Multi Agent Resource Allocation: a Comparison of Five Negotiation Protocols

Daniela Briola and Viviana Mascardi
Dipartimento di Informatica e Scienze dell'Informazione (DISI)
Università degli Studi di Genova
Email: {briola, mascardi}@disi.unige.it

***Abstract*—This paper describes five systems that exploit negotiation strategies to solve multiagent resource allocation problems. A deep comparison is drawn among them according to different criteria that involve general features of the systems; adherence to widely accepted agent definitions; domain, purpose, and approach; analysis, design and implementation of the negotiation protocol. Considerations on how extending one of the analyzed systems in order to move a concrete step towards the realization of an *integrated platform for developing negotiation protocols* are also provided in the conclusions.**

I. Introduction

Allocating resources amongst multiple entities is a central matter of concern in both Computer Science and Economics. It has inter-disciplinary characteristics which make it relevant to disparate application domains including industrial procurement, manufacturing and scheduling, network routing, airport traffic management, crisis management, logistics, public transport, and the timely allocation of resources in grid architectures [14].

The abstractions of agents and multiagent systems are very suitable to model and implement distributed problems of this kind, and the Multiagent Resource Allocation (MARA) research sub-field is recently gaining more and more attention.

A tentative definition of MARA can be found in [14]:

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

However, as the authors of [14] observe, this definition needs to be further qualified: What kind of items (resources) are being distributed? How are they being distributed (in other words, what kind of allocation procedure or mechanism do they employ)? And finally, why are they being distributed (that is, what are the objectives of searching for an allocation and how are these objectives determined)?

In this paper, we analyze five existing systems that solve different MARA problems, and whose allocation procedure mechanism is based on *negotiation*.

For each of them we answer the questions of what kind of items (resources) are being distributed and why are they being distributed, we provide some details on the negotiation protocol they adopt, and we answer many other questions that, in our opinion, are relevant to gain a deeper insight of the system and to understand whether and how it might be exploited to solve the specific user's MARA problem.

Due to this large amount of MARA problems and related solutions described in literature, we limit ourselves to analyze those works that are closer to the FYPA one we modeled and implemented in our recent research activities [7], [12].

In particular, we focus on works where agents interact to coordinate themselves for moving in a well defined physical area or for accessing shared resources during time. For each system we discuss, we motivate why we did not select it to solve our specific FYPA MARA problem.

The paper is structured in the following way. After discussing the state of the art in agent-based negotiation (Section II), in sections III-VII we introduce the five systems and the MARA problems they solve:

- FYPA (Find Your Path, Agent!) [7], [11], [12], developed by ourselves for solving a real industrial problem of dynamic (re-)allocation of tracks to trains inside a station (Section III);
- MPCA (Multi-Party Collision Avoidance) [40] for airplane collision avoidance (Section IV);
- APR (Airplane rerouting) [3] for airplane collision avoidance (Section V);
- Waypoints [34] for autonomous robots collision avoidance (Section VI); and
- SPAM (Scalable Protocol for Anytime Multi-level) [27] for target tracking with sensors (Section VII).

In Section VIII we provide a systematic comparison among these systems, carried out taking different criteria into account ranging from how much do they stick to the agent definition given in [24], to their industrial exploitation, to the language and platforms used for their implementation. Section IX concludes the paper and outlines the future directions of our research.

II. Agent-based negotiation: the state-of-the-art

A. Trends and research areas

Multiagent research has long been divided into two fields, one concerned with cooperative (benevolent) agents and the other concerned with self-interested agents [18]. There has been very little cross-fertilization of ideas between these fields. Research on self-interested agents is often based on classical game theory with its assumptions of common knowledge among agents and complete rationality of agent reasoning. This is in contrast with the research on cooperative agents

which makes no such assumptions; rather, it has generally been based on heuristic approaches having their roots in knowledge-based AI search, planning and scheduling mechanisms.

However, perhaps the most fundamental and powerful mechanism for managing inter-agent dependencies at run-time is negotiation, that is the process by which a group of agents come to a mutually acceptable agreement on some matter. Negotiation underpins attempts to cooperate and coordinate (both between artificial and human agents) and is required both when the agents are self interested and when they are cooperative.

In the remainder of this section we will concentrate on cooperative agents and on the main areas where they are used. Most of our knowledge on this subject comes from the overview on negotiation in MASs made by Lesser [26], whose research activity, carried out in collaboration with many other scientists, covers a large variety of case studies, domains and applicative projects.

Examples of application domains that have used a multiagent approach for allowing the involved entities to negotiate in a cooperative way are:

- Distributed situation assessment, which emphasizes how (diagnostic) agents with different spheres of awareness and control (network segments) should share their local interpretations to arrive at consistent and comprehensive explanations and responses. Examples of applications belonging to this category are:
  - network diagnosis [37];
  - information gathering on the Internet [17], [32];
  - distributed sensor networks [13], [29]).
- Distributed resource scheduling and planning, which emphasizes how (scheduling) agents (associated with each work cell) should coordinate their schedules to avoid and resolve conflicts over resources, and to maximize system output. Under this category we can find for example:
  - factory scheduling [31], [33], [38];
  - network management [1];
  - intelligent environments [6], [22].
- Distributed expert systems, which emphasize how agents share information and negotiate over collective solutions (designs) given their different expertise and solution criteria. The following applications fall under this category:
  - concurrent engineering [25];
  - network service restoration [15], [23].

The need for a multiagent approach can also come from applications where agents represent the interests of different organizational entities (e.g., electronic commerce [39] and enterprise integration [4]). Other emerging uses of multiagent systems are in layered systems architectures where agents at different layers need to coordinate their decisions (e.g., to achieve appropriate configurations of resources and computational processing [41]), and in the design of survivable systems where agents dynamically reorganize to respond to changes in resources availability, software and hardware malfunctions, and intrusions.

In general, multiagent systems provide a framework where both the inherent distribution of processing and information in an application and the complexities that come from issues of scale can be handled in a natural way.

In these areas the word "Negotiation" is rarely expressly and explicitly used, but nevertheless in these fields agents usually need to cooperate to achieve a goal, and cooperation is a type of negotiation.

There has also been a long tradition of work dating back to the inception of the field on coordination based on logical reasoning about the beliefs, desires, intentions (BDI) and commitments of agents [16], [20], [35], [36], and more recent work on the use of market mechanisms for solving multiagent resource allocation problems [42].

The synthesis of ideas from each of these different approaches to coordination holds great potential for future developments in the field.

III. FYPA

The FYPA problem was proposed to us by Ansaldo STS, the Italian leader in design and construction of signaling and automation systems for conventional and high speed railway lines.

Ansaldo STS uses a system that computes, few times a year, the global distribution of trains over the Italian railway network. That system does not compute safe paths of trains inside stations: the system is only able to generate paths for crossing the station that are consistent with the station physical configuration, but cannot check that each path does not raise conflicts with paths of all the other trains crossing the station in the same time period. This check and the final identification of safe paths inside the station were indeed performed by human operators by hand.

In order to automatize the train schedule definition process as much as possible, Ansaldo needed another system that could be used off-line to identify a safe and (sub)optimal allocation of tracks inside a station to trains, given the time trains enter and exit the stations (times that are decided by the pre-existing system).

One of Ansaldo's requirements was to have a system designed and implemented is such a way that, if its performances demonstrated to suit the real time constraints, it could be used on-line as well, in order to face re-allocation problems due to unavailability of the pre-computed path because of tracks being out of order or already occupied by other trains.

The FYPA system that we developed is suitable both for on-line and off-line usage. It is already used for off-line computation of paths inside stations, and its on-line application is under way: FYPA is able to compute a complete re-allocation of trains in huge stations (tests made by Ansaldo STS engineers with data from Mestre and Pisa) in less than 2 seconds, hence quasi real-time [10].

In order to solve Ansaldo STS' problems and implement the FYPA system, we designed a multiagent system able to manage the real-time allocation (and re-allocation) of a set of limited resources. The resources are railway tracks (or only

"railway" further on) inside a station plus a set of segments of railway where trains are allowed to stop (for example to let the passengers in and out) that we call "stop nodes": these nodes are connected by one, or usually more, railways. A "stop node" can be occupied by only one train at a time. Due to the physical distribution in the space of "stop nodes", some of the railways intersect with each other so it is not possible for different trains to use them at the same time, because they may crash.

A train can enter or leave the station using only a set of "stop nodes", that we call "entering" and "exiting nodes" respectively. Once inside the station, the train can move following the railways to reach some "stop nodes", can wait there for a period it autonomously determines and then can reach an "exiting nodes" and leave the station. FYPA only manages trains inside the station or entering it. When they reach an "exiting nodes", they leave the station and are no longer managed by FYPA.

Every train has a predefined path in the station that it should follow, that is generated by the pre-existing Ansaldo system but that does not meet the requirements of being conflict-free with respect to the other paths. Each train can hence change some "stop nodes" if needed to reach the conflict-freeness requirement, but it usually has to deviate as little as possible from its original plan.

The main aim of our protocol is to dynamically find a solution to conflicts and unavailabilities (that may be dealt with in an homogeneous way, since both amount to a resource needed by a train not being available), or in other words to find a new path in the station for every train, respecting all the safety, minimum-delay and minimal-changes constrains.

When an allocation problem arises, the system has to change the "station crossing plan" of one, or more, trains: every train has a specific typology (may be a slow passenger train, a fast one, a goods carrier and so on) and every typology has a priority. So if the system has to make a train stop for more time on a "stop node", waiting for the next node to become free (forcing in this way the train to wait), usually it prefers to stop the train with lower priority. The aim of the algorithm is also to find a solution where the delay of the involved trains is as limited as possible, and where the higher priority of a train implies a minor delay. Moreover the protocol has to keep the changes made to the original "station crossing plan" as limited as possible.

IV. MPCA

An area where multiagent systems are very often used is the one of "collision avoidance": in this domain the system reflects a situation where unmanned entities (autonomous airplanes) need to move avoiding crashing, respecting constraints on the path they can use, time to make a decision, distance they need to maintain between them and so on.

Usually, this type of problem is faced in the airspace management, where unmanned little planes need a protocol to negotiate the route, or in software applications where the multiagent system must suggest the user possible alternative reorganizations of the route of many planes.

Regarding the airspace management, an accurate research has been made by the "Agent technology center" (http://agents.felk.cvut.cz/) that developed AGENTFLY, "*a multi-agent system enabling large-scale simulation of civilian and unmanned air traffic. The system integrates advanced flight path planning, decentralized collision avoidance with highly detailed models of the airplanes and the environment*" [2]. The AGENTFLY project is still maintained and its authors are continuously improving it in many ways. Its license was sold to BAE Systems as a testbed simulation platform. It is used by US Air Force, IHMS at Florida and several universities. Its authors are working with the Federal Aviation Authority (FAA) using AGENTFLY as a tool on their computation grids.

Besides this system, the authors presented many studies on protocols to avoid collisions in airspace.

In [40] two algorithms to avoid collisions among aerial vehicles are presented. The authors kindly accepted to review our analysis (in this section and in section VIII) of their work, so we are able to provide some details also on the AGENTFLY project in Section VIII, where we add some footnotes to Tables to report additional information submitted by the authors.

In a three dimensions space a group of autonomous airplanes with a mission need to coordinate themselves to avoid collisions: a mission is made of several points that must be reached in a specified time interval. To fulfill the mission, the airplane will follow a list of steps, each characterized by a maneuver, a direction, a velocity, starting from the previous step. More, every plane must maintain a minimum distance from all the others, and there are no-flight zones in the airspace. When two, or more, airplanes have a part of the plan in common, they need to change it to avoid crashing.

Every agent can only interact with the ones within a range R defined at the start of the simulation: these are the other planes that it can "see" (on a virtual radar) from its position in the space. Every agent sends to the others it sees an update on its future mission steps so they can check if there are conflicts.

The authors propose two solutions to the above problem: a local one and a global one.

In the local one, the two agents involved in a conflict propose, at first, a list of possible changes to their path and look if they can adopt one of these change to avoid the conflict. If they are not able to find a solution based on the list of generated maneuvers, they generate more maneuvers (accepting higher values for the parameters, for examples trying to move more to left, speeding up/slowing down more, making the altitude higher and so on) and add them to the list: then they try again to solve the conflict. They repeat this procedure until they find a solution. In this way the list can be filled with tens or hundreds maneuvers and the solution is always found. A random choice is used when there are several solutions with the same utility value (using an utility function not described here).

When the agents have solved their conflict, they will check

if other collisions exist and will start the protocol again to solve the first one that arises.

If an agent is involved in more than one collision, it will solve the one that is expected to occur first in time.

This algorithm is called "Iterative peer-to-peer collision avoidance (IPPCA)".

In the second algorithm (the one proposing a global solution) called Multi-Party Collision Avoidance (MPCA), authors enlarge the set of agents involved in the collision to find a better solution: in this case the idea is to give the colliding agents enough space to make their evasion maneuvers (to avoid all the agents around) without changing their plans too much. The agents that have a colliding path create a group: then they try to change their paths (they create a list with all the possible modifications to their paths) and add to the group those agents that could be interested by these path changes. So the group is enlarged to involve all the agents that are near to the ones that have a conflict. Then, the possible alternative plans are analyzed by the group till a solution is found, and all the agents will modify their path as decided.

The group searches the states space of possible plan changes (due to all the possible applications of evasion maneuvers sequences, velocity change and so on) using an A* algorithm modified for the situation.

To simplify the communication and synchronization issues the authors have implemented this protocol creating, for each group, a coordinator agent that collects all the information it needs to solve the problem and than to find a solution. In this way the organization of the algorithm is partially centralized.

If an agent is involved in more multi-party groups, it will only join the one with the earliest expected collision in the time line.

As the authors explained to us, MPCA could be used as a global protocol, but it is not intended to be so. The idea is to used it as a "local/global" protocol. This means to compute "globally" a solution for a small group of airplanes, that are involved in a single collision situation, but in the meanwhile another group of agents can solve different collision in their corner of the world. So MPCA was designed to be something between local and global: it is used to solve "globally" ad hoc local problems.

The MPCA has been tested using AGENTFLY and the comparison has been made using the IPPCA algorithm, implemented as plug-ins in AGENTFLY.

The observed differences between IPPCA and MPCA are:

- message flow characteristics: IPPCA has quite steady bandwidth of communication flow. MPCA has high peaks and then very low communication (as the data are computed by the coordinator);
- quality of solution: MPCA provides better solutions because the A* is able to find solutions that are not checked with IPPCA;
- computation demands: as the MPCA goes through much bigger space, the computation is much longer. This one of the reasons to keep MPCA local by restricted size of the group.

The MPCA algorithm in particular is very similar to the FYPA one because it is based on the same idea of "moving others to get space for you": for this reason it has been selected for our analysis and comparison. The MPCA domain, instead, is quite different from ours because airplanes have much more flexibility in their movements than trains (they must avoid certain zones and avoid other agents, but must neither follow rigid and limited paths, nor have to reach a fixed point but and area) and, above all, they move in a 3D space. For these reasons MPCA could not be selected for solving our FYPA problem.

V. Airplane rerouting (APR)

Another attempt to apply multiagent system negotiation to the airplane collision avoidance problem comes from Agogino and Tumer [3]. In their work the authors present a multiagent structure to control air traffic flow using tree kinds of "change mechanisms", explained later, and then analyze a learning algorithm to improve the system efficiency (that we do not report here because it is out of our scope). The domain is the one of US airspace, where the space is divided into regional centers and again into sectors. The algorithm uses a global evaluation function that considers the congestion in a particular set of sectors and the global air traffic delay. Using this common function agents independently take decisions about how changing their plans. In this system agents are ground location throughout the airspace and are called "fixes". Each agent is responsible for the aircrafts going through its fix. Every airplane has a "flight plan" consisting of a sequence of fixes. In this organization agents can change the plan of the interested airplanes in three ways:

- Miles in trail (MIT): agents control the distance that the airplane must keep from each other while approaching a fix. If the MIT values is high, fewer planes will be able to cross this area because they need to slow down their velocity to maintain the distance
- Ground delays: an agent can control how long aircrafts that will eventually go through a fix should wait on the ground, that is, the airplanes will arrive later at the fix
- Rerouting: an agent can divert the foreseen planes of its fix making them choosing another path.

The algorithm identifies sets of agents that can influence themselves rerouting airplanes in their fixes: each agent lists the possible solutions to the congestion problem and then chooses the best solution using different learning algorithms (the authors study some types of learning strategies and compare them).

This agent's organization is quite similar to the one we adopted, so we included this algorithm in our comparison.

From the available documentation, it is not clear whether the agents can use in the same simulation all the three techniques seen above to change the plan (or if they must use only one type in a run) and furthermore we did not find the strategy used to choose among these strategies (if they are foreseen simultaneously).

The algorithm has been developed using FACET (Future ATM Concepts Evaluation Tool) [5], that receives scripts from the agents, simulates the execution of the algorithm, and returns the overall impact of their changes to the airplane's plans.

This algorithm is interesting but seems to work only if the groups are limited to few agents, because it is based on the list of possible choices, that becomes too long if there are large sets of agents, or too many possibilities of plan changes. More, the agents seem not to negotiate, but only to work independently using the same evaluation strategy and only one possible change to the airplane route. All these constraints seem prevent the system from being applicable to very complex scenarios or short term simulations, as FYPA instead does.

VI. Waypoints

Similar problems and solutions as those discussed above can be found in the field of autonomous robots which are able to independently move and need to avoid collisions while trying to reach a desired destination. This is a common situation for example in the military area, where robots are entities that must explore territories, or in industrial applications where little robots could be used in places that are not suitable for humans (under-water, in mines, and so on).

In the area of autonomous robots it is worth mentioning the proposal made by Purwin, D'Andrea and Lee [34]: the authors present a cooperative decentralized path-planning algorithm for a group of autonomous agents that provides guaranteed collision free trajectories in real-time. The algorithm is based on the idea that every agent reserves an exclusive area (called A area) for itself and always remains inside that area, and no two reserved areas are allowed to intersect at any time.

The core of the algorithm is based on pair-wise conflict resolution among two agents. Both agents operate according to exactly the same rules, with the only exception being how priority is assigned. For the algorithm the following assumptions are made:

- decentralized agents: all computation/control is done on board;
- the total number of agents is known;
- motion primitives are available to move the agents in a deterministic fashion;
- point-to-point communication between agents is supported;
- agents can localize themselves, but not others.

The algorithm is executed considering discrete time (agents work with "frames", that is, they assume a discrete divisions of time).

Every agent is trying to reach its task location, which has been selected by some higher level entity that is not in the scope of that work. Position and velocity of the agent are expressed in a global Cartesian coordinate system. The agent's motion is controlled by a deterministic motion primitive MP(), which contains trajectory generation and low-level control of the mechanical actuators (also this aspect is out of the scope of that research). Upon specification of a desired destination D the motion primitive will compute a path that takes the agent to D with zero final velocity.

Every agent stores information about itself and the other agents and is able to communicate using a wireless network.

The algorithm grants that every agent will always, and only, move inside its reserved area, A. Instead of choosing the reserved area (A) directly, agents are indicating their intentions by requesting an area (called B area) first, and exchanging it with the others. Agents can change B arbitrarily. However, significant changes to B can cause the negotiation cycle to start over. The requested area B has to contain the reserved area A at all times, hence, an agent cannot move to a location that is not inside the requested area.

Two intersecting B areas indicate a possible conflict. In this case the agents will negotiate to find out which one gets priority and how the A areas are being selected. The base for this negotiation is a scalar cost function: the agents with a conflict exchange their respective costs and choose who has the higher priority using this, choosing also who will change its A area. The agent with the lower priority will reduce its A area (stopping and waiting for the other agent to pass over).

With this solution an agent should stop and wait several times, getting a huge delay: the algorithm proposes also a second way of acting. Instead of stopping, an agent can decide to change its path to avoid an obstacle (or simply a point of its paths that intersects with many other agents) and to move around that. It will choose a "way-point", that is a new intermediate destination, and will try to reach it using the same algorithm shown above. Then it will start again to reach the initial destination.

In the basic algorithm all agents can communicate with all the others: this implementation is foreseen for autonomous robots, with limited battery autonomy and wireless communication limitation, so if the total number of agents is huge then the robots can limit their communication range to reach only the nearest agents, saving energy. This algorithm's limitation is realistic because probably only the agents that are close to each other can have intersecting paths, consequently an agent must inform only its neighbors of the changes to A and B areas.

We do not describe the algorithm with more details because it presents a complex and out-of-our-scope description of the primitives chosen for changing the reserved and requested area, that are managed by a geometrical function and not by the negotiation protocol: the A and B areas will be always rectangles, chosen to approximate the new path. Moreover, the paper presents the pseudo-code of the protocol, so reporting it here is not interesting, and the code in C++ is also available on the author's web site.

This algorithm is similar to FYPA in the idea of how the agents collaborate to solve the conflicts and how they can change their strategy: in both protocols agents can stop and wait for the other to move, or can change their path. The difference is that in our domain the paths are limited to a predefined set and are divided into fixed parts. Every subpart

is managed by a Resource agent, whereas in that article agents are able to move without limitations and without intermediate agents. More, those agents operate in a wireless environment, so the number of exchanged messages and the real distance of the agents can make the difference on the behavior of the entities: in this case some limitations and heuristics must be adopted (not described here), while in our algorithm the only limitation is due to the computational time of managing all the messages. Anyway we tested our system also with complex configurations and the number of messages does not make the performances degrade.

VII. SPAM

In [27] a "*cooperative negotiation protocol that solves a distributed resource allocation problem while conforming to soft real-time constraints in a dynamic environment*" called SPAM (Scalable Protocol for Anytime Multi-level) is presented by Mailler, Lesser and Horling. In particular that proposal models the resource allocation problem as a constraint satisfaction problem.

As in the other domains described above, in that work we can find a set of limited resources and some agents interested in using them in different moments for different periods: the resources are three sensors platforms, and agents need at least three sensors to track an entity moving in the environment. More sensors give a more accurate target's location.

In the proposed solution each platform is managed by an agent, which is also in charge of localizing and following a target (this task allocation is made out of the system), deciding which sensors (of which platform) it needs to do this and when. So if two, or more, agents (called also track managers) need the same sensor for different tasks then a conflict arises and it must be solved. The solution is centralized because the agent that first creates the conflict becomes the "mediator" of it and must solve it, asking the other involved agents information and then propagating its decision. Every agent has an utility function U that uses to evaluate the proposed solution (the set of sensors and the period of usage), and can change U to solve quickly a conflict.

SPAM protocol works in two main phases, trying at first to find a solution which avoids the negotiation. At the end of the first stage there is always a solution, even if it is not the optimal one or if it generates conflicts. The algorithm foresees also to lose a target, solution that anyway causes a huge penalization of the social utility.

Stage 1 of SPAM serves two primary functions. The first one is to try to find a solution within the context of the information that the protocol has when it starts up. However, since the protocol attempts to maximize the social utility, each of the agents tries to maximize its local utility without causing new constraint violations. If this can be done, then no further negotiation is necessary, and the protocol terminates at the end of stage 1.

Moreover, in this stage the agents use a "concession rate" to decide if they have to activate the second phase, more expensive, or not: this rate is a percentage of the agent's utility and specifies how much the agent will concede before skipping to the second phase.

The second function of stage 1 is to ensure that some utility is obtained while waiting for stage 2 to complete. If the reason the protocol was started was a resource requirement change, a temporary solution is applied to the problem: this solution, although not conflicts free, has the ability to obtain at least some utility while the mediator tries to get a better solution. Conflicts that are unresolved are actually left to the individual sensor agents to handle.

If stage 1 was activated because of a newly discovered conflict, and a conflict-free solution cannot be found, then the manager just enters stage 2: in this case it does not concede, does not bind a temporary solution, and it does not reset its objective level, but it only enters stage 2 to find a solution.

Stage 2 attempts to solve all local conflicts that a track manager has by elevating the negotiation to the track managers that are in direct conflict over the desired resources. The originating track manager takes the role of the negotiation mediator and starts collecting all the information it needs to generate alternative solutions. These solutions are generated without a global vision, so they are conflict free only from the point of view of the mediator. What this means is that the view of the mediating manager is limited to only the constraints that arise from the sharing of a resource with it. If the solution that will be imposed by the mediator will cause other conflicts then other agents will try to solve them, even starting again all the algorithm.

When an agent is the mediator of a conflict, it starts asking the other agents for meta level information, than it elaborates them and generates all the possible alternative solutions, including those where one, or more agents, must lower its utility function. Then it will send the list with the alternative solutions to the agents involved and will wait for their responses: the other agents will reorder the list using their local information and utility function and will send back this reordered list to the mediator. Last, the mediator chooses a solution, possibly a good one for all the agents involved, and sends it to the agents that must apply it.

At this point, each of the track managers is free to propagate and mediate a new negotiation if it chooses to enter the second stage. At the time when this article was written, SPAM allows the agents to enter potential oscillations, maintaining no prior state other than objective levels, from negotiation to negotiation and rely on the environment to break oscillations.

To test the SPAM protocol the authors implemented a model of the domain in a simulation environment called Farm [21], a component-based distributed simulation environment written in Java. They performed tests to evaluate the performance of SPAM compared with those of a Greedy Tracking Agent and an Optimal Tracking Agent. To do this, the simulation used not moving targets.

In the simulation with moving targets (a not comparing test, only a performance test), SPAM gives good results, near to the optimal, and shows a linear increase in the time needed to converge when the problem gets harder, but they do not have

implemented other solutions to compare with.

Comparing this protocol to FYPA, the main difference that emerges seems to be in the possibility for a track manager agent to "loose a track", namely give up tracking an object, if it becomes too hard to do that: in SPAM this event is allowed, even if it is the last option, whereas in our protocol it is not possible for a user agent - a train - to give up obtaining resources - railway tracks and nodes: the train has to stay somewhere in the station!. Furthermore, the representation of the domain as a constraint problem should be almost difficult for the FYPA domain, and finally we preferred a real distributed negotiation, while the solution proposed in [27] is partially centralized. In a way similar to the Contract Net protocol, agents do not negotiate, they choose one of them to solve the constraint problem and then apply the solution it proposes.

VIII. Comparison

A brief comparison among FYPA and these algorithms has been already presented in the previous sections, with the aim of clarifying why they have been chosen for the comparison and why they have been not adopted to solve our resource allocation problem.

Now, in order to draw a systematic comparison between the five systems introduced in the previous sections, we identify a set of features relevant for characterizing a negotiation protocol, besides the general ones proposed in [14] and summarized in Table I.

A. Accepted agent definition

We adhere to the definition given by Jennings, Sycara and Wooldridge [24]:

> "*An agent is a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives. There are thus three key concepts in our definition: situatedness, autonomy, and flexibility.*"

Hence, the first characterizing feature we consider in our comparison is what definition of agent is accepted by the authors, specifying which standard features can be found in the MAS discussed above.

- **Are agents autonomous?**
- **Are agents situated?**
- **Are agents responsive?**
- **Are agents pro-active?**
- **Are agents social?**

Table II provides a comparison among the protocols we described in this paper, with respect to the accepted definition of agenthood. For every parameter the possible values are:

- YES: the agents show all the typical features associated with the parameter;
- NO: the agents show none, or very few, of the typical features associated with the parameter;
- LIMITED: the agents show some of the typical main features associated with the parameter.

In particular, "Limited" referred to sociality underlies a solution where the agents communicate among each others but use very simple data structures and, above all, sociality amounts to simple exchange of information: the solution to the problem is partially centralized, so the agents are more "communicative" than "social". The same value for the "Pro-activity"[1] parameter underlines that the main behavior of agents is passive, that is, they wait for some changes in the environment happen before acting. So these agents are more reactive than proactive.

B. Domain, Purpose, Approach of the MAS

Table III provides a comparison with respect to the general features of the protocols.

- **Which is the domain where the protocol is applied?** The described protocols refer to different applicative domains, that are summarized in this way: Railway management ("RAIL"), Air traffic management ("AIR"), Sensors management ("SENSOR") and Autonomous robots's path management ("ROBOT").
- **Is the protocol used for simulation purposes (for example, for performing a what-if analysis or for implementing a decision support system)?** For this parameter the value will be "YES" if the system/protocol has been implemented/used in simulations too
- **Is the protocol used for controlling agents (hence, not for simulation purposes, but for allowing real agents to negotiate)?** In this case, the value will be "YES" if, and only if, the system has the real control over physical entities, that is, its decisions are not checked by an human operator. The value will be "Limited" if a human user will accept the proposed solutions before applying them.
- **Which is the approach underlying the protocol?** For this parameter we list the main standard techniques used in the protocol, for example Game theory, Auctions and so on.
- **Is the system used for a real industrial application? Which one?**

C. Analysis and design of the MAS negotiation protocol

Table IV shows the values for the parameters that we considered for the design features of the protocols.

- **Which is the detail level of the MAS design?** The possible values are: High (Verbal description of the system), Low (Detailed description with use cases or Class Diagram or other languages)
- **Is the pseudo-code, or a simplified version of the code, of the negotiation protocol available?**

[1]Concerning the MPCA protocol, the value NO for the filed "Pro-activity" is correct if we limit our evaluation only to the MPCA protocol, but if we consider agents in the whole AGENTFLY project (those called "pilot agent"), they are pro-active as well. At the beginning, every pilot agent receives its mission (which can be changed by human operator during the simulation) and the agent plans its trajectory to fulfill its mission or to communicate with other agents if it is in a group mission (see "tactical-agentfly" at http://agents.felk.cvut.cz/)

	What kind of resources are being distributed?	How are they being distributed?	Why are they being distributed?
MPCA	volumes of air space	see details in Section IV	to avoid collisions among airplanes
APR	volumes of air space	see details in Section V	to avoid collisions among airplanes
Waypoint	rectangular areas	see details in Section VI	to avoid collisions among robots
SPAM	sensors	see details in Section VII	to track targets
FYPA	railway tracks and nodes inside a station	see details in Section III	to avoid collisions among trains

Table I
COMPARISON: GENERAL FEATURES

	Autonomy	Situatedness	Reactivity	Pro-activity	Sociality
MPCA	YES	YES	YES	NO	YES
APR	YES	YES	YES	LIMITED	LIMITED
Waypoint	YES	YES	YES	NO	LIMITED
SPAM	YES	YES	YES	NO	LIMITED
FYPA	LIMITED	YES	YES	YES	YES

Table II
COMPARISON: ACCEPTED AGENT DEFINITION

	Domain	Simulation	Real control	Approach	Industrial
MPCA	AIR	YES	YES	A* state search	YES (BAE Systems and others)
APR	AIR	YES	LIMITED	Learning algorithm	LIMITED (foreseen for the Federal Flight Administration)
Waypoint	ROBOT	NO	YES	Scalar cost function, Computational Geometry	NO
SPAM	SENSOR	YES	NO	Distributed Constraints Satisfaction Problem	NO
FYPA	RAIL	YES	LIMITED	Distributed resources allocation	YES (Ansaldo STS)

Table III
COMPARISON: DOMAIN AND SIMULATION

- **How many different agent roles does the MAS include?**
- **How many different kinds of messages do the agents exchange?** The number indicates the type of different ACL messages (if known) or different semantics (request, answer, update...)
- **Is the solution computed in a partially centralized way (at least, in any iteration step of the negotiation)?** The value will be YES if the final solution in calculated by only one agent[2], NO if the solution emerges from a real negotiation among agents.
- **Is the algorithm guaranteed to terminate? Under which conditions?** The value will be NO if the algorithm is allowed to enter a loop or a situation where it does not assure to find a solution within a specified time, while the value will be YES if the algorithm will always find a solution. In this case, if the solution is partial or it accepts some constraints violation, this will be specified between round brackets.

If, from the available documentation, we were not able to clearly understand the correct value to be assigned to some parameter, we use the value UNK, that stands for "unknown". This applies to Table V too.

[2]Concerning the MPCA protocol, a decentralized implementation exists but it is not published, so we do not consider it in this paper.

D. Implementation of the MAS negotiation protocol

In Table V we summarize the values for the parameters regarding the implementation features of the protocols.

- **Is the MAS implemented?**
- **In which programming language?**
- **Is the MAS based upon an existing agent platform?**
- **Can the strategy of the agent vary during the same execution run?** With "varying strategy agent"[3] we intend an agent that is able to act in different ways using different rules. That is, it is able to find many ways to solve a problem during the execution/simulation
- **Is the code implementing the protocol available? Under which license?** The possible values are: "FREE" if the code is available under an open license, "NO" if the code is protected by a Non disclosure agreement or it has been registered for a third private entity or "UNK" if we do not know how/it the code is available.

[3]Concerning the MPCA protocol, once the MPCA or IPPCA is used for solving particular problem, the agent will use it until a solution is find or timeout will pass. Considering the global "AGENTFLY" project, pilots agent have several different methods to solve collision avoidance (MPCA and IPPCA are just some of them) and they select the best method based on current situation (time to solution, mission goals, environment and so on).

	Design	Pseudo-Code	Roles	Messages	Centralized	Termination
MPCA	HIGH	YES	2	2	YES	YES
APR	UNK	NO	1	2	NO	YES
Waypoint	UNK	YES	1	3	NO	YES
SPAM	UNK	NO	2	4	YES	YES (but with no assurance that all tracks have been managed)
FYPA	Low	YES	3	7	NO	YES (a no-way-out situation is reported to the user)

Table IV
COMPARISON OVER THE DESIGN FEATURES

	Impl.	Lang.	Platforms	Var. Strategy	Code available	User	GUI
MPCA	YES	Java	AGENTFLY/ AGLOBE	NO	YES (for academic purposes)	YES	ON LINE
APR	YES	UNK	FACET	NO	UNK	NO	NO
Waypoint	YES	C++	2005 Cornell RoboCup	YES	FREE	NO	PH
SPAM	YES	Java	FARM	YES	UNK	NO	NO
FYPA	YES	Java	JADE	YES	NO	YES (JADE interface)	OFF LINE, [10]

Table V
COMPARISON OVER THE IMPLEMENTATION FEATURES

- **Can the user interact with the MAS on-line during the MAS execution?** If the value is "YES" it means that the user can change the execution of the system while it is running, using a GUI or other techniques.
- **Which GUI is available?** The value can be "ON LINE" if a GUI exists and the user is allowed to modifies the execution of the system, "OFF LINE" if something exists that shows the user the execution of the protocol (during the execution or later), "NO" if the system only gives the result but is not able to let the user understand the intermediate steps of the protocol. The value will be "PH" if the system is physically implemented so its execution is visible for the user (you can see for example the physical entities moving) but it can not be considered a GUI.

IX. CONCLUSIONS AND FUTURE WORK

This paper complements the one presented at WOA 2009 [12] and further refined in [11], where we discussed the preliminary design and implementation of the FYPA system.

In this paper we propose a systematic comparison among five MARA systems, including FYPA. The comparison is detailed enough to help a developer/scientist looking for implemented MARA solutions, in choosing among them.

From the comparison, a fact emerges: all the analyzed systems (and many others that we took under consideration during our research activity but that we did not report here) are designed to solve very specific problems, although in principle they might be easily generalized (with little or much effort depending on the system) to face similar problems in domains other than those they were designed for.

If this generalization process took place, an *integrated platform for developing negotiation protocols* could be provided to users, instead of many different and not integrated MARA solutions.

As far as FYPA is concerned, we already made the successful effort to generalize the problem that Ansaldo STS posed to our attention, in order to exploit the solution we developed in situations other than the specific Ansaldo STS one [7]. If we will be able to prove that the FYPA protocol is suited to manage even more MARA problems, we could move another step further and design a *platform for developing negotiation protocols*, starting from the FYPA one. This platform could help users to develop new negotiation protocols step by step, starting from the definition of the model (entities, resources and their dependencies) and moving on with the definition of the rules regulating the interaction (specifying the time-outs, the priorities of entities...) and, for example, those to calculate the alternative allocations.

The next extension would be to integrate this *platform for developing negotiation protocols* into the DCaseLP rapid prototyping framework [28], thus resulting into an "Enhanced DCaseLP", in order to allow developers to specify rules regulating the agents behavior using a logical language such as tuProlog.

Finally, as proposed in [9], a further extension to our work could be to integrate the verification capabilities offered by Concurrent MetateM [19] into the "Enhanced DCaseLP" framework.

Regarding the proposal to integrate ontologies in MASs that we presented at WOA 2008 [8], our research group has already extended the Ontology Agent, as described in [30]. The "Enhanced DCaseLP" will take advantage of these results without any further effort since the extended Ontology Agent, being developed in JADE, will be able to be integrated into it

for free.

ACKNOWLEDGEMENTS

This paper is based on Chapter 7 of Daniela Briola's Ph.D. Thesis, [7].

We thank Riccardo Caccia from Ansaldo STS for his help and support during all the stages of FYPA development.

REFERENCES

[1] M. R. Adler, A. B. Davis, R. Weihmayer, and R. W. Worrest. Conflict resolution strategies for nonhierarchical distributed agents. In L. Gasser and M. N. Huhns, editors, *Distributed artificial intelligence: vol. 2*, pages 139–161, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[2] *AgentFly: reference homepage*. http://agents.felk.cvut.cz/projects/agentfly/.

[3] A. Agogino and K. Tumer. Regulating air traffic flow with coupled agents. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 535–542, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[4] M. Barbuceanu and M. S. Fox. Cool: A language for describing coordination in multi agent systems, 1995.

[5] K. D. Bilimoria, B. Sridhar, G. B. Chatterji, K. S. Shethand, and S. R. Grabbe. Future atm concepts evaluation tool. In *Air Traffic Control Quarterly , 9(1)*, 2001.

[6] M. Boman, P. Davidsson, N. Skarmeas, K. Clark, and R. Gustavsson. Energy saving and added customer value in intelligent buildings. *Building*, 1:505–516, 1998.

[7] D. Briola. *Negotiation in Multiagent Systems: Protocols, Ontologies and Applications*. PhD thesis, DISI, University of Genova, Italy, 2011.

[8] D. Briola, A. Locoro, and V. Mascardi. Ontology agents in fipa-compliant platforms: a survey and a new proposal. In *From Objects to Agents Workshop, WOA 2008, Proceedings*, 2008.

[9] D. Briola, M. Martelli, and V. Mascardi. Specification, simulation and verification of negotiation protocols in a unified agent-based framework (extended abstract). In *ICTCS 2010: 12th Italian Conference on Theoretical Computer Science*, 2010.

[10] D. Briola and V. Mascardi. Design and implementation of a NetLogo interface for the stand-alone FYPA system. *In this volume*.

[11] D. Briola, V. Mascardi, and M. Martelli. Intelligent agents that monitor, diagnose and solve problems: Two success stories of industry-university collaboration. In *Journal of Information Assurance and Security*, volume 4, pages 106–117, 2009.

[12] D. Briola, V. Mascardi, M. Martelli, R. Caccia, and C. Milani. Dynamic resource allocation in a MAS: A case study from the industry. In *From Objects to Agents Workshop, WOA 2009, Proceedings*, 2009.

[13] N. Carver and V. Lesser. A new framework for sensor interpretation: Planning to resolve sources of uncertainty. In *In Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 724–731, 1991.

[14] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. A. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica (Slovenia)*, 30(1):3–31, 2006.

[15] D. Cockburn and N. R. Jennings. Archon: A distributed artificial intelligence system for industrial applications, 1995.

[16] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artif. Intell.*, 42:213–261, March 1990.

[17] K. Decker. Designing behaviors for information agents. In *In Proceedings of the 1st Intl. Conf. on Autonomous Agents*, pages 404–412. ACM Press, 1997.

[18] E. H. Durfee and J. S. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. In *Proc. 13th Intl Distributed Artificial Intelligence Workshop*, pages 94–104, 1994.

[19] M. Fisher. A survey of concurrent metatem - the language and its applications. In *Proceedings of the First International Conference on Temporal Logic*, ICTL '94, pages 480–505, London, UK, 1994. Springer-Verlag.

[20] B. Grosz and C. Sidner. Plans for discourse. In P. Cohen, J.Morgan, and M. Pollack, editors, *Intentions in Comm.*, pages 417–444. MIT Press, Cambridge, Mass., 1990.

[21] B. Horling, R. Mailler, and V. Lesser. Farm: A scalable environment for multi-agent development and evaluation. In A. G. C. Lucena, J. C. A. Romanovsky, and P. Alencar, editors, *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237. Springer-Verlag, Berlin, February 2004.

[22] B. Hubermanand and S. Clearwater. A multi-agent system for controlling building environments. In V. Lesser, editor, *Proceedings of the 1st International Conference on Multiagent Systems, 1995 June 12-14; San Francisco, CA*, volume 1, pages 171–176, 1995.

[23] N. R. Jennings. *Cooperation in industrial multi-agent systems*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1994.

[24] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[25] S. Lander and V. Lesser. Sharing Meta-Information to Guide Cooperative Search Among Heterogeneous Reusable Agents. *IEEE Transactions on Knowledge and Data Engineering*, 9(2):193–208, January 1997.

[26] V. R. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11:133–142, 1999.

[27] R. Mailler, V. Lesser, and B. Horling. Cooperative negotiation for soft real-time distributed resource allocation. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 576–583, New York, NY, USA, 2003. ACM.

[28] V. Mascardi, M. Martelli, and I. Gungui. DCaseLP: a prototyping environment for multi-language agent systems. In M. Dastani, A. E.-F. Seghrouchni, J. Leite, and P. Torroni, editors, *In Proceedings of the First Workshop on LAnguages, methodologies and Development tools for multi-agent systemS, LADS'007 Post-proceedings*, volume 5118 of *LNCS*, pages 139–155. Springer-Verlag, 2008.

[29] C. L. Mason and R. R. Johnson. Datms: a framework for distributed assumption based reasoning. In L. Gasser and M. N. Huhns, editors, *Distributed artificial intelligence: vol. 2*, pages 293–317, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[30] F. Mulattieri. Progettazione ed implementazione di un ontology agent. Master's thesis, DISI, University of Genova, Italy, 2010.

[31] D. E. Neiman, D. W. Hildum, V. R. Lesser, and T. W. Sandholm. Exploiting meta-level information in a distributed scheduling system. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, AAAI '94, pages 394–400, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[32] T. Oates, M. V. N. Prasad, and V. R. Lesser. Cooperative information-gathering: a distributed problem-solving approach. *IEE Proceedings - Software*, pages 72–88, 1997.

[33] H. Parunak. Manufacturing experience with the contract net. *Distributed Artificial Intelligence*, pages 285–310, 1987.

[34] O. Purwin, R. D'Andrea, and J.-W. Lee. Theory and implementation of path planning by negotiation for decentralized agents. *Robot. Auton. Syst.*, 56(5):422–436, 2008.

[35] A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi-architecture. In *KR'91*, pages 473–484, 1991.

[36] M. P. Singh. Towards a formal theory of communication for multi-agent systems. In *In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91*, pages 69–74. Morgan Kaufmann, 1991.

[37] T. Sugawara and K. Murakami. A Multiagent Diagnostic System for Internetwork Problems. *Proceedings of INET'92*, January 1992.

[38] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:1446–1461, 1991.

[39] U.S. Congress, Office of Technology Assessment. *Electronic Enterprises: Looking to the Future*. U.S. Government Printing Office, 1994.

[40] D. Šišlák, J. Samek, and M. Pěchouček. Decentralized algorithms for collision avoidance in airspace. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 543–550, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[41] R. Weihmayer and R. Brandau. A distributed ai architecture for customer network control. In *IEEE Global Telecommunications Conference, Globecom'90, Proceedings*, pages 656–662. IEEE, 1990.

[42] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.