Towards a Flexible Air Traffic Management: Dealing With Conflicts

Stefan Resmerita
C. Doppler Laboratory for Embedded Software Systems
University of Salzburg
A-5020 Salzburg, Austria
E-mail: stefan.resmerita@cs.uni-salzburg.at

Michael Heymann
Department of Computer Science
Technion-Israel Institute of Technology
Haifa 32000, Israel
E-mail: heymann@cs.technion.ac.il

George Meyer
NASA Ames Research Center
Moffett Field, CA 94035, USA
E-mail: gmeyer@mail.arc.nasa.gov

Abstract

We propose a multi-agent framework for safe navigation in a discrete environment, which allows employment of distributed conflict avoidance procedures while preventing the occurrence of domino effects (propagation of conflicts in the system as a result of conflict resolution). The navigation environment is modeled by a graph. Each agent has a set of alternate trajectories from its source vertex to its destination vertex, where a trajectory represents a sequence of resources (graph vertices and occupancy times) to be successively occupied by the agent. A safety requirement, that specifies a legal occupancy of a resource by the agents in the system, must be satisfied. Resources where the safety condition is violated are called contested resources and represent conflicts between the involved agent trajectories. Contested resources are prioritized over the competing agents, and an agent can have different priorities for different contested resources. We present a methodology for selection by each agent, of a subset of its set of available trajectories, such that the selected trajectories of distinct agents are conflict-free. The proposed methodology yields maximal solutions; that is, solutions in which an agent cannot improve its selected subset without creating conflicts with selected trajectories of other agents. Three types of safety requirements are considered: (1) Mutual exclusion, where no two agents may occupy a resource at the same time, (2) Resource capacity, where each resource has a fixed capacity and the number of agents that share the resource must not exceed the resource's capacity, and (3) Agent capacity, where each agent has a fixed capacity and the safety condition requires that the number of agents that share a resource is at most the capacity of each involved agent. The methodology, which is motivated by the Free-Flight paradigm in Air Traffic Management, is also applicable to various ground transport settings.

1 Introduction

Motivated by the Free-Flight paradigm in Air Traffic Management (ATM), we describe in this paper a multi-agent framework for safe navigation of aircraft equipped with distributed collision avoidance capabilities. The current ATM system is based on centralized control, with limited flexibility for individual aircraft to choose trajectories. The system relies on human operators in local air traffic control (ATC) centers (that are further partitioned into sectors), to track all aircraft along their nominal pathways and control their trajectories so as to insure adequate aircraft separation. Consequently, in order to keep the system manageable, there are strict limits on the number of aircraft that are allowed into a sector, creating capacity and flexibility constraints, which are now being challenged by the increasing numbers of aircraft in the sky.

An intense research effort is currently under way to overcome some of the above mentioned limitations and better utilize existing technological capabilities. A significant part of this research deals with mid-air collision avoidance, where aircraft separation is ensured by requiring aircraft to follow safe maneuvers [4][9][10][14][17]. A broader approach that aims also at increasing individual aircraft flexibility is the Free-Flight/Free-Routing paradigm [11], in which pilots would be allowed to choose optimal trajectories, and possibly be responsible for maintaining flight safety. In addition to the potential savings that could be achieved by airlines (in terms of fuel consumption and travel time), decentralization of control could reduce the workload at ATC's (which would then play a more supervisory role). As a consequence, it might be possible to increase the number of aircraft supervised by ATC, thereby achieving a better utilization of the airspace.

One possible approach to Free Flight implementation is the Distributed Air-Ground Traffic Management (DAG-TM) operational framework [1][2], which includes conditions under which separation responsibility may be delegated from ground-based control centers to individual aircraft which are adequately equipped with on-board conflict detection and resolution capabilities. The main concerns are achieving safety (aircraft separation), efficiency (minimal trajectory deviation) and stability under distributed conflict resolution rules. Here, stability refers to the so-called *domino effect,* where resolution of a local conflict involving a small number of aircraft may trigger a system-wide propagation of conflicts: an aircraft resolves a conflict by a suitable deviation from its nominal path, which may create new conflicts with neighboring aircraft flying along conflict-free routes. This situation may recursively occur for the neighboring aircraft, ultimately leading to trajectory deviations of large numbers of aircraft.

Challenged by the Free-Flight idea, we proposed in previous work [16] a framework for distributed ATM, which, in addition to safety and individual optimality, addressed explicitly the issue of efficient utilization of the air space. In that framework, conflicts were resolved off-line, such that the on-line operation of the system was conflict-free (so the

system was inherently stable). The framework was based on a general methodology for conflict resolution in multi-agent systems introduced in [15]. The methodology was applied to ATM by considering aircraft as autonomous agents and by viewing the 'airspace' as partitioned into cells, so that the required aircraft separation constraint is satisfied by guaranteeing that at most one agent occupies a cell at any time. Thus, the airspace is modeled by an undirected graph, where a vertex corresponds to a cell and an edge represents an adjacency relation between two cells. An agent must travel from an initial vertex to a destination vertex (both of which are specific to each agent). A *resource* is a pair *(vertex, time step)* and an agent travel is represented by a *trajectory*: a finite sequence of resources to be successively occupied by the agent during the travel. The number of agents in the system is not specified and may change with time. An agent can enter the system at any arbitrary (integer) instant of time and exits the system upon task completion.

An agent is required to announce the set of all its optimal trajectories (which are of equal quality to the agent). We call this the agent's *model*. Optimal trajectories are determined by a specified set of performance criteria, a computation not further discussed in the present paper. We assume that an agent can do that. Two trajectories of distinct agents are in conflict if they contain a common resource. To satisfy safety, an agent's movement is restricted to a *legal* subset of its model, called *legal plan*, such that legal trajectories of different agents are conflict-free. An agent follows an arbitrary trajectory in its legal plan. An *incoming* agent enters the resource system (starts up its travel) upon determining a nonempty legal plan, at which time it becomes *active*. An active agent cannot be stopped or suspended while executing its travel. Thus, a *liveness* specification, that insures that an active agent always has a nonempty legal plan, is satisfied.

At the heart of the proposed framework is the mechanism by which agents select their legal paths, so as to insure safety, liveness, and efficient resource utilization. The proposed methodology consists of two algorithmic phases, preceded by an initialization phase. First, an incoming agent determines the subset of its optimal trajectories that are conflict-free with the legal trajectories of the active agents (already in the system). Then, in the *conflict resolution* phase, the agent resolves potential conflicts with trajectories of all other incoming agents. If no legal trajectory is obtained, then the *accommodation* phase is executed, where the agent can request and obtain resources owned by active agents (who will try to accommodate an incoming agent).

In this paper, we describe significant extensions of the above framework that allow active agents to share resources, subject to capacity constraints. Regarding ATM, in the new framework a vertex represents a bigger cell of airspace, within which the separation constraint between multiple aircraft can be satisfied by employing local conflict avoidance procedures. A variety of methods can be used in this respect (see for example [4][9][10][14][17]). To achieve stability, we require satisfaction of capacity constraints that represent upper bounds on the number of agents that may occupy resources.

The focus of this paper is the conflict resolution phase, which now must guarantee safe operation under capacity constraints. Thus, we consider here two relaxations of the original safety requirement:

- At most k_q agents may occupy a resource q. The number k_q is called the *resource capacity* associated with q. In the case of ATM, resource capacities can be used to ensure stability of the system under distributed conflict avoidance methods. Moreover, this corresponds to the present situation where operators in ATC centers are dealing with multiple aircraft in a sector and there is an upper bound on the number of aircraft that are in the same sector at any time.

- Each agent has certain conflict resolution abilities expressed by the *agent's capacity*: The capacity of agent i represents the maximum number of agents with which agent i is able to resolve conflicts. It is required that the number of agents occupying a resource does not exceed the capacity of each involved agent. In ATM, the agent capacity is a measure of the aircraft's on-board capabilities for resolving conflicts with neighboring aircraft.

It can be seen that the original framework is a special case of both extensions, where all capacities are equal to 1. This paper presents the non-trivial generalization of the previous conflict resolution phase to both cases. In relation to ATM, the main properties of the original approach (safety, flexibility of path choices, efficient utilization of the air space) are preserved. In addition, the new framework ensures stability under a variety of distributed conflict avoidance procedures, with all the attached benefits: more aircraft in the system, robustness, amenability to a gradual implementation in the current ATM system.

A literature review on conflict resolution in ATM can be found in [7]. A stability evaluation of two distributed conflict resolution methods is presented in [1]. Decentralized conflict avoidance rules that guarantee stability of intersecting aircraft flows are presented in [13]. Multi-agent based approaches to Free Flight can be found for example in [8][18]. A token-based framework for decentralized ATM is described in [3].

2 Agent Models

Let V be the set of graph vertices. A *resource* is a pair (v,t), where $v \in V$ and $t \in \mathbb{N}$. An *agent model* is a finite set $\mathcal{P}$ of trajectories, where an *agent trajectory* is a sequence of the form:

$$p : (v_0,0) \to (v_1,1) \ldots \to (v_{m-1}, m-1) \to (v_m, m),$$

where $v_j \in V$, $j = 0, m$. The number $k \in \{0,\ldots,m\}$ represents the time step of this trajectory's execution at which v_k

is occupied by the agent. The same resource may be contained in more than one trajectory in the agent model. The vertex v_0 represents the initial vertex, which is the same in all trajectories of the same agent. Also, v_m is the final vertex, which is the same in all trajectories of the same agent.

We use n to denote the number of agents and $\mathcal{P}_i$ to represent the model of agent i ($i = 1,...,n$). A resource is called a *shared resource* between two trajectories of distinct agents if both trajectories contain the resource. Given a safety requirement, a shared resource is called a *contested* resource, or a *conflict*, if the safety requirement is violated at the resource. A collection of sets of trajectories $(L_1,\ldots,L_n)$ with $L_i \subseteq \mathcal{P}_i$ is *conflict-free* if it contains no contested resources.

Given a set of n agent models and a safety requirement, for the purpose of conflict resolution it is sufficient to consider only agent trajectories that contain contested resources. Moreover, resources that are not contested are of no interest (and consequence) to the resolution of conflicts. Therefore, instead of working with full agent trajectories, we shall consider reduced trajectories, containing only the contested resources. For simplicity of presentation, initial and final resources are also included in the reduced trajectories. The *reduced trajectory* associated to trajectory p is the sequence $r(p):(v_0,0)\rightarrow(x_1,t_1)\rightarrow(x_2,t_2)\rightarrow\ldots\rightarrow(x_h,t_h)\rightarrow(v_m,m)$, where $(x_1,t_1),(x_2,t_2),\ldots,(x_h,t_h)$ are all the contested resources contained in p (except for the initial and final resources), with $t_k > t_{k-1}$, for $k = 2,...,h$.

To illustrate some of the above terminology, consider the following example. Figure 1 represents a portion of 2D space partitioned into cells and three agents, denoted by *R*, *S*, and *T*. Agent *R* must travel from cell *A1* to cell *C21*, *S* from *C1* to *B21*, and *T* from *B1* to *D21*. The agent models are represented in Table 1, where each column corresponds to a time step and each row lists the sequence of cells representing an agent trajectory. Shared resources are outlined in boldface.

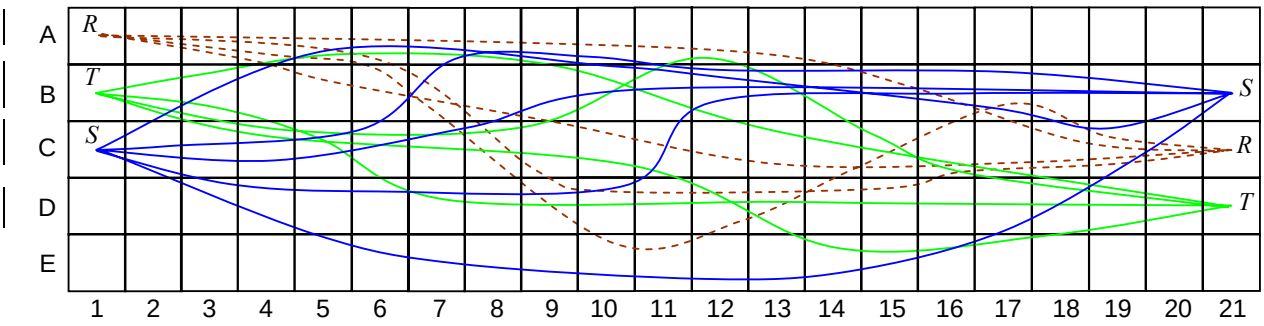

Figure 1. Cell partitioning example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	A1	A1	A2	A2	A3	A3	A3	A4	A5	A6	A7	A8	A9	A10	A10	**A11**
	A1	A2	A3	A4	A5	A6	**B6**	B6	B7	C7	C8	C8	D8	D9	D9	D10
	A1	A2	A3	A4	**B4**	B5	**B6**	B6	B7	B8	B8	B8	**B9**	C9	C10	**C11**
	A1	A2	A3	A4	A5	A6	**B6**	B6	B7	B8	C8	C8	C9	C9	D9	D10
S	C1	C2	B2	B3	**B4**	A4	A5	A6	A7	A8	A9	A10	B10	B11	B12	B13
	C1	C2	C3	C4	C5	C6	**B6**	B7	A7	A8	A9	A10	A10	A11	A11	**A11**
	C1	C2	C3	D3	D4	D5	D6	D7	D7	D7	D8	D9	D10	D11	D11	**C11**
	C1	C2	D2	D3	D4	D5	E5	E6	E6	E7	E7	E8	E8	E9	E9	E10
	C1	C2	C3	C4	C5	C6	C7	C8	B8	B9	B9	B9	**B9**	B9	B10	B11
T	B1	B2	B3	B4	**B4**	C4	C5	C6	D6	D6	D7	D8	D9	D10	D10	D11
	B1	B2	B3	C3	C4	C5	C6	C7	C8	C9	C9	C9	C10	C10	C11	**C11**
	B1	B2	B3	B4	**B4**	B4	A4	A5	A6	A7	A8	A9	**B9**	B10	B11	B12
	B1	B2	B3	C3	C4	C5	C6	C7	C7	C8	C9	C9	**B9**	B10	B11	**A11**
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
A12	A13	A13	A14	A14	B14	B15	B16	B17	C17	**C18**	C18	C18	C18	C19	C20	C21
E10	E11	E12	D12	D13	D14	C14	C15	C16	B16	B17	**B18**	C18	C18	C19	C20	C21
C11	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21					
D10	D11	D12	D12	D13	D14	**D15**	D15	D16	D16	C16	C17	C18	C18	C19	C20	C21
B14	B15	B15	B16	B16	B17	B17	B18	B18	B18	**C18**	C19	C20	C20	B20	B21	
B11	B11	B12	B13	B14	B15	B16	B17	B18	B18	B18	**B18**	B19	B20	B21		
C11	B11	B12	B13	B14	B15	B16	B17	B18	B19	B20	B21					
E10	E10	E11	E12	**E13**	E13	E14	E15	E16	E17	D17	D18	D19	C19	C20	B20	B21
B11	B11	B12	B13	B14	B15	B16	B17	B18	B19	B20	B21					
D12	D13	D14	D14	D15	D15	**D15**	D16	D17	D18	D19	D20	D21				
D11	D12	D13	D13	**E13**	E14	E15	E16	E17	E18	D18	D19	D20	D21			
B12	C12	C13	C14	C15	C16	C17	C18	D18	D19	D20	D21					
A11	A12	A12	B12	B13	B13	B14	C14	C15	C16	C17	D17	D18	D19	D20	D21	

Table 1. Example of full agent models

3 Conflict Resolution

Consider the situation when several incoming agents want to enter the system at the same time. They have access to a common database, containing the models and current legal plans of all active agents in the system. They must also register their optimal trajectories in the database. Thus, after registration, an incoming agent will know the models of all other incoming agents. Initially, an incoming agent determines the subset of its trajectories that are conflict-free with the legal plans of the active agents. However, this subset may have resources that are contested with the corresponding subsets of other incoming agents. Thus, an incoming agent must determine which of these trajectories are ultimately legal, so that legal trajectories of different incoming agents are conflict-free. This is the *conflict resolution* phase. If the resultant legal plan of the incoming agent contains at least one legal trajectory, the agent becomes active. Otherwise, it

executes the accommodation phase, which is not further considered in this paper.

A solution of the conflict resolution problem for a given set of incoming agents is a collection of legal plans that are conflict free. A solution *S* is *less restrictive* than another solution *S'* if, for each agent, the legal plan in *S'* is included in the legal plan given by *S*, and the inclusion is strict for at least one agent. A solution is *least restrictive*, or *maximal*, if no other less restrictive solution exists. While maximal solutions need not be unique, a maximal solution means that no agent can unilaterally improve its legal plan without creating a conflict with some other agents' legal plans (and thus violating the safety constraint). An algorithm that always finds a maximal solution is called *optimal*.

The following safety requirements are considered in this paper:

- **Resource capacity:** Each resource has a specified capacity. The number of agents that may occupy a resource must not exceed the capacity of the resource.
- **Agent capacity:** Each agent has a specified capacity. The number of agents that may occupy a resource must not exceed the capacity of each agent involved.

A special case of both the above requirements is the *mutual exclusion* requirement: no two agents are allowed to occupy a vertex at the same time. In this case, every shared resource is contested.

Our algorithmic approach to conflict resolution is based on a prioritization of agents over contested resources. This prioritization can be obtained by evaluation of domain specific attributes at every contested resource. A possible evaluation method is to use an ordered set of rules that is applied to each pair of agents involved in a conflict, such that the prioritization of an agent pair is established by the first rule that can prioritize the two agents. In other words, rule r_{l+1} is used to prioritize only those agent pairs that could not be prioritized by rules r_1, r_2, ..., r_l. One can obtain conditions under which a pairwise application of a rule set to a conflict yields a total ordering of all agents involved in the conflict. These conditions are not presented here due to space constraints. In ATM, prioritization of aircraft for local conflict resolution has been employed in various settings [8] [9] [12]. Pairwise prioritization was employed to resolve conflicts between rules in logic programming [5][6].

For example, the following rules can be used to prioritize the agents given in Table 1 under the mutual exclusion requirement. We call a trajectory containing resource *q* a *q*-trajectory.

The agent that has priority for a contested resource *q* is the one which:

1. Has a *q*-trajectory by which it spends a smaller amount of time in the vertex of *q* than its opponent, regardless of the *q*-trajectory that is used by the competing agent.
2. Has a *q*-trajectory by which it leaves first the vertex of *q*, regardless of the *q*-trajectory that is used by the competing agent.

3. Has more q-trajectories than the competing agent.

4. Has a shorter q-trajectory to its destination than any of the q-trajectories of the competing agent.

The prioritization obtained by these rules is denoted by π and is given in Table 2. We employ here the convention that a lower number means a higher priority. For example, both R and S spend one unit of time in (*B4,4*), therefore each of them has priority over T. Agents R and S are prioritized for (*B4,4*) by rule 4, which gives priority to R because R has a shorter trajectory from (*B4,4*) to its destination. For (*B9,12*), both R and T have priority over S by rule 1, and T has priority over R by rule 3. For (*A11,15*), rule 1 suffices to prioritize all the three agents. Agents S and T are prioritized for (*C11,15*) by rule 2. For simplicity in the sequel, contested resources are denoted by *a,b,c,...*, as shown in the table.

Resource	*Notation*	$\pi(R)$	$\pi(S)$	$\pi(T)$
(B4, 4)	a	1	2	3
(B6, 6)	b	2	1	
(B9, 12)	c	2	3	1
(A11, 15)	d	1	3	2
(C11, 15)	e	3	2	1
(C11, 16)	f	2	1	
(E10, 16)	g	1	2	
(E13, 20)	h		2	1
(D15, 22)	i	1		2
(C18, 26)	j	2	1	
(B18, 27)	k	1	2	

Table 2. Prioritization example

For the remainder of the paper, unless otherwise stated, by *resource* we shall mean *contested resource*.

Our approach to conflict resolution in the mutual exclusion case (see [15] [16]) was based on the following optimality principle. An agent acquires a resource if and only if the following conditions are both met:

O1) The agent has the *highest priority* for the resource among all agents that *have access* to the resource.

O2) The agent can make successful use of the resource by *completing a legal execution*.

The notion of access is defined inductively: an agent has access to its initial resource and it has access to a non-initial resource q if condition O1 is satisfied for all resources preceding q on an agent's trajectory. For a resource q that satisfies O1, condition O2 means that all resources succeeding q on an agent's trajectory also satisfy condition O1.

This, in particular, implies that an agent is not permitted to reserve a resource (for which it may have priority) if the reservation cannot be applied toward a successful task completion. Indeed, it is easily seen that if condition O2 had not been imposed, the obtained solution might not be optimal. This is because if an agent acquires a resource based only on condition O1, it is possible that the resource may actually not be used in a legal execution, because each trajectory

containing the resource includes also subsequent resources acquired by other agents. Thus, although acquired by the agent, the agent would not be able to utilize such a resource. The system would consequently be under-utilized and hence, such an algorithm would not be optimal.

In this paper, we show how to apply the above principles for achieving optimal conflict resolution in the cases of resource capacity and agent capacity. To this end, we first review the mutual exclusion algorithm.

3.1 Mutual exclusion

In [15], we presented resolution algorithms for the mutual exclusion case under different conditions regarding the knowledge available to an agent about the other agents' models and prioritization. Here, we briefly present only the situation where each agent knows the models of the other agents as well as the prioritization.

The optimality principle is implemented by a rule for acquiring full trajectories, as follows. An agent *claims* a resource q if and only if it has the highest priority for q among all agents that have access to q. An agent has access to q if it has claimed all resources preceding q on at least one of the agent's trajectories. An agent *acquires* a full trajectory if the agent can claim all resources contained in the trajectory. An iterative execution of the algorithm consists of the following steps:

1. **Resource claiming:** Each agents claims resources until no further claims can be made on the current working sets of agent trajectories. At this stage, each resource is either claimed by some agent or unclaimed (if no agent could claim the resource).

2. **Trajectory acquisition:** Each agent acquires (designates as legal) all of its trajectories that have all resources claimed (by the agent).

3. **Trajectory removal:** All trajectories that have conflicts with the newly acquired trajectories are designated as illegal. All legal and illegal trajectories are removed from the working sets of agent trajectories. This opens the way for new possible claims of resources contained in undecided trajectories, which are neither legal nor illegal at this step.

4. **Stopping condition:** If no undecided trajectory remains in the system, then stop. Else, go to step 1 (begin a new iteration).

We illustrate the resolution algorithm on the agents given in Table 1. The reduced models for the mutual exclusion case are schematically depicted in Figure 2, which employs the resource notation given in Table 2.

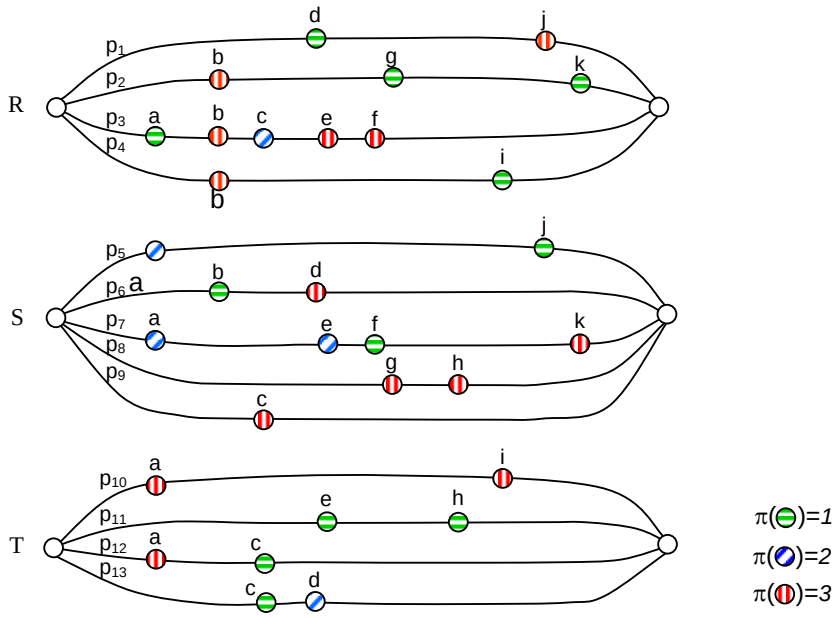

Figure 2. Reduced agent models

Agent *R* has access to *a* and *d* for which it also has priority; therefore *R* claims *a* and *d*. Similarly, *S* claims *b*, and *T* claims *c* and *e*. Since *S* is blocked at *a* (having access to *a* but no priority), it has no access to *j*. Thus, *R* is the only agent with access to *j*, and consequently *R* claims *j*. *R* is blocked at *b* (by *S*), having no access to *g*. Therefore, *S* is the only agent with access to *g*, so it claims the resource. *T* has access to *h* (having claimed *e*) and has also priority for the resource, so *T* claims *h*. No further claims can be made at this step. Resources *f*, *i* and *k* remain unclaimed.

Since agent *R* has claimed all contested resources on trajectory p_1, *R* acquires p_1. Similarly, *T* acquires p_{11}. These are all trajectories that can be designated as legal in the first iteration.

Since p_5 and p_6 have conflicts with p_1 (at *j* and *d*, respectively), they are now designated as illegal. Also, p_{13} is illegal due to the conflict with p_1 at *d* and p_3, p_7 and p_8 are illegal due to the conflicts with p_{11} at *e* and *h*, respectively. All the legal and illegal trajectories are now removed.

The second iteration starts with the sets of undecided trajectories as shown in Figure 3. Notice that the only contested resources here are *c* and *i*. In the claiming step of the second iteration, *R* claims *b*, *g*, *k* and *i*, while *T* claims *a* and *c*. The trajectories p_2, p_4 and p_{12} are designated as legal, while p_9 and p_{10} are illegal. No undecided trajectory remains and the algorithm stops. The maximal solution of the conflict resolution problem is as follows: $LP_R = \{p_1, p_2, p_4\}$, $LP_S = \varnothing$, $LP_T = \{p_{11}, p_{12}\}$, which is shown also in Figure 4. Notice that agent *S* obtained no legal trajectory.

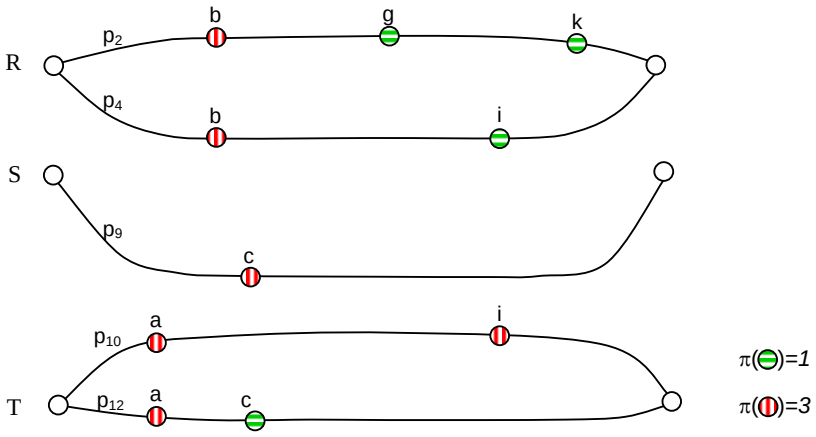

Figure 3. The second iteration in the mutual exclusion example

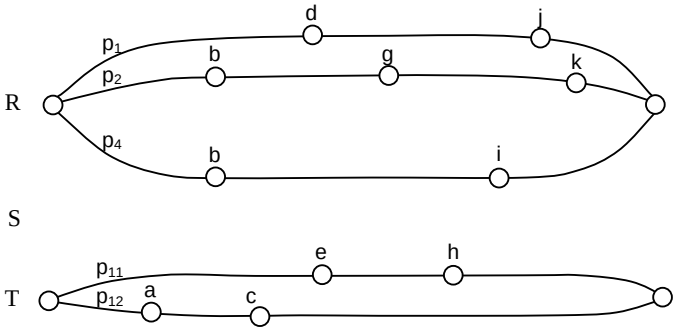

Figure 4. The solution for the mutual exclusion example

In [15], we discussed a variety of possible approaches, as follows. On one hand, the most that an agent can do is to take into consideration all the agents and conflicts in the system, including those in which it is not directly involved (assuming it knows the prioritization for all of them). For this case, the resolution algorithm is optimal. On the other hand, the least that an agent should do is to take into consideration only the agents with which it has conflicts, and to ignore the others. Conflicts are resolved with each of the competing agents, pairwise, the final result being the intersection of the partial results. Since the number of agents conflicting with a given one is usually much lower than the number of all agents in the system, the pairwise approach is computationally efficient, but the result is, in general, not maximal.

3.2 Resource capacity

Let us consider the situation where each resource has a capacity, expressed as a (nonzero) natural number. The safety constraint requires that the number of agents that may occupy a resource must not exceed the resource's capacity.

In ATM, the resource capacity can be used to represent the maximum number of aircraft that are allowed to be in the same airspace cell at the same time. This number depends on the ability of ground controllers to safely route air traffic in the corresponding control sector. This upper bound may vary with time, depending on weather, time of day, etc. This time variance is accommodated in our framework by assigning capacities to resources (which have a time component) rather than to graph vertices.

The structure of the conflict resolution algorithm is mostly the same as in the mutual exclusion case. To achieve the safety requirement, the claiming principle and the trajectory removal are modified as follows.

At the beginning of an iteration, every contested resource in the current working sets of trajectories has an associated *available capacity*, which represents the difference between the initial capacity and the number of agents that acquired the resource in previous iterations. It can be readily seen that the safety requirement is satisfied if, in every iteration, the number of agents that acquire a resource does not exceed the resource's available capacity at that iteration. If the number of agents that acquire a resource in an iteration reaches the available capacity of the resource, then the initial resource capacity is reached and the available capacity of the resource becomes zero, blocking further acquisitions in subsequent iterations. Consequently, all unacquired trajectories containing the resource are designated as illegal and eliminated from the working sets of trajectories. This implies, in particular, that the available capacities of all contested resources considered in an iteration are positive.

Notice that an agent acquires a resource only after it has claimed it (along with all the other resources on a trajectory). To ensure that the number of agents that acquire a resource in an iteration does not exceed the resource's available capacity, we require that the number of agents that *claim* the resource does not exceed this capacity. This can be easily checked if agents that have access to the resource claim it in order of priorities (highest priority first). Thus, an agent claims the resource if and only if the number of all higher-priority agents that have claimed the resource is smaller than the available capacity of the resource.

Remember that the conflict resolution algorithm involves only idle agents, i.e., agents that are not currently executing tasks. In our conflict resolution framework, liveness of active agents is guaranteed by ensuring that no legal trajectory of an active agent conflicts with trajectories designated as legal for idle agents in the ensuing conflict resolution phase. In the mutual exclusion case, this was done by simply eliminating from the input of the conflict resolution phase all trajectories of idle agents that shared resources with legal trajectories of active agents. This must be changed, since now idle agents may share resources with active agents so long as resource capacities are not exceeded. To ensure liveness of active agents, the initial available capacity of each resource is set to be the difference between the resource capacity and the number of active agents that have legal trajectories which share the resource. Then, all the trajectories of idle agents that have at least one resource whose initial available capacity equals zero, are removed. The conflict resolution algorithm is executed on the remaining trajectories. Thus, only the resource slots which are not occupied by active

agents are made available to the idle agents for conflict resolution.

The conflict resolution algorithm for the resource capacity case has the following iteration structure:

1. **Resource claiming:** Each agent claims resources until no further claims can be made on the current working sets of agent trajectories. An agent claims a resource if it has access to the resource and if the number of all agents that have access to the resource and have higher priority than the agent, is smaller than the current available capacity of the resource.

2. **Trajectory acquisition:** Each agent acquires (designates as legal) all of its trajectories that have all resources claimed (by the agent).

3. **Capacity update:** For each resource on a trajectory acquired in the previous step of the current iteration, a new available capacity is computed as the difference between the previous available capacity and the number of agents that have acquired the resource in the previous step.

4. **Trajectory removal:** Each trajectory that has not been acquired at step 2 and that has a resource with available capacity equal to zero is designated as illegal. All legal and illegal trajectories are removed from the working sets of agent trajectories.

5. **Stopping condition:** If no undecided trajectory remains in the system, then stop. Else, go to step 1 (begin a new iteration).

For example, consider the agent models represented in Figure 2, with the initial capacities of *a*, *b*, *g*, *h*, *i* and *j* equal to *1* and the initial capacities of *c*, *d*, *e*, *f* and *k* equal to *2*. The prioritization is according to Table 2. For simplicity, in this example by capacity we mean available capacity.

Similarly to the mutual exclusion case, agent *R* claims *a* and *d*, *S* claims *b* and *T* claims *c*, *e* and *h*. Notice that only *S* and *T* have access to *c*, which has a capacity of *2*. The number of agents that have higher priority than *S* for *c* equals *1*, which is smaller than the capacity of *c*, therefore also *S* claims *c*. All three agents have access to *d*, which is claimed by *R* and *T*. *S* claims *g*, being the only agent with access to *g*. Since *S* has no access to *j*, *R* claims *j*. No further claims can be made in the system at this step. Resources *f*, *i* and *k* remain unclaimed.

Agent *R* acquires p_1, *S* acquires p_9 and *T* acquires p_{11} and p_{13}. All these are designated as legal. The available capacities of *c*, *d*, *h* and *j* become zero. The available capacity of *e* changes to *1*. The available capacities of all the other resources remain unchanged.

The following trajectories are illegal: p_3 (due to *c*), p_5 (because of *j*), p_6 (due to *d*) and p_8 (due to *h*). All legal and illegal trajectories are removed, and the second iteration starts with the undecided trajectories shown in Figure 5.

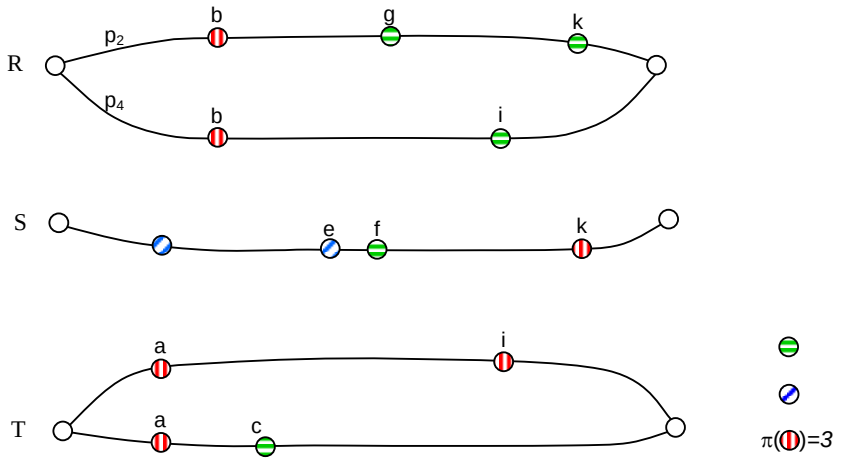

Figure 5. The second iteration for the resource capacity example

The claiming step of the second iteration proceeds as follows: *R* claims *b*, *g*, *k* and *i,* and *S* claims *a*, *e*, *f* and *k* (which has a capacity of *2*). No further claims can be made. Agent *R* acquires p_2 and p_4, while agent *S* acquires p_7. The available capacity of *a* becomes zero, therefore trajectories p_{10} and p_{12} are designated as illegal. No undecided trajectory remains, so the algorithm stops with the following solution: $LP_R = \{p_1, p_2, p_4\}$, $LP_S = \{p_7, p_9\}$, $LP_T = \{p_{11}, p_{13}\}$, which is shown also in Figure 6. Notice that this maximal solution cannot be compared in terms of set inclusion with the one obtained for the mutual exclusion case, even though the mutual exclusion is a particular situation where all resource capacities are equal to *1*. However, as expected, this solution provides more flexibility due to the legal sharing of resources between different agents.

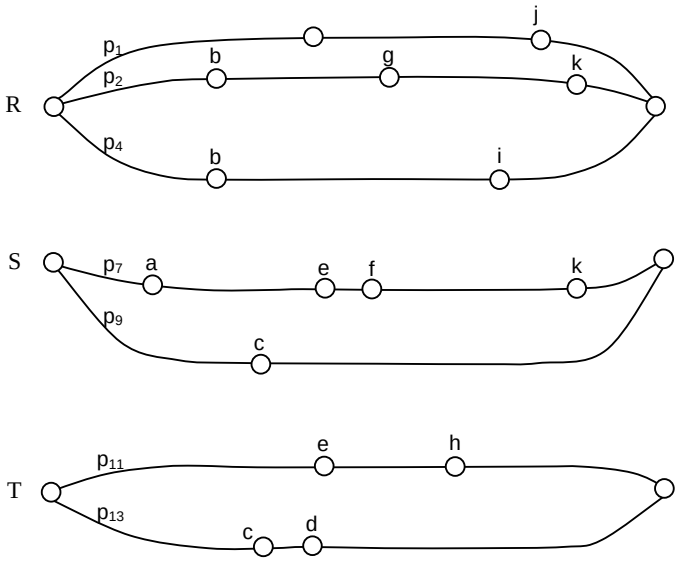

Figure 6. The solution for the resource capacity example

3.3 Agent capacity

In this section we deal with the case where each agent has a capacity, expressed as a (nonzero) natural number. The safety constraint requires that the number of agents that may occupy a resource does not exceed the capacity of each agent involved.

In ATM, the capacity of an aircraft represents, for example, the maximum number of aircraft with which the aircraft is able to avoid mid-air collisions. Different aircraft may be equipped with different generations of on-board collision avoidance systems, where newer generations are able to resolve conflicts involving more aircraft. Clearly, one has to ensure that newer generations will be compatible (and hence be able to coexist) with older ones.

One might try to approach the conflict resolution problem for agent capacities by casting it into a problem involving only resource capacities and then using the algorithm outlined in Section 3.2: set the capacity of each contested resource as the minimum of all the capacities of the agents involved in the conflict. It can be seen that maximality is not guaranteed in this case, since an agent with minimum capacity may not necessarily acquire the resource.

Similarly to the case of resource capacity, we associate an *available capacity* to each resource acquired by some agent. At the beginning of an iteration, the available capacity of a resource is equal to the minimum of the capacities of all agents that have acquired the resource prior to the current iteration minus the number of these agents.

For example, suppose that prior to the current iteration a resource q has been acquired by three agents: R with capacity 5, S with capacity 6, and T with capacity 7. Then, for the current iteration, the available capacity of q equals 2.

The safety requirement is satisfied by imposing the following condition: The agents that acquire a resource in an iteration are such that the available capacity of the resource for the next iteration is nonnegative. In the above three-agent example, this condition is violated if the resource is acquired in the current iteration by:

- an agent of capacity 3, or
- two agents, one of which has capacity 4, or
- more than two agents.

The acquisition condition is enforced by an adequate claiming of resources. Similarly to the resource capacity case, agents claim a resource in order of their priorities (highest priority first). An agent claims a resource q if and only if the following conditions are all met:

C1) The agent has access to q. That is, the agent has previously claimed all resources preceding q on at least one of the agent's trajectories.

C2) The agent's capacity is greater than the number of agents which have acquired q prior to the current

iteration plus the number of (higher priority) agents that have already claimed q in the current iteration.

C3) The minimum of the capacities of all agents that have acquired q prior to the current iteration or have already claimed q in the current iteration is greater than the number of these agents.

Similarly to the previous algorithms, illegal trajectories are determined at the end of an iteration, after the trajectory acquisition step. An unacquired trajectory of an agent i is designated as illegal if it contains a resource q such that one or both of the following conditions are met:

I1) The capacity of agent i is smaller than or equal to the total number of agents that have acquired q (including previous iterations). This means that agent i will never acquire q in subsequent iterations.

I2) The available capacity of q becomes zero for the next iteration. This means that the total number of agents that have acquired q is equal to the capacity of one of these agents. Thus, no agent will acquire q in subsequent iterations.

All illegal trajectories are eliminated from the working sets of trajectories for the next iteration.

Notice that liveness of active agents is ensured by the above rules, since active agents sharing a resource are agents that have acquired the resource prior to any iteration of the conflict resolution algorithm. The structure of the conflict resolution algorithm is given below. Initially, conditions *I1* and *I2* are checked to determine and eliminate initial illegal trajectories of idle agents which are in conflict with legal trajectories of active agents. Then the following iterative execution is applied on the remaining sets of agent trajectories of idle agents:

1. **Resource claiming:** Resources are claimed according to rules *C1, C2,* and *C3*, until no further claims can be made on the current working sets of agent trajectories.

2. **Trajectory acquisition: :** Each agent acquires (designates as legal) all of its trajectories that have all resources claimed (by the agent).

3. **Capacity update:** For each resource on a trajectory acquired at step 2 of the current iteration, determine the available capacity of the resource for the next iteration.

4. **Trajectory removal:** Illegal trajectories are determined according to conditions *I1* and *I2.* All legal and illegal trajectories are removed from the working sets of agent trajectories.

5. **Stopping condition:** If no undecided trajectory remains in the system, then stop. Else, go to step 1 (begin a new iteration).

Let us apply this algorithm on the problem consisting of the agent models in Figure 2, the prioritization in Table

2, with capacities of agents *R, S, T* being 1, 2, 2, respectively.

Agent *R* claims resource *a,* for which it has the highest priority. According to rule C3, agents *S* and *T* cannot claim *a* (which has been claimed by one agent of capacity 1). Resource *b* is first claimed by agent *S*. Agent *R* cannot claim *b* (C2 is not satisfied). Resource *c* is claimed by agent *T* via p_{13}. Unlike R, agent *S* has access to *c* and it claims *c* (C1 – C3 are satisfied). Resource *d* is claimed by *R*. Consequently, agents *S* and *T* cannot claim it (C3 is not satisfied). Resource *e* is claimed by *T* (which is the only agent with access to it). Agent *S* is the only agent with access to *g*, hence it claims *g*. Similarly, agent R claims *j*. Resource *h* is claimed by both *S* and *T*. No agent has access to any of the resources *f, i,* and *k*, which remain unclaimed. At this step, no further claims can be made in the system. Trajectories p_1, p_8, p_9 and p_{11} are acquired by the corresponding agents. Next, illegal trajectories are determined by checking conditions I1 and I2 on the unacquired trajectories:

- Trajectory p_2 contains resource *g*, which has been acquired by *S*. Since the capacity of agent *R* is equal to 1, *R* will never be able to acquire *g* (regardless of the capacity of *S*). Condition I1 applies, hence p_2 is illegal.

- Similarly, p_3 is illegal due to resource *e* having been acquired by *T*.

- Trajectory p_4 has no acquired resource at this step, so it is undecided.

- Trajectory p_5 contains resource *j*, which was acquired by agent *R*. Since *R* has capacity 1, the available capacity of *j* for the next iteration is zero. Thus, no agent can further acquire *j* (without violating the capacity of *R*). According to I2, p_5 is illegal.

- Similarly, p_6 is designated as illegal due to resource *d* having been acquired by *R*.

- Trajectory p_7 contains one acquired resource: *e*, which was acquired by *T*. It can be easily seen that I1 and I2 are not satisfied: since both *S* and *T* have capacity 2, *S* might still be able to acquire *e* in further iterations. Thus, p_7 remains undecided.

- Trajectory p_{10} has no acquired resource at this step, so it is undecided.

- Trajectory p_{12} (of agent *T*) remains undecided even though it contains a resource (*c*) acquired by another agent (*S*). As in the case of p_7, conditions I1 and I2 are not satisfied.

- Both resources of trajectory p_{13} were acquired, but not by *T*: *c* was acquired by *S* and *d* was acquired by *R*. The available capacity of *d* becomes zero, I2 applies and p_{13} is illegal.

The legal and illegal trajectories are removed and the second iteration begins with the sets of trajectories shown in Figure 7. Notice that two resources on the undecided trajectories were acquired in the first iteration: *e* and *c*. At the

beginning of the second iteration, each of them has an available capacity equal to 1.

In the claiming step, agent *R* claims *b* and *i*, *S* claims *a, e, f, k*, and *T* claims *a* and *c*. Consequently, agent *R* acquires trajectory p_4, *S* acquires p_7 and *T* acquires p_{12}. Both resources of p_{10} are acquired by other agents. The available capacity of *a* becomes zero and therefore p_{10} is illegal.

The algorithm stops with the following maximal solution: $LP_R = \{p_1, p_4\}$, $LP_S = \{p_7, p_8, p_9\}$, $LP_T = \{p_{11}, p_{12}\}$, as shown in Figure 8.

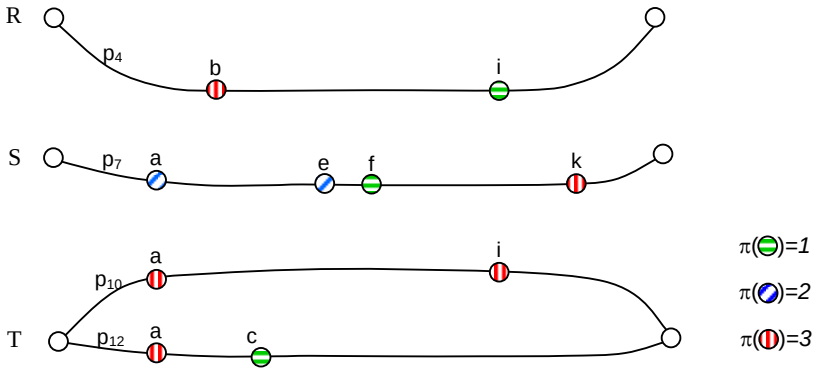

Figure 7. The second iteration for the agent capacity example

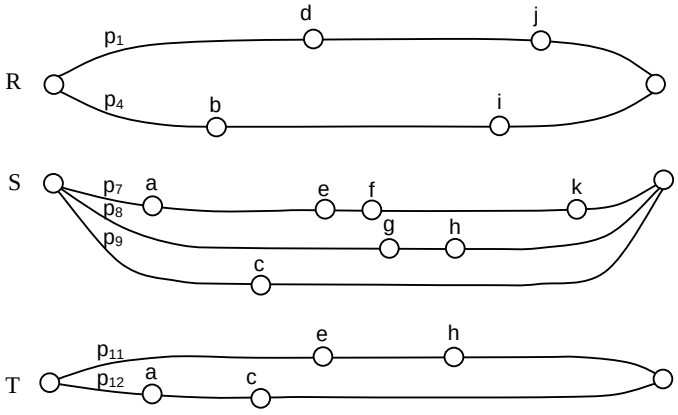

Figure 8. The solution for the agent capacity example

4 Concluding Remarks

A conflict resolution methodology which addresses both the resource capacity and the agent capacity requirements can be readily obtained by combining the algorithms of Sections 3.2 and 3.3. This algorithm can be executed independently by each agent, if the agent has accurate information about the other agents' models, agent and

resource capacities, and about the prioritization rules. Since the algorithm finds legal trajectories incrementally, an agent can terminate its execution of the algorithm at any step, before finding all its legal trajectories. Thus, distinct agents may perform different executions of the algorithm without sacrificing safety of the global solution (but possibly sacrificing maximality).

Regardless of how the algorithm is applied (centralized or distributed), its outcome guarantees system stability when using distributed conflict avoidance methods, due to satisfaction of capacity constraints. This is especially relevant to Air Traffic Management, where sector capacities play a significant role in traffic control and the domino effect is an important consequence of using distributed methods for conflict avoidance. Moreover, if safety is to be delegated to individual aircraft, one has to take into account situations where different aircraft are equipped with different compatible on-board conflict avoidance systems. Taking into account both resource and agent capacity in the same framework allows for a gradual, supervised shifting from the present, sector capacity based air traffic control, to the future, agent capacity based Free Flight.

The distribution of control achieved by means of capacities and local conflict resolution has also advantages regarding the computational complexity of the algorithm. As compared to the mutual exclusion case, the resource system is modeled at a coarser granularity (the same volume of airspace corresponds to a smaller number of bigger cells). Thus, the number of contested resources and the number of agent trajectories can be significantly smaller than in the mutual exclusion case. In fact, granularity of space partitioning is a parameter which establishes the tradeoff between centralized and distributed control in our framework. If capacities are sufficiently large, most of the conflict resolution is executed in a distributed way. For the example of Figure 2, if all resource and agent capacities are equal to 3, then there is no contested resource and all trajectories are immediately designated as legal – the actual conflict avoidance is delegated to the distributed procedures. On the other hand, if all resource capacities are equal to one (the mutual exclusion case), all conflicts are resolved off-line by the algorithm presented in this paper and no conflict occurs in the subsequent operation of the system.

There are many issues which will be further investigated in relation to the framework presented in this paper. An important one is fairness of trajectory allocation to agents. Various optimality criteria may be added to select a maximal solution, e.g., maximum number of legal trajectories, minimum number of shared resources, etc. These may be addressed by appropriate prioritization functions and/or suitable allocation of resource capacities. We plan to develop a simulation environment based on this framework, which will allow performance evaluation of various prioritization policies and distributed conflict avoidance methods.

References

[1] K. D. Bilimoria, K.S. Sheth, H.Q. Lee, and S.R. Grabbe, 2000. Performance evaluation of airborne separation assurance for Free Flight. AIAA 2000-4269.

[2] DAG-TM web page: http://as.nasa.gov/aatt/dag.html.

[3] S. Devasia, M. Heymann and G. Meyer, 2002. Automation procedures for Air Traffic Management: a token-based approach. Proceedings of the ACC02.

[4] V. N. Duong, K. Zeghal, 1997. Conflict resolution advisory for autonomous airborne separation in low-density airspace. Proceedings of the CDC97.

[5] B.N. Grosof, 1997. Courteous logic programs: Prioritized conflict handling for rules. Technical report RC28036, IBM T.J. Watson Research Center.

[6] B.N. Grosof, 1997. Prioritized conflict handling for logic programs. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the International Symposium (ILPS-97)*, MIT Press, Cambridge, MA, pp. 197-211.

[7] J. K. Kuchar and L. C. Yang, 2000. A review of conflict detection and resolution modeling methods. *IEEE Trans. Intell. Transp. Syst. 1(4)*, pp. 179-189.

[8] J. C. Hill, J. K. Archibald, W. C. Stirling, and R. L. Frost, 2005. A Multi-Agent System Architecture for Distributed Air Traffic Control. AIAA 2005-6049.

[9] J. Hu, M. Prandini, A. Nilim, and S. Sastry, 2001. Optimal coordinated maneuvers for three dimensional aircraft conflict resolution, AIAA 2001-4294.

[10] I. Hwang and C. Tomlin, 2002. Protocol-based Conflict Resolution for Finite Information Horizon, Proceedings of the ACC02.

[11] R. Jacobsen, 2000. NASA's Free Flight Air Traffic Management Research, NASA Free Flight/DAG-ATM Workshop.

[12] M. R. Jardin, 2004. Air traffic conflict models. AIAA 2004-6393.

[13] Z.-H. Mao, E. Feron, and K. Bilimoria, 2000. Stability of intersecting aircraft flows under decentralized conflict avoidance rules. AIAA 2000-4271.

[14] P.K. Menon, G.D. Sweriduk, and B. Sridhar, 1999. Optimal strategies for Free Flight air traffic conflict resolution, *Journal of Guidance, Control and Dynamics 22(2)*.

[15] S. Resmerita and M. Heymann, 2003. Conflict Resolution in Multi-Agent Systems, Proceedings of the CDC03.

[16] S. Resmerita, M. Heymann, and G. Meyer, 2003. A Framework for Conflict Resolution in Air Traffic Management, Proceedings of the CDC03.

[17] C. Tomlin, G.J. Pappas, and S. Sastry, 1998. Conflict Resolution for Air Traffic Management: A Study in Multi-Agent Hybrid Systems, *IEEE Transactions on Automatic Control, 43(4)*.

[18] J.P. Wangermann and R.F. Stengel, 1998. Principled negotiation between intelligent agents: a model for air traffic management. *Artificial Intelligence in Engineering (12)*.