Towards a Reliable Air Traffic Control[1]

Minh Nguyen-Duc, Zahia Guessoum
Computer Science Laboratory
Paris 6 University
104, av. Président Kennedy
75016 Paris, France
{minh.nguyen-duc, zahia.guessoum}@lip6.fr

Olivier Marin, Jean-François Perrot, Jean-Pierre Briot
Computer Science Laboratory
Paris 6 University
104, av. du Président Kennedy
75016 Paris, France
{olivier.marin, jean-françois.perrot, jean-pierre.briot}@lip6.fr

Vu Duong
Eurocontrol Experimental Centre
Centre du Bois des Bordes B.P. 15
F-91222 Brétigny-sur-Orge
France
vu.duong@eurocontrol.int

ABSTRACT
Since critical socio-technical systems include people interacting with equipments in workplaces, their intrinsic reliability problems have been concerned with both these two "actors". *Air Traffic Control* (ATC) is going to be such a 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. Indeed, a partial failure of a tool in use can have tragic consequences. This paper presents a multi-agent approach to this 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. We build and test our system in an ATC simulation environment, thus develop an *Agent-Based Simulation* (ABS). Experimental work has demonstrated the significance of our system to air traffic controllers.

Keywords
Agent-based decision-aided system, Reliability, Socio-technical system, *Air Traffic Control*.

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 (keep an aircraft in a minimum distance from another aircraft), to ensure safe orderly and expeditious flow of traffic, and to give information to pilots, such as weather and navigation information.

1.1 A critical socio-technical system
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. This will be made possible by the use of software tools to support air traffic controllers (see the *First ATC Support Tools Implementation* (FASTI) program [18]).

Our work is concerned with the next generation of software systems for ATC. These systems will process some advanced flight data [7], which will be much more complicated than the currently used data. This will increase the controllers' capacity at the expense of a complexification of their task, but also will raise the technical issue of reliability.

On the "human" side, moreover, the integration of sophisticated tools in the controllers' daily work currently faces difficulties, like in any critical socio-technical system [9][16][20][21][23][28]. 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 functionalities of their tools. And this 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. In this paper, we report on an experiment with a *Multi-Agent System* (MAS) which aims at building confidence for air traffic controllers.

1.2 A multi-agent approach
We argue that the best way to prove a support system's reliability 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* [6], 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 3).

1.3 Outline of the paper
The paper is organized as follows. Section 2 presents the ATC system and analyzes a typical example of technical incident.

[1]This is an improved version of a paper with the same title which was accepted as a short paper at AAMAS'08 Industrial Track.

Section 3 proposes our MAS solution. Section 4 describes the development of our ABS for experiments. Then in Section 5, an experimental scenario clearly shows how our agents react to a typical network failure. Section 6 summaries feedback from ATC experts on our MAS, which has been gathered in several demonstration sessions. Section 7 discusses related work. Finally, Section 8 draws a conclusion.

2. TYPICAL EXAMPLE

2.1 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. The system will be distributed over local area networks (LANs) in each control center and the wide area network (WAN) between centers. 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. Each tool is thus at the same time exchanging data with the common flight data-processing centre and interacting with the controller user interfaces of the different controllers in the same control centre.

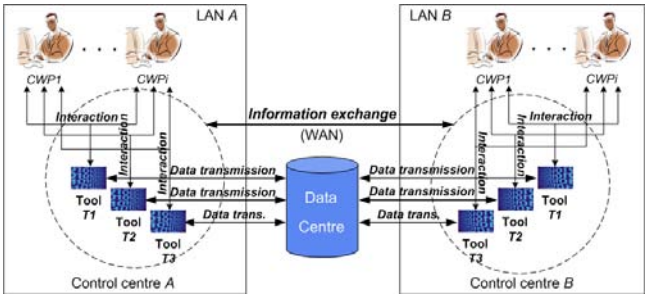

Figure 1. Basic future ATC system architecture.

A typical example of support tool is the *Medium-Term Conflict Detection* (MTCD) [18]. 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).

2.2 Scenario

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 a situation.

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 (and therefore from the responsibility of one control center to the other center).

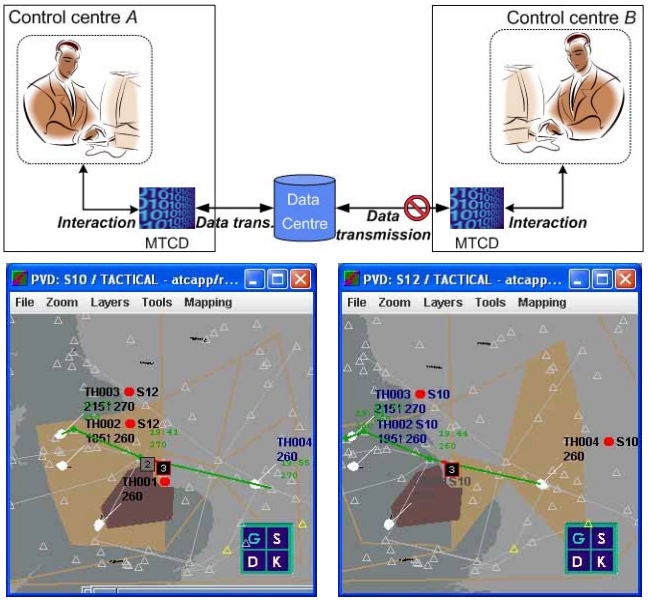

Figure 2. Typical example of a handover situation: the two controller's screens on both sides of the border.

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 aircraft real 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 Co_1. 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 [1] 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.

3. OUR MULTI-AGENT SYSTEM

3.1 Objectives

To fulfill the need presented in the previous section, a decision-aided system for 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.

More importantly, 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 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.

3.2 Agent design

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 agents to perform these two common tasks. We currently use three monitoring agents for each tool, *i.e.* a data sentinel, a middleware 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; for example, as shown in the experimental scenario below, data sentinels ubiquitously check exchanged data (the absence of needed data at some network node often results from data losses).

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; for instance, a computation sentinel checks timeout errors of a computation module of a tool.

3 *Middleware sentinel agent*: receives from the middleware the notifications of faults related to a specific software tool (this also means that the monitoring agents do not employ any sophisticated fault detection technique [10][12][24]).

4 *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, all the agents presented above 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 simplicity of the agents

would bring not only more reactivity, but also more robustness to the MAS.

3.3 Distribution of our MAS

Based on the basic ATC system architecture presented in Section 2, we install in a control center a group of coordinated agents that are distributed over the whole corresponding LAN. The agents on the various LANs also communicate over the inter-center WAN, making up a global MAS.

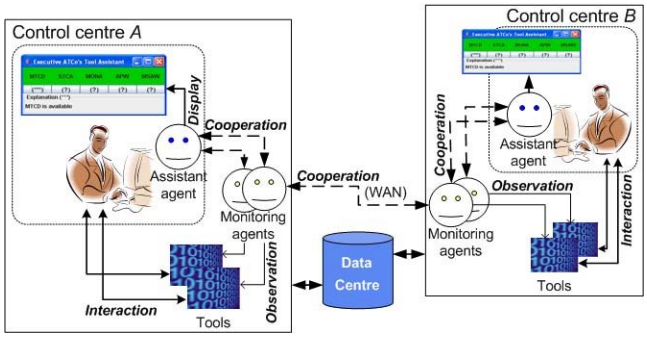

Figure 3. Monitoring and assistant agents.

Each local group of agents is composed of assistant agents and of monitoring agents. We associate an assistant agent with each *Controller Working Position* (CWP). 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. At the level of implementation, only special agents playing the role of interface between our MAS and the whole tool system (see 4.3) need to be hosted on the same machines as the tools to which they connect.

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 exchanging observed tools' data and controllers' actions with each other, the agents can inform the controllers about the possible actions to take after the incident. For instance, in the scenario presented in Section 5, 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.

4. SIMULATING OUR SYSTEM

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 hence build and test our MAS into a simulation environment, thus develop an *Agent-Based Simulation* (ABS). We employ the eDEP platform (*Early Demonstration & Evaluation Platform*) [5], which offers not only realistic air traffic data but also a distributed simulated ATC environment. We also use the DimaX platform [2], which helps with developing reliable MASs.

4.1 ATC simulation platform - eDEP

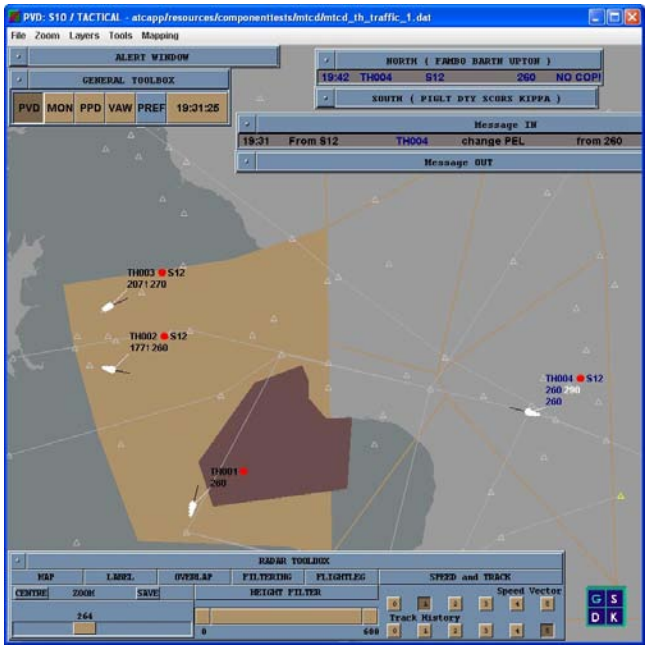

Figure 4. CWP for the executive controller of the sector *S10* (*S10* is the sector in light brown; white radar tracks mark aircraft; and red dots indicate aircraft potentially in conflict).

eDEP [4] 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.*:

1 *Airspace*: a database of static airspace information;

2 *Integrated air surveillance*: a database of surveillance radar tracks;

3 *Flight Path Monitor*: uses track data produced by the IAS to monitor the progress of a flight according to its planned path through the airspace;

4 *Initial flight plan*: an initial plan that defines route constraint points and altitude limits;

5 *Trajectory predictor*: a trajectory prediction algorithm which uses aircrafts' kinematic models to predict the real motion of a particular aircraft;

6 *Controller Working Position* (CWP): the main graphical interface to the system based on a plane view display of the control sector (see Figure 4).

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.

4.2 Multi-agent platform – DimaX

DimaX [2] is a Java multi-agent platform which provides 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.

In addition, DimaX provides a *Naming Service* which localizes agents at the time of message sending. A name server maintains the list (*i.e.* white pages) of all the agents within its administration domain. When an agent requests interacting with the others, it does not need to know their physical locations. Given the agent identifiers, the name server returns the corresponding agents physical addresses.

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 [17]), which brings in services, *e.g. Fault Detection Service* and *Replication Service*, which provides transparent support for making MAS fault-tolerant through adaptive replication.

4.3 Implementing 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* [8]. 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".

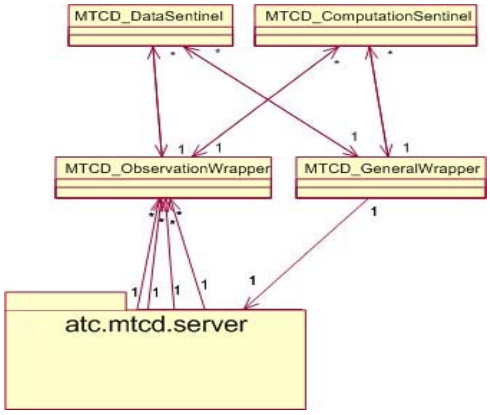

Figure 5. Monitoring and wrapper agents for MTCD (partial UML diagram).

Based on the agent model discussed in 3.2, as a first step we install three monitoring agents and two wrapper agents for each of software tools:

1 `XXX_DataSentinel`, `XXX_ComputationSentinel`, `XXX_MiddlewareSentinel`: observes the XXX[2] component's input/output data, communicates with other agents and collects notifications from the middleware in order to discover faults;

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

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 the status of the various tools that run on the LAN. Additionally, it displays a few lines of explanation about the tool that is marked with (***).

Figure 6. `CWP_Assistant`'s user interface.

The `CWP_Assistant` observes the controller's actions through observation services provided by a `CWP_InterfaceWrapper`, which monitors the CWP's GUI in the same way as the monitoring agents access to the software tools through their observation wrappers.

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

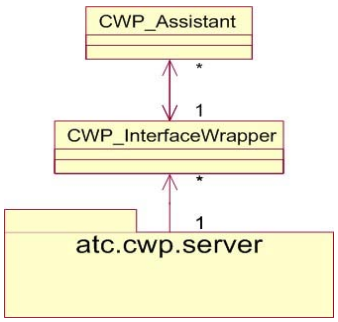

Figure 7. `CWP_Assistant` and wrapper agents (partial UML diagram).

5. AN EXPERIMENTAL SCENARIO

5.1 Objective

The first tests of our agents on the ABS use several experimental scenarios one of which corresponds to the example presented in Section 2. In this section, we will describe in detail this scenario which will show how the assistant and monitoring agents behave in situations where a typical fault occurs within the support system. This experiment also aims at demonstrating the usefulness of our MAS for air traffic controllers.

Precisely speaking, this scenario illustrates the reaction of our MAS to the possible unavailability of a tool due to a failure of the connection between a control centre and the common flight data-processing centre.

5.2 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.

5.3 Event sequence

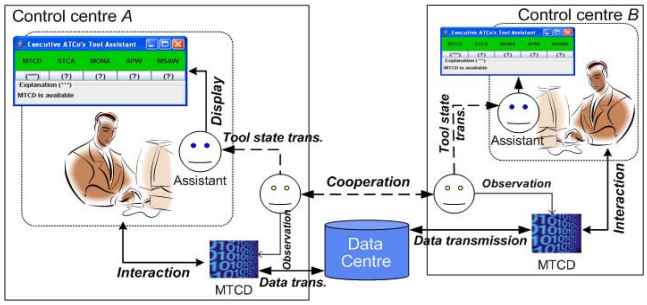

Figure 8. All machines run smoothly and are fully connected in a handover situation.

We are in a handover situation (as described in 2.2): there are aircraft flying from control center *B* to control center *A*. At first, all machines run smoothly and are fully connected. 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 (see Figure 8).

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.

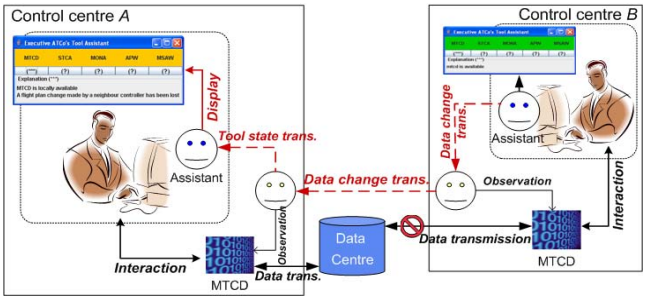

Figure 9. Control center *B* is disconnected from the flight data-processing center (1st phase).

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.

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.

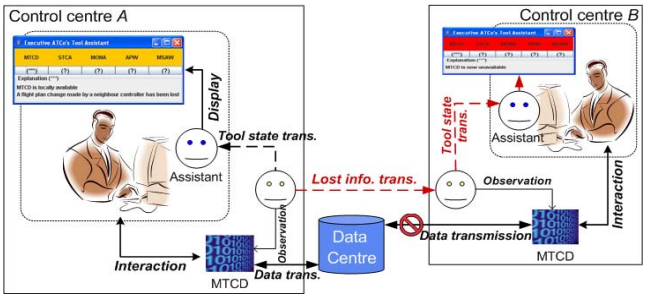

Figure 10. Control center B is disconnected from the flight data-processing center (2nd phase).

In consequence, the assistant agent of the controller in center *A* displays a yellow flag, informing his controller that MTCD is

only available locally, *i.e*. it only gives correct results for aircraft under control of center *A* (see Figure 9).

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.

Finally, the *CB*'s assistant agent then displays a red flag, informing his controller that MTCD is now unavailable (see Figure 10).

6. VALIDATION

6.1 Technical incidents

We have used our ABS to demonstrate typical scenarios to experts including both researchers in the field of ATC and professional air traffic controllers. These scenarios showed the reaction of our system to various technical incidents, each of which belongs to one of the following categories:

1 *Network failure*: the instant unavailability of tools due to a failure of the WAN connection between centers (like in the scenario presented above);

2 *Timeout error*: the instant unavailability of tools due to an unexpected timeout error of a tool's elementary computation module;

3 *Machine failure*: the instant unavailability of tools due to successive failures of tool servers.

6.2 Feedback

The first series of demonstration sessions was addressed to researchers from Eurocontrol (*European Organisation for the Safety of Air Navigation*). They questioned the kind of information exchanged by agents. They particularly sought to understand the advantages of this exchange in comparison to a simple duplication of lost data. Furthermore, they stressed the need of the involvement of operational points of view in the validation process.

The second series of experiments was therefore concerned with professional controllers. They firstly concentrated on the `CWP_Assistant`'s interface and recommended many modifications for it. Although they admitted that adequate information of tools' unavailability would be important for them (if there was any), they still found it difficult to accept eventual technical incidents, to think of incidents and to discuss solutions.

In order to merge the two different visions for our MAS solution to the reliability problem in ATC, we mixed researchers and operational operators (*i.e*. controllers) together in the third series of experiments. With the help of the researchers, the controllers concentrated on the information they require in each situation and gave valuable recommendations for each of scenarios. For example, concerning the one we have presented in Section 5, they suggested that although this scenario was only related to MTCD, in such situation of network failure, all the other tools (*i.e*. STCA, MONA, APW, MSAW) would find themselves in the same state as MTCD.

7. RELATED WORK

Researchers often take into account human factors in critical socio-technical systems [9][21] either by specifying users' working procedures [16][20] or by applying system design methods that help to prevent human errors [23][28]. 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 [3][13][14] 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 [11] and Shah et al. [25][26][27] employ sentinel agents to detect and recover errors in negotiation processes between 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.

Regarding other agent-based systems used in ATC, we have been motivated by the work of Ljungberg, Rao and Georgeff [15][19] on a decision-support tool that helps air traffic controllers at an airport to predict optimal landing flows. It is a good example of *Distributed Artificial Intelligence* built into a single program. However, we argue that in order to develop such assistant tools in the future distributed ATC system (see Section 2) it will be necessary to distribute them over LAN/WAN networks, and to make them interact with other automated tools. In this perspective, they will not assist an individual controller but a group of cooperating controllers. Moreover, one could recognize that the technical framework presented in this paper, which supports the integration of a MAS in a distributed ATC simulation environment, would not be limited to fault-tolerant objectives. It could be reused to build and test agent-based systems that, for example, make predictions on the evolution of the traffic.

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 [8].

This simulation has been used to demonstrate the usefulness of our MAS for the future ATC system to air traffic controllers. Indeed, we run typical applicative scenarios that show the reaction of our MAS to the instant unavailability of software

tools due to incident techniques. Some experiments with professional controllers have also been performed 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 [17] and exception handling [4].

9. REFERENCES

[1] Cristian, F., Dancey, B. and Dehn J. 1996. Fault-tolerance in air traffic control systems. In ACM Transactions Computer Systems, vol. 14, n°. 3, pp. 265-286.

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

[3] Dellarocas, C. 1998. Toward Exception Handling Infrastructures in Component-Based Software. International Workshop on Component-based Software Engineering.

[4] Dony, C., Knudsen, J. L., Romanovsky, A.B. and Tripathi, A. 2006. Advances Topics in Exception Handling Techniques. In Lecture Notes in Computer Science, vol. 4119.

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

[6] European Safety Programme. 2007 Draft of the Guidance Material for Contingency Planning. Technical Report, EUROCONTROL.

[7] EUROCAE project team. 2006 Flight Object Interoperability Proposed Standard (FOIPS) Study. Technical Report, EUROCONTROL.

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

[9] Gregoriades, A., Sutcliffe, A. G. and Shin, J. E. 2002. Assessing the Reliability of Socio-Technical Systems. Proceedings of the 12th Annual Symposium of the International Council on Systems Engineering (INCOSE 2002).

[10] Gustafsson, F. 2007. Statistical Signal Processing Approaches to Fault Detection. In Annual Reviews in Control (JARAP), vol. 31, n°. 1, pp. 41-54.

[11] Hägg, S. 1996. A Sentinel Approach to Fault Handling in Multi-Agent Systems. In Distributed AI, Lecture Notes in Computer Science, vol. 1286, pp. 181–195.

[12] Isermann, R., 2006. Fault-Diagnosis Systems - An Introduction from Fault Detection to Fault Tolerance. Springer-Verlag Berlin Herdelberg.

[13] Klein, M. and Dellarocas, C. 1999. Exception Handling in Agent Systems. Proceedings of the Third International Conference on Autonomous Agents, Seattle, pp. 62–68.

[14] Klein, M., Rodriguez-Aguilar, J. A. and Dellarocas, C. 2003. Using Domain-Independent Exception Handling Services to Enable Robust Open Multiagent Systems: The Case of Agent Death. In Autonomous Agents and Multi-Agent Systems, vol. 7, n°. 1-2, pp. 179–189.

[15] Ljungberg, M. 1992. The Oasis Air Traffic Management System. Technical Note 28, Australian Artificial Intelligence Institute, Carlton, Australia.

[16] Mearns, K., Whitaker, S., Flin, R., Gordon, R. and O'Connor, P. 2000. Factoring the Human into Safety: Translating Research into Practice. In Benchmarking Human and Organisational Factors in Offshore Safety, OTO 2000 061, Sudbury: HSE Books, vol. 1.

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

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

[19] Rao, A. and Georgeff, M. 1995. BDI Agents From Theory to Practice. Technical Note 56, AAII.

[20] Reiman, T. and Oedewald, P. 2006. Organizational Culture and Social Construction of Safety in Industrial Organizations. In Svenson, O., Salo, I., Skjerve, A. B., Reiman, T. and Oedewald, P., eds., Nordic Perspectives on Safety Management in High Reliability Organizations: Theory and Applications.

[21] Rouhiainen, V. 2006. Safety and Reliability. Technology Theme – Final Report.

[22] Saha, G. K. 2006. A Single- Version Scheme of Fault Tolerant Computing. In Journal of Computer Science & Technology, vol. 6, n°. 1.

[23] Scacchi, W. 2004. Socio-Technical Design. In Bainbridge, W. S., ed., The Encyclopedia of Human-Computer Interaction. Berkshire Publishing Group.

[24] Scheina, J., Bushbya, S. T., Castroa, N. S. and House, J. M. 2007. A Rule-Based Fault Detection Method for Air Handling Units. In Energy and Buildings, vol. 38, n°. 12, pp. 1485-1492.

[25] Shah, N., Chao, K-M., Godwin, N. and James, A.E. 2005. Exception Diagnosis in Open Multi-agent Systems. Intelligent Agent Technology, IEEE Computer Society, pp. 483–486.

[26] Shah, N. Chao, K-M., Godwin, N., James, A.E. and Tasi, C-F. 2006. An Empirical Evaluation of a Sentinel-Based Approach to Exception Diagnosis in Multi-Agent Systems. Advanced Information Networking and Applications, IEEE Computer Society, vol. 1, pp. 379–386.

[27] Shah, N., Chao, K-M., Godwin, N., Younas, M. and Laing, C. 2004. Exception Diagnosis in Agent-Based Grid Computing. International Conference on Systems, Man and Cybernetics, IEEE, pp.3213–3219.

[28] Wilpert, B. 2005. Psychology and Design Processes. In European Psychologist, vol. 10, n°. 3, pp. 229-236.