceptable delay: it cancels all the previous reservations and reserves the new path;
  - it is not able to find a new path with an acceptable delay: it asks to the UA who stole the resource to release it. The thief searches for another path and, if it finds one with an acceptable delay, it cancels the reservation and changes its path, otherwise it answers "no". This is the only situation where two UAs interact directly.
- Finally, if the priority of the sender is lesser than to the one of the holder, the sender cannot steal the resource and must look for another path.

If a UA succeeds in getting the reservation for all the nodes of the path it suspends until it receives a new message.

If a UA receives a "cancel" message from a RA, this means that it lost a resource (either the resource became no longer available or another UA stole it: the cause is specified in the RA's message). In the first case the UA cancels all its reservations by sending a "cancel" message to all the RAs involved in the path that it had reserved and re-starts the process of finding a path.

4.3 Behavior of RAs

When a RA is created by the RA Manager, its knowledge base is filled with information about its neighbors, about the arcs it has to manage, and about conflicting arcs. Let us suppose that RA controls resource R, where R is the label of node N in the graph. The neighbors of RA are those RAs that control resources labeling

[4]In order to calculate the arrival and departure times to/from each node, the UA assumes that il will remain X time slots in each node and that traversing an arc requires Y time slots; the UA might also know that there is one special "parking node" in the path where it must stop for Z time slots. In our application, $X = 2$ and $Y = 1$; if there is the special "parking node", Z varies from UA to UA.

nodes N_1, ..., N_m, from which one or more arcs depart towards N. For every arc connecting to a neighbor, the RA knows the list of conflicting arcs, that is, the list of RAs that must be informed when the arc is allocated. It may happen that a reservation made by a UA over a resource affects another RA because it will have no free and non-conflicting arc left. Then, every change to the local reservation table must be communicated to the neighbors in order to allow RAs to have an up-to-date and synchronized view of the state of the arcs.

RAs receive requests from UAs containing the following information:

- node from which the UA arrives,
- arrival and departure time,
- priority of the UA,
- information about the disputability of the request.

The RA answers according to the rules below:

- if the resource is not yet occupied and there is one free entering arc A, it answers "ok", updates its reservation table and informs the neighbors that arc A will be occupied in the given time slot;
- if the resource is not yet occupied but there are no free arcs for reaching the resource in the time slot specified by the UA, it looks for the first time instant when an arc will become available in its reservation table; it answers with a "not available" message including the first time instant when the resource will become free again;
- if the resource is already occupied by another UA but this allocation is disputable, it sends an "occupied" message specifying when the resource will become free again and the priority of the UA that currently holds it;
- if the resource is already occupied by another UA and the allocation is "not disputable", it answers with a "not available" message including the time instant when the resource will become free again;
- if the resource is out of order, it answers with a "not available" message including the time instant when the resource will be, hopefully, fixed. This time instant is obtained by adding a default amount of time to the current time. After the expiration of this default amount of time, the RA will check whether the resource has been fixed or not.

Every request received from the RA has a time out. If the resource is already occupied, the RA immediately sends a "not available" or "occupied" message to the UA trying to reserve it. If the resource is free and there are other requests from other UAs that are still valid, the RA waits for possible confirmations from them: after the time out, it will consider the new request from the UA and will answer "ok" to it, if it is able to find a free entering arc.

Otherwise, if during the time out period one of the "pending" UAs confirms its request, the RA immediately answers "occupied" to all the other UAs that tried to reserve the resource. In this way, if two UAs try to reserve the same resource for the same time slot, the first that asks is the one that wins.

4.4 Convergence towards a solution

According to the features of the real application that we addressed, the negotiation algorithm converges towards a solution provided that there are always enough free nodes for allowing each UA to traverse the graph even if following a path different from the original one. The number of arcs in the graph ensures a redundancy in the choice of paths (most of the nodes are reachable from at least two nodes, and at least two arcs depart from them). Also, UAs enter the graph at the time stated by their static allocation plan or later.

The static allocation plan itself ensures that UAs enter the MAS at a "reasonable" pace. This avoids that to many UAs are in the graph in the same time instant, generating a large amount of re-negotiations that might cause some UA to undergo continuous plan changes.

Although in the real application for which the negotiation algorithm has been designed it is very unlikely that two UAs reserve the same node with "non disputable" priority, the situation may nevertheless occur. This situation represents a final solution too, in the sense that the MAS realizes that it cannot cope with it and the negotiation stops. In our real application, two UAs that attempt to reserve the same node with "non disputable" priority evidence a serious problem: the number of damaged nodes and arcs in the graph is far above the physiological number of failures, and this crisis must be faced by a human expert.

5. EXAMPLES

In this section we show how our distributed allocation algorithm works by means of two examples.

5.1 Example 1: Conflicts over a Resource

The graph that we use in our first example is represented in Figure 1. There are three UAs, UA1, UA2 and UA3, with the same start and end nodes, A and H respectively.

Figure 2 shows how the resources (on the rows) are allocated to the UAs in the time (columns) according to the static allocation plan. Note that Figure 2 shows time starting from T1, which is the time slot when UA2 enters the graph, but there were other time slots before T1. For example, UA1 entered the graph at time T0 which is not shown in the figure.

Let us suppose that the current time slot is either T1 or T2, and that UA1 realizes that it must stop on D not only for the time slot T3 as the original allocation plan states, but also for T4 and T5, as it has no means to move from D until T6.

UA1 sends a "non disputable" reservation for the time slots T3, T4 and T5 to the RA controlling D, that we name D[5]. D has already a reservation from UA2 for time slots T4 and T5, but, since UA1's reservation is "non disputable", it sends a "cancel" message to UA2 specifying that UA1 stole D for time slots T4 and T5 with a "non disputable" reservation.

[5]For sake of readability, in this section we will identify the RAs with the names of the resources they control, which also label the nodes in the graph.

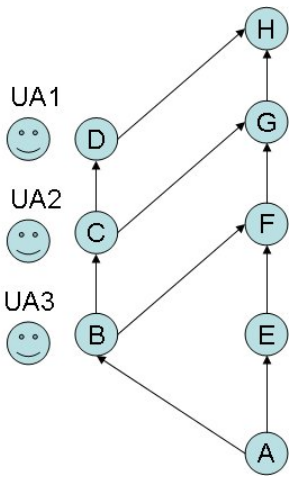

Figure 1: Example 1: the graph

	T1	T2	T3	T4	T5	T6	T7
A	UA2	UA3					
B	UA1	UA2	UA3				
C		UA1	UA2	UA3	UA3		
D			UA1	UA2	UA2	UA3	
F							
G							
H				UA1		UA2	UA3

Figure 2: Example 1: the static allocation plan

	T1	T2	T3	T4	T5	T6	T7	T8
A	UA2	UA3						
B	UA1	UA2	UA3					
C		UA1	UA2	UA2	UA2			
D			UA1	UA1	UA1	UA2	UA2	
F				UA3	UA3			
G						UA3		
H						UA1	UA3	UA2

Figure 3: Example 1: new plan under hypothesis P2 >= P3

UA2 has to change its original path for reaching H:

Case 1 if the delay that it would undergo if it stopped on C for T4 and T5 (waiting for UA1 to release D) is acceptable, it can accept to stop on C for T4 and T5 and then move to D; otherwise

Case 2 it must look for a new path.

Case 1. In the first case, UA2 sends a request to D for T6 and T7, and a request to C for T4 and T5.

Both C and D answer with an "occupied" message, specifying that the corresponding resource is occupied by UA3 that has priority P3.

UA2 has priority P2 and will behave in two different ways according to the relationships between P2 and P3:

Case 1.1 if P2 >= P3, UA2 will reserve anyway the resources, stealing them to UA3;

Case 1.2 if P2 < P3, UA2 must look for a new path.

In the first case (1.1), UA3 will search for a new path because it has lost C and D: let us suppose that it is B → F → G → H. UA3 will send the requests to the RAs in charge for the resources in this path. It will succeed in reserving them and it will exit the graph in time slot T7 (Figure 3).

In the second case (1.2), UA2 will use the path C → G → H and will succeed in exiting the graph at T8, while the reservations of UA3 will not be affected (Figure 4).

Case 2. Case 2, namely the case where UA2 should gain too much delay if stopping on C for T4 and T5, has the same solution as case 1.2: UA2 will use G and H, in the same time slots as shown in Figure 4.

5.2 Example 2: Conflicting Arcs

Figure 5 shows a graph with a conflict between the arc connecting C to D and the arc connecting F and G.

	T1	T2	T3	T4	T5	T6	T7	T8
A	UA2	UA3						
B	UA1	UA2	UA3					
C		UA1	UA2	UA3	UA3			
D			UA1	UA1	UA1	UA3		
F								
G				UA2	UA2	UA2	UA2	
H						UA1	UA3	UA2

Figure 4: Example 1: new plan under hypothesis P2 < P3

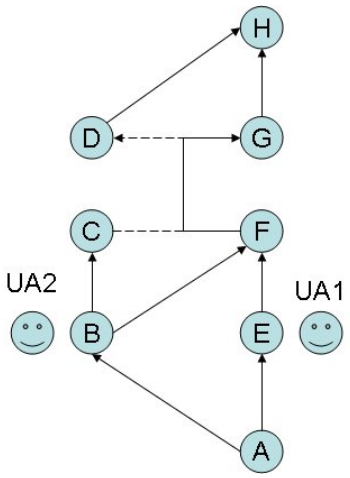

Figure 5: Example 2: the graph

	T1	T2	T3	T4	T5	T6	T7	T8
A	UA2	UA1						
B		UA2						
C			UA2					
D				UA2	UA2			
F				UA1				
G					UA1	UA1	UA1	
E			UA1					
H						UA2		UA1

Figure 6: Example 2: the static allocation plan

	T1	T2	T3	T4	T5	T6	T7	T8
A	UA2	UA1						
B		UA2	UA2					
C				UA2				
D					UA2	UA2		
F				UA1				
G					UA1	UA1	UA1	
E			UA1					
H							UA2	UA1

Figure 7: Example 2: plan with wrong reservations

	T1	T2	T3	T4	T5	T6	T7	T8	T9
A	UA2	UA1							
B		UA2	UA2						
C				UA2	UA2				
D						UA2	UA2	UA2	
F				UA1					
G					UA1	UA1	UA1		
E			UA1						
H								UA1	UA2

Figure 8: Example 2: the final reservations

In Figure 6 the static allocation plan for UA1 and UA2 is shown. They enter the graph in T1 ad T2 and need to follow two different paths. After entering the graph, both agents succeed in reserving the resources.

Let us suppose that, after the reservation has succeeded, UA2 needs to stop on B for one more time slot (T3), shifting all its reservations ahead. It tries to reserve the resources as shown in Figure 7: even if all the RAs are free in the time slots requested, UA2 receives a "not available" answer from D.

This is because, at the beginning of the negotiation algorithm, G informed D that it had a reservation for time slots T5, T6 and T7 issued by UA1, and that it would use the arc F $\rightarrow$ G to reach G. This arc has a conflict with C $\rightarrow$ D, so G informed D about this. Then, D answers to UA2 that D is free starting from T6 (one time slot is the time required to traverse an arc).

UA2 tries to reserve a new path stopping on D in the time slots T6 and T7 and exiting in T8. H answers with an "occupied" message and, if UA2 has less priority than UA1, it will finally reserve the path as shown in Figure 8. Otherwise, UA2 will stole H to UA1 in T8, and UA1 will wait on G until T9, when it will get a reservation for H and will exit the graph.

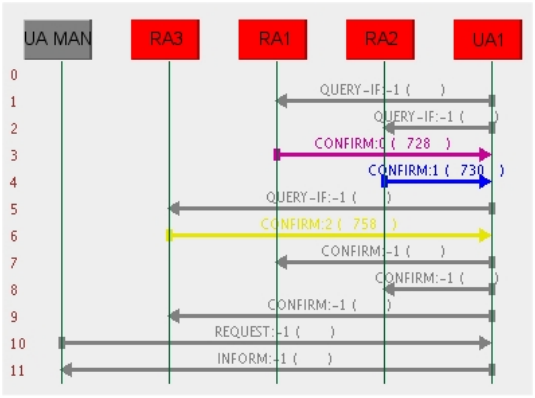

Figure 9: Screenshot of the JADE Sniffer Agent

5.3 Running the MAS in JADE

The MAS has been implemented with JADE and uses JADE WSIG for interfacing with applications outside the MAS. Due to the confidentiality of the project we can not go into the details of the implementation. We limit ourselves to showing a screenshot of the JADE Sniffer Agent (Figure 9) taken during one of our tests run on a Notebook PC with Intel Core 2 Duo Processors, 2.4 GHz, 4MB L2 cache Processor, 2048MB SDRAM, Linux SUSE OS.

In this screenshot we see UA1 that enters the graph in node 1 managed by RA1 and wants to move to node 2 managed by RA2 and to exit in node 3 managed by RA3.

As soon as the the UA Manager creates UA1, UA1 sends a "query-if" to the three RAs in order to reserve the resources it needs.

After receiving all the three messages from the RAs with "confirm" performative (namely the messages that we named "ok" in the paper), UA1 replies with three confirmations in order to reserve the path.

Finally the UA Manager sends an information request about the reserved path to UA1, and UA1 informs it of the path that it just reserved.

We have run more complex tests with 50 RAs, 3 to 4 UAs, an average of 50 incompatibilities for each arc caused by 10-12 nodes. For any update of an arc's state, the RA in charge for that arc sends messages to 10 RAs and each message contains 10 to 15 incompatible itineraries. Although very complex, the message exchange for updating the state of arcs requires few milliseconds.

6. CONCLUSIONS AND FUTURE WORK

The paper outlines the engineering of a MAS designed and developed by an industrial and an academic partner, Ansaldo-STS and DISI. The implementation and testing are close to completion and the MAS is almost ready to be integrated into the Ansaldo-STS application for which it was developed.

The relevance of the paper mainly lies in the story it tells: industry trusts the agent technology and is willing to integrate it into its core business applications. University looks for significant case studies and is willing to demonstrate that agent technology deserves the trust from industry. When these complementary wills meet, the results may lead to innovative industrial products and to constructive academic experiences.

The main direction of our future work is to provide a systematic comparison of our distributed negotiation algorithm with the existing literature. We started this activity and we realized that a vast amount of proposals similar to ours exist. However, each of them had its own peculiarity that made it suitable for a very specific application and not suitable for ours. This initial experience convinced us that a serious comparison with the related work, aimed at understanding whether our algorithm is more general, efficient, flexible than others, may require months. Since we had tight time constraints due to Ansaldo-STS's commitments, we opted for postponing the comparison activity after the end of the MAS implementation and testing.

7. REFERENCES

[1] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[2] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica*, 30:3–31, 2006.

[3] JADE Board. Jade web services integration gateway (wsig) guide. `http://jade.cselt.it/doc/tutorials/WSIG_Guide.pdf`, 2008.

[4] V. Mascardi, D. Briola, M. Martelli, G. Arecco, R. Caccia, and C. Milani. A prolog-based mas for railway signalling monitoring: Implementation and experiments. In M. Baldoni, M. Cossentino, F. D. Paoli, and V. Seidita, editors, *Workshop Dagli Oggetti agli Agenti, WOA'08, Proceedings*. Seneca Edizioni, 2008.

[5] Multi-Agent Systems Lab, Department of Computer Science at the University of Massachusetts at Amherst. Publications on negotiation in multi-agent systems. `http://mas.cs.umass.edu/pub/search.html?search=true&author=&title=&year=&keywords%5B%5D=Negotiation`, 2009.

[6] Stanford AI Lab, Stanford Computer Science Department. Publications on game theory and multi-agent systems. `http://robotics.stanford.edu/~shoham/YoavPublications.htm`, 2009.