A Multi-Agent Approach to Reliable Air Traffic Control

Minh Nguyen-Duc, Zahia Guessoum, Olivier Marin,
Jean-François Perrot, Jean-Pierre Briot

Computer Science Laboratory of Paris 6
104, avenue du Président Kennedy
75016 Paris
France

{minh.nguyen-duc, zahia.guessoum, olivier.marin, jean-francois.perrot, jean-pierre.briot}@lip6.fr

Abstract

Air Traffic Control (ATC) is going to be a typical critical socio-technical system in which controllers use a large number of distributed software tools to provide safety ATC services. The reliability of these services relies on the availability of the various tools. In the process of integrating more and more sophisticated tools in their daily work, controllers need to feel confident in the reliability of their tool set. This paper presents a multi-agent approach to this reliability problem. We propose an agent-based decision-aided system that helps controllers in using their multiple software tools in situations where some tools are not available due to technical incidents. As it is critical to conduct experiments on real air traffic, we build and test our *Multi-Agent System* (MAS) in a simulation environment, thus develop an *Agent-Based Simulation* (ABS). Experimental work on this ABS has demonstrated the significance of our system to air traffic controllers.

1 Introduction

Air Traffic Control (ATC) provides services whose objective is to direct aircraft on the ground and in the air. Its tasks are to separate aircraft, to ensure safe orderly and expeditious flow of traffic, and to give information to pilots, such as weather and navigation information.

1.1 Reliability problem

According to the *First ATC Support Tools Implementation* (FASTI) program [Petricel and Costelloe 2007], the need for support tools for air traffic controllers has never been greater. The forecast growth in air traffic requires the adoption of new technologies and related procedures enabling the safe and efficient provision of ATC services to a larger number of aircraft. Our work is concerned with the next generation of software systems for ATC, which will include software tools. These tools will increase the controllers' capacity at the expense of a complexification of their task. Architecturally, they will be distributed over local area networks (LANs) in each control center and the wide area network (WAN) between centers (see Section 2 for more details).

The reliability of ATC services relies on the availability of the various tools. Indeed, the failure of a tool or of the connection of a tool with a *Controller Working Position* may cause delays or even, in some cases, crashes. To avoid such incidents, mechanisms for fault tolerance must be built into the support system.

Moreover, like in any critical socio-technical system, the integration of sophisticated tools in the controllers' daily work currently faces difficulties. On the one hand, controllers have to change their usual, trusted working procedures. The safety and power that the tools are expected to provide will only become effective if the controllers are able to make the most of the tools' functionalities. And that strongly depends on their familiarity with the tools. On the other hand, the controllers need to feel confident in the reliability of their software tools.

1.2 Fault tolerance

Tools can encounter partial or total failures as a result of internal or external events. Fault tolerance is intended to preserve the delivery of correct service in the presence of active faults [Cristian et al. 1996]. It is generally implemented by error detection and subsequent system recovery, and possibly by error containment. There are mainly two types of approaches to improving the fault tolerance of services: replication (or preventive approaches) and diagnostic (or corrective approaches).

At the heart of all fault tolerance techniques is some form of redundancy. This means that components that are prone to defects are replicated in such a way that if a component fails, one or more of the non-failed replicas will continue to provide services with no appreciable disruption.

Diagnosis informs the user of the damage and of the recovery possibilities. The diagnostic can be relying on organization of agents in the environment, exception handling, monitoring, etc.

Actually, preventive and corrective approaches are complementary [Guessoum et al. 2006] (see also 5.3).

This is our longer term objective, but will not be addressed in this paper.

1.3 Our approach

We adopt a corrective approach, especially applied when faults occur. We argue that the best way to prove the reliability of a support system is to show that the ATC system, as a whole, can still provide full traffic control services when errors suddenly appear. Indeed, if the controllers are timely and adequately informed of the incidents, they can accordingly adjust their current control tasks and the following tasks. They can often manage without some of their tools. According to the *Guidance Material for Contingency Planning* [ESP 2007], this kind of working mode can be seen as a type of *Degraded Mode of Operation*.

Therefore, there exists a need for a decision-aided system that helps the controllers in using their multiple software tools, particularly in situations where some tools are not available, or in other words when technical incidents happen. Our aim is to show that a suitable use of multi-agent technology can help in this respect. To develop such a system, we propose a solution based on software agents (see Section 4).

1.4 Outline of the paper

The paper is organized as follows. Section 2 presents the ATC system and its basic future architecture. Section 3 analyzes a typical example of technical incident. Section 4 proposes our MAS solution. Section 5 describes the development of our ABS for experiments. Then in Section 6, an experiment clearly shows how our agents react to a typical network failure. Section 7 discusses related work. Finally, Section 8 draws a conclusion.

2 Air Traffic Control

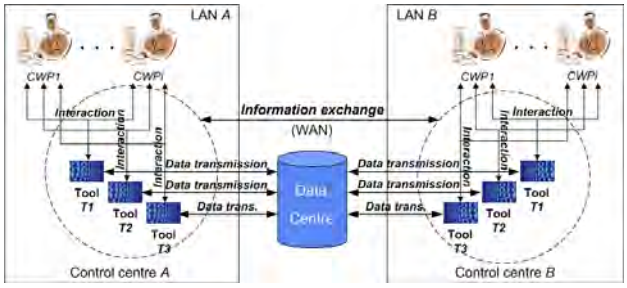

Figure 1. Basic future ATC system architecture.

The current ATC system is airspace-based. The airspace is divided into many sectors whose size depends on the average traffic volume and the geometry of air routes. There are usually two air traffic controllers to handle the traffic in each air sector: an executive controller who communicates with pilots, and a planning controller who plans his colleague's work. Also, the sectors are regrouped into regions each of which is under control of a control center. For example, the Athis-Mons center is responsible for air traffic control in the Parisian region.

The general structure of the ATC system sketched above would not be expected to change. But a new architecture would have to support the introduction of distributed software tools, as discussed above. Figure 1 illustrates a typical application context where two different control centers are connected with a common flight data-processing center through the inter-center network (a WAN). In each control center one (or several) application server(s) host(s) the various software tools in use. These application servers are connected with the *Controller Working Positions* (CWP) by means of a local network (LAN). The LANs of the control centers are connected via the inter-center WAN.

A typical example of support tool is the *Medium-Term Conflict Detection* (MTCD) [Petricel and Costelloe 2007]. Once aircraft trajectories have been predicted, they can be employed to detect medium-term conflicts. There also exist many other tools such as *Short-Term Conflict Alert* (STCA), *MONitoring Aid* (MONA), *Airspace Penetration Warning* (APW), and *Minimum Safe Altitude Warning* (MSAW).

3 Typical Example

In this section, we would like to analyze a typical situation in which a technical incident occurs. It can help with understanding the influence of the incident on the controllers' work and what they need in such situation.

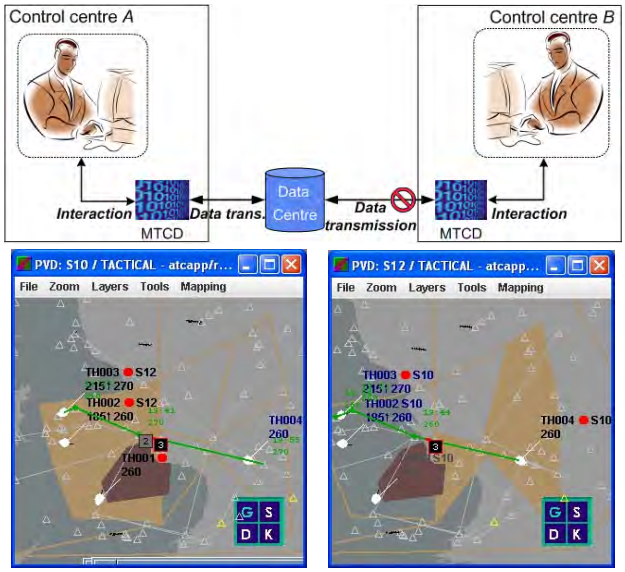

Figure 2. Typical example of a handover situation: the two controller's screens on both sides of the border.

We hence consider two (executive) controllers (named Co_1 and Co_2) responsible for two neighboring sectors (named S_{10} and S_{12}), at the border of two control centers (named *A* and *B*). They are often in handover situations, *i.e.* they have to transfer the control of aircraft flying from one sector to the other. We suppose that at a moment there are several potential conflicts among which a particular one concerns two aircraft: *TH003* flying from S_{10} to S_{12}, and *TH004* flying

in the opposite direction. Moreover, all the conflicts are going to happen in S_{10}.

These conflicts can be automatically detected by Co_1's MTCD or "manually" by Co_1 himself. He does or does not perform control operations to resolve a detected conflict, depending on the real aircraft trajectories which probably evolve before the conflict happens. Since Co_1 can only resolve conflicts one by one, he has to sequence all the detected conflicts to be resolved. Therefore, he needs to decide to (or not to) resolve a conflict at least T minutes before it happens. Indeed, T is common to all the detected conflicts, and it has to be sufficiently large that the controller can perform good conflict sequencing.

We suppose that a network failure occurs at the time when the potential conflicts appear: center B is disconnected from the flight data-processing center (see the basic ATC system architected illustrated by Figure 1). Consequently, a demand for exit flight level change for *TH004* sent by Co_2 to the data-processing center is lost. Accordingly, the flight data of *TH004* are no longer accessible from center A and therefore unusable for Co_1's MTCD.

This failure makes Co_1's MTCD unable to detect conflicts not only for *TH004*, but also for all the aircraft flying from center B. However, it still correctly detects the "local conflicts" that only concern the aircraft flying in S_{10}. So we can see it as "locally available".

We now consider the controller's (Co_1's) reaction to the unavailability of his MTCD. It is supposed that there exists a fault detection equipment which will give him some warning. We are thus interested in when he is informed of the tool unavailability and in which additional information he gets. We would like to underline the two following cases.

In the first case, Co_1 is only aware of the unavailability of his MTCD when some potential conflicts are closely going to happen (in less than T minutes). As discussed above, this will embarrass his conflict sequencing, and can then make him nervous.

In the second case, Co_1 is already aware of the unavailability of his MTCD but does not know its "local availability". He has to detect himself all the potential conflicts. To do that, he verifies and follows all the aircraft he suspects. However in reality, MTCD is not totally unavailable, and still locally available. Such an exhaustive verification will unnecessarily increase Co_1's workload because, in fact, he can still rely on the results given by MTCD for the local conflicts.

In conclusion, we notice that, in situations where some tools are not available, the controllers need not only to be timely informed of the unavailability of the tools, but also to obtain adequate information about their state, *e.g.* the "local availability" of MTCD. One could see that if the controllers are guaranteed to be provided what they require to manage without the unavailable tools, they will feel more confident on the reliability of the support system.

4 Our Multi-Agent System Solution

To solve the problem presented in the previous section, a decision-aided system for the air traffic controllers is needed. Its missions are to communicate with the controllers, to inform them of the environment state and to show them information of tools' availability. More ambitiously, the decision-aided system would be endowed with the capacity to propose corrective actions to be performed following technical incidents.

Besides, this system helps with mitigating the effects of software faults in a distributed environment. It monitors software components which run on different machines, and keeps an eye on the interactions between the users (*i.e.* the controllers) and these components. To this end, it also has to be distributed. It observes complex data (*e.g.* air traffic data) at the input and output of each computation module of any software tool.

Also, this system builds up confidence for users of a safety-critical software system. In consequence, it has to guarantee a safety level with respect to the services it offers. All its monitoring services have to run in real-time so that it can inform the users of some change of the software system's state as soon as it happens. Moreover, information it provides need to be not only concise but also adequate, in such a way that the users can determine exactly what to do in response to this change.

In view of these requirements of the decision-aided system to be developed, we propose a *Multi-Agent System* (MAS) solution. Our MAS communicates with the controllers through assistant agents and monitor the software tools through monitor agents. The capacity of the agents to exchange data with each other will allow acquiring in real-time information to be presented to the controllers.

4.1 Agents

Since our agents have to take care of the monitoring of software tools and of the communication with the controllers, we design different kinds of agent aiming to perform these two common tasks. We currently use two monitoring agents for each tool, *i.e.* a data sentinel and a computation sentinel, and assign an assistant agent to each controller.

1 *Data sentinel agent*: observes the input and output data of a specific software tool and communicates with other agents in order to discover data losses (see Section 6 for more details).

2 *Computation sentinel agent*: observes the input and output data of a specific software tool and communicates with other agents in order to discover computation faults (*e.g.* a computation sentinel checks timeout errors).

3 *Assistant agent*: communicates with other agents in order to determine the automated tools' availability, and informs a controller of this availability; an assistant agent can observe the controller's actions

in such a way that it can notify the monitoring agents of relevant events.

At the individual level, these agents need not to be complicated. A monitoring agent simply reacts to technical incidents that it discovers itself or of which it is notified by other monitoring agents. An assistant agent would only be endowed with some limited reasoning capacity to be able to propose corrective actions to perform (which are predefined) following incidents. The agents' simplicity would bring not only more reactivity, but also more robustness to the MAS.

4.2 Organization of our MAS

On each LAN in a control center we install a group of coordinated agents that are distributed over the whole local network. Each local group of agents is composed of assistant agents and of monitoring agents. We associate with each *Controller Working Position* (CWP) an assistant agent. Each tool instance is observed by monitoring agents. Please note that monitoring and assistant agents may be hosted on any of the machines, or even on additional independent network nodes, as long as they retain the capacity to display information on the controller's screen.

When an incident occurs, the related tool's monitoring agent first discovers the critical situation by using the data it gathers from the tool's input/output, as well as the information it receives from other monitoring or assistant agents. Then, it transmits information about the tool's state to the assistant agents of the CWPs that use this tool. These assistants display green/yellow/red flags on their controller's screen, thereby indicating the tool's total/partial availability, together with the relevant information.

By communicating about observed tools' data and controllers' actions with each other, the agents can notify the controllers of what they should consider following their actions perturbed by the incident. For instance, in the scenario presented in Section 6, the monitoring agents exchange events of request for data change so that they can find out lost data due to network failures. Then, when the controllers see this information shown by their assistants, they know that the related aircraft's flight plan is inconsistent and that they cannot use MTCD anymore to detect conflicts for it.

5 Agent-Based Simulation

5.1 Objective

Of course the future ATC system we have described above has not yet been implemented. Moreover, any novel application to a critical system like ATC has to be tested in simulations before its real world implementation. We embed our agents into the eDEP platform (*Early Demonstration & Evaluation Platform*) [eDEP 2007], which provides a distributed simulated ATC environment with realistic air traffic. In this way, we obtain an *Agent-based Simulation*. Our agents are built with the DimaX platform [DimaX 2007], which helps with developing reliable multi-agent systems.

5.2 ATC simulation platform - eDEP

eDEP [2007] is a Java platform developed by Eurocontrol that uses RMI (*Remote Method Invocation*) to distribute its components over a LAN. It gives a set of standard ATC elements, *e.g.* "airspace" (a database of static airspace information); "integrated air surveillance" (a database of surveillance radar tracks); "initial flight plan" (an initial plan that defines route constraint points and altitude limits); "trajectory predictor" (a trajectory prediction algorithm which uses aircrafts' kinematic models to predict the real motion of a particular aircraft); and *Controller Working Position* (the main graphical interface to the system based on a plan view display of the control sector). The support tools for air traffic controllers, *e.g.* STCA and MTCD, are implemented in eDEP as independent components which can run on different machines.

5.3 Multi-agent platform - DimaX

DimaX is a Java multi-agent platform which gives a generic and modular agent architecture, and allows high heterogeneity in agent types (reactive, deliberative and hybrid). It is in fact based on the extension of modeling and implementation facilities offered by object-oriented languages. In DimaX, an agent at the smallest granularity is simply a single-threaded object, and a complicated agent can be constituted by smaller agents. Also, this platform allows adding new behaviors to any agent by using programming libraries.

Since we would like our MAS to be used in a critical socio-technical system like ATC, the MAS itself has to be reliable. DimaX can help with developing such MAS. This multi-agent platform is in fact the result of the integration of its previous generation (named DIMA – *Development and Implementation of MAs*) and a fault-tolerance framework (named DarX [Marin et al. 2003]), which brings in services, *e.g. Fault Detection Service* and *Replication Service*, which provides transparent support for making MAS fault-tolerant through adaptive replication.

5.4 Overview of our ABS

We manage at least two *Controller Working Positions* (implemented by the CWP component in eDEP), belonging to two different control centers. The LAN of each control center is realized on at least two computers (one for the CWP and the other for the application server). The data-processing center is realized as a separate machine. This machine together with the two LANs make up our image of the inter-center WAN. Each application server runs a copy of each of five tools, *i.e.* MTCD, STCA, MONA, APW, MSAW (also provided as eDEP components).

The integration of our DimaX agents and eDEP components follows the FIPA *Agent Software Integration Specification* [FIPA 2001]. The DimaX platform already includes a generic wrapper agent ready to provide any other agent (*e.g.* a monitoring agent or an

assistant agent) with services which allow this latter agent to connect to software components. Special wrappers are then built by extending the generic one. They need to be hosted on the same machine as the components they "wrap".

Based on the agent model discussed in 4.2, as a first step we install two monitoring agents and two wrapper agents for each of software tools:

1 *XXX_DataSentinel*, *XXX_ComputationSentinel*: observes the XXX[1] component's input/output data and communicates with other agents in order to discover faults;

2 *XXX_ObservationWrapper*, *XXX_GeneralWrapper*: special wrappers which respectively provide XXX observation and general-purpose services to the two other XXX_agents;

Additionally, we endow the CWP with a CWP_Assistant which communicates with other agents in order to determine the automated tools' availability, and shows this availability in its user interface. The following figure illustrates the CWP_Assistant's user interface. It uses green/yellow/red flags to show tools' status.

Figure 3. CWP_Assistant's user interface.

6 Testing Scenario

6.1 Objective

The first tests of our agents on the ABS use several experimental scenarios one of which corresponds to the example presented in Section 3. In this section, we will describe in detail this scenario which will illustrate the reaction of our MAS to the possible unavailability of a tool due to a network failure. This experiment also aims to demonstrate the usefulness of our MAS to the air traffic controllers.

6.2 Experimental setup

The experiment runs on the following connected machines:

1 Two client machines hosting two CWPs for two controllers belonging to two different control centers (named centers *A* and *B*);

2 Two tool servers hosting two MTCD instances for the two control centers;

3 A data server placed in the common flight data-processing center (see 5.4).

[1] XXX stands for the tool name, *e.g.* MTCD or STCA.

6.2 Agent behavior

We are in a handover situation (as described in Section 3). At first, all machines run smoothly and are fully connected in a handover situation (*e.g.* there are aircraft flying from the control center *B* to the control center *A*). Each controller has unlimited access to the tool server on his LAN and can freely obtain the flight data he needs. The assistant agents display green labels indicating that the software tools are working at full capacity.

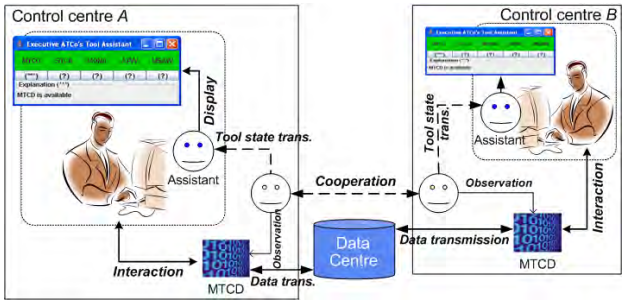

Figure 4. All machines run smoothly and are fully connected in a handover situation.

The controller in center *B* (called *CB*) then makes a flight data change request (*e.g.* a demand for exit flight level change for an outgoing aircraft). However, due to some accident, control center *B* has been disconnected from the flight data-processing center. Due to the disconnection, this request is not sent to the data center.

Now, CB's assistant agent detects that a data change request was issued by *CB*. It notifies the data sentinel agent of MTCD in *B* of this request. This agent in its turn informs the monitoring and assistant agents in control center *A* through their simulated WAN connection.

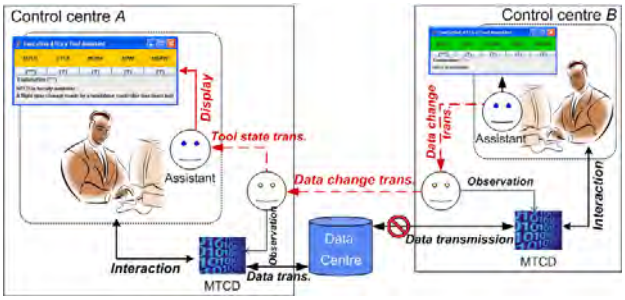

Figure 5. Control center *B* is disconnected from the flight data-processing center (1st phase).

The data sentinel agent of MTCD in *A* discovers that no such flight data change was received from the data-processing center. This also means that the flight data concerning an aircraft which is controlled by center *B* are no longer accessible from *A* and therefore unusable for conflict detection.

In consequence, the assistant agent of the controller in center *A* displays a yellow flag, informing his controller that the software tool is only available locally, *i.e.* it only gives correct results for aircraft under control of center *A*.

Knowing this, the data sentinel agent of MTCD in control centre *A* signals back to the monitoring agents in *B* that there was on its side a flight data change request which was not taken into account. This agent notifies the *CB*'s assistant agent of this incident.

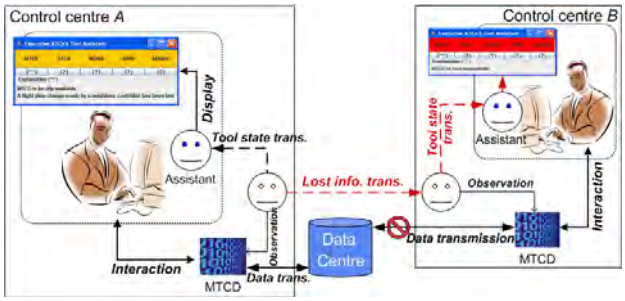

Figure 6. Control center *B* is disconnected from the flight data-processing center (2nd phase).

Finally, the *CB*'s assistant agent then displays a red flag, informing his controller that the software tool is now unavailable. This is what we intended as explained in 1.3.

7 Related Work

Researchers often take into account human factors in critical socio-technical systems either by specifying users' working procedures or by applying system design methods that help to prevent human errors. Little work has dealt with the daily relation between human operators and their powerful equipments, particularly in situations where technical incidents happen. On the other hand, fault-tolerant methods applied to this kind of system have mainly solved purely technical reliability problem. Then they could not build total confidence for human operators while using automated tools.

Concerning the use of so-called "sentinels" in fault-tolerant component-based systems, as well as in certain MAS, the work of Klein, Dellarocas and colleagues [2003] is also related to the monitoring of a complex critical system. However, they do not use simple communicating sentinel agents but complicated "sentinel components" to detect and deal with exceptions occurring inside application components. These "big" sentinels hence have their own reliability problem. Besides, Hägg [1996] employs BDI sentinel agents to detect and recover errors in negotiation processes between other BDI agents. Nevertheless, these application agents have to be sufficiently "small" that the sentinel agents can fully inspect their code. This condition does not hold in a system having complicated equipments like ATC.

8 Conclusion and Future Work

This paper describes the way in which a MAS can help in mitigating the effects of software malfunction in a complex critical system and building confidence for its users, *i.e.* air traffic controllers. Because of safety restrictions, experiments on real traffic control are not allowed. Therefore, we have developed an ABS, by using eDEP, an ATC simulation platform, and DimaX, a multi-agent platform, following the FIPA specifications [FIPA 2001].

We ran several typical applicative scenarios that showed the reaction of our MAS to the instant unavailability of a software tool due to a network failure. The next step will be to perform human-in-the-loop experiments with controllers in order to validate the conformity of the information provided to them with what they require in situations where some software tools are not available.

However, the agents themselves, like any supplementary layer added to a system, bring their own liability to fault. A natural extension of the present work will be to set up mechanism for ensuring a degree of fault-tolerance at the agent level, which would be of a computational, domain independent nature. The possible techniques would include adaptive replication and exception handling [Marin et al. 2003].

References

[Cristian et al. 1996] F. Cristian, B. Dancey, and J. Dehn. *Fault-tolerance in air traffic control systems*. In ACM Transactions Computer Systems, 14(3):265-286.

[DimaX 2007] DimaX project team. *http://www-poleia.lip6.fr/~guessoum/DimaX/index.html*.

[eDEP 2007] eDEP project team. *http://www. eurocontrol.fr/projects/edep/*. September 2007.

[ESP 2007] European Safety Programme. *Draft of the Guidance Material for Contingency Planning*. Technical report, EUROCONTROL.

[FIPA 2001] FIPA. *FIPA Agent Software Integration Specification*.

[Guessoum et al. 2006] Z. Guessoum, N. Faci and J.-P. Briot. *Adaptive Replication of Large-Scale Multi-Agent Systems - Towards a Fault-Tolerant Multi-Agent Platform*. In Software Engineering for Multi-Agent Systems IV, LNCS, 3914: 238-253.

[Hägg 1996] S. Hägg. *A Sentinel Approach to Fault Handling in Multi-Agent Systems*. In Distributed AI, LNCS, 1286:181–195.

[Klein et al. 2003] M. Klein, J. A. Rodriguez-Aguilar and C. Dellarocas. *Using domain-independent exception handling services to enable robust open multiagent systems: The case of agent death*. In Autonomous Agents and Multi-Agent Systems, 7(1-2):179–189.

[Marin et al. 2003] O. Marin, M. Bertier and P. Sens. *DARX - A Framework for the Fault-Tolerant Support of Agent Software*. ISSRE'03, Denver.

[Petricel and Costelloe 2007] B. Petricel and C. Costelloe. *First ATC Support Tools Implementation (FASTI) Operational Concept*. Technical report, EUROCONTROL.