Airports for Agents:
An Open MAS Infrastructure for Mobile Agents

Jan Tožička

Gerstner Laboratory, Agent Technology Group, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague,
Technická 2, 166 27, Prague 6, Czech Republic
jan.tozicka@matfyz.cz

Abstract. *Airports for Agents*[1] (AA) is an implemented distributed multi-agent infrastructure designed for dynamic and unstable Internet environment. The infrastructure consists of platforms called *Airports* that enable agents to communicate together and to be persistent. Furthermore, the Airports allow agents to migrate trough the system and to use local resources. Any Airport can host any agent from the network, therefore we considered high requirements for the security. Network of Airports can dynamically change in the time as new Airports connect to the system, or disconnect. We designed distributed stochastic algorithm keeping the system connected, because AA has no central element. The agent migration brings a communication problem known in the field of distributed systems: where to find the agent I have been communicated with, previously, while he changed his location (platfrom)? We present *Kept Connection* as a transparent solution of this problem. System is designed to be distributed over large amount of computers.

1   Introduction

Using distributed MAS with mobility can reduce network traffic and the waste of the time caused by the network slack. A stable, widely accessible and extensible infrastructure seems to be necessary to get maximum of the potentials of multi-agent systems.

We will use the following terminology. A *MAS platform* is an empty system without any agent. A *MAS* can be either a platform with implemented agents, or only several agents communicating together (in this case the MAS platform is the operating system). *Distributed MAS infrastructure* consist of several MAS platforms running on several host computers. If agents can migrate form one host platform to another in the distributed MAS infrastructure we say, that it

---

[1] Project team was led by Mgr. Roman Neruda, CSc., and included also these members: Jaroslav Kameník, Tomáš Kasl, Eva Poučková, Pavel Socha and Roman Vaculín. Project was developed at Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic. The implementation in Java language has been demonstrated with several demo-agents. More information can be found at web site http://letiste.zde.cz.

is an infrastructure supporting the agent *mobility*. An platform supporting the mobility is *open* if unknown agents can come over the network. All components of open platform defending the host computer and other agents against misbehavior and cheating of hostile agents compile platform *security*. Distributed MAS with the elements, which can shutdown and start anytime or are connected to the network only temporarily, is called *dynamic*.

In this paper, we present open dynamic distributed MAS infrastructure supporting the agent mobility called *Airports for Agents*.

The rest of the paper is organized as follows: in section 2 we define main features of the distributed MAS; in section 3 we discuss our implementation of the *Airports for Agent* infrastructure; in section 4 we compare AA infrastructure with the most known MAS; in section 5 we present several main points of our further work and finally we conclude in section 6.

2   Distributed Multi-agent System

We consider these requirements for MAS:

- **Agent communication.** Two potentially the most used types of agent communication are message-based communication and communication using the function calling.
- **Agent persistency.** Agent does not have to carry about his data and his existence as whole.

Additionally, we consider these requirements for the distributed MAS, that has the potential to be widely used:

- **Agent mobility.** Agents can easily move trough the system with their data.
- **Openness.** Unknown agents can come to the host connected to the system.
- **Security.** In open MAS platform, security against agents misbehavior or cheating is very important. Generally, there are four types of security [6, 7]:
  - security for the host against the agents,
  - security for other agents,
  - security for the agents against the host, and
  - security for the underlying network.
  Researchers have found serious solution only for first two types. Our security model also focuses on these two issues. The enlargement of current security model to include remaining issues is one of the points of our future work.
- **Dynamics.** The set of elements of distributed MAS changes in the time.
- **Complete distribution.** Distributed system has no central element. This requirement is necessary when we design system distributed over unstable network, e.g. Internet, and consider its dynamics.
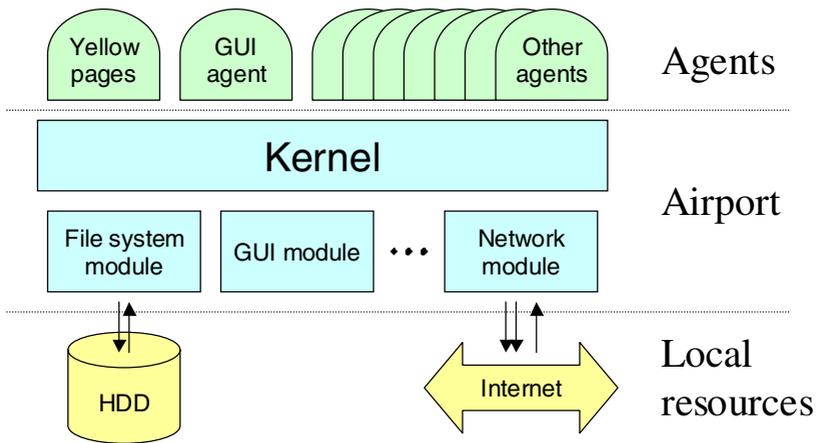- **Accessibility.** All on-line platforms are findable for all agents in the system.

Fig. 1. High-level Airport architecture. Two main components of an airport are kernel and modules. Kernel provides necessary services for the agents, and facilitates them access to modules to access additional services

3   Airports for Agents

AA platform provides all the infrastructure allowing the MAS designers to concentrate on agent's functional core implementation.

AA infrastructure consists of the elements called Airports. Every Airport is denoted by an unique identifier. Airports are distributed and connected over a network. The topology of the Airports' connection is dynamic. Airports can connect to the network or disconnect anytime. Dynamics of AA topology is described in subsection 3.2.

The *Airports for Agents* infrastructure was implemented in Java language.

Let us describe main features of AA infrastructure in the following subsections.

3.1   System Architecture

The high-level architecture of an Airport is illustrated in figure 1. Airport composes of kernel and several modules. Agents can be understood as the top level of the architecture.

Kernel. Airport's kernel provides necessary services for the agents' life-cycle:

 – agents creation,
 – agents migration between Airports (see 3.4),
 – inter-agent message-based communication (see 3.6),
 – agent persistency, etc.

Kernel also keeps the Airport connected in the *AA network* (described in 3.2).

Kernel notifies the agents about important events via bit-based signals, e.g. system shutdown, arrive of new message, etc.

Modules. Kernel functionality is extended by plug-in modules. They are used for accessing local resources, such as filesystem, network communication, user interface, etc. These resources either do not have to be accessible in every Airport or can be accessible in different way. For example, consider a disk less station. It is necessary to fulfil agents necessity to backup data in another way, e.g. storing them on a remote disk. This is simply made by changing the *FileSystem* module. And agent is shielding from the knowledge of these unsubstantial details.

Agents. Agent's access to kernel and module functionality is controlled by access policy.

From the Airport view, all the agents are treated the same way based on individual access rights. There are no privileged system-agents. The only one exception is *Yellow Pages Agent* (described in 3.7), which is known also under his alias name in the Airport.

A module can offer agents several interfaces. One of them is often used for managing the resource and the access policy. Only agents responsible for the resource can access this interface. In our system, we call them *System agents*, although, from the Airport view, they are only ordinary agents. E.g. The *FileSystem* module offers two interfaces. First one is used for agent's direct access to the files on the HDD. This access is controlled by agent's access rules. These rules can be changed only using the second interface. It is accessible only by the *FileSystem System Agent*.

3.2 AA Dynamics

AA infrastructure is designed to be widely used, also by users, which has dial-up or another type of *not-still-online* internet connection.

Every Airport platform connected to the Internet should be findable, to be useable by other agents. Because we assume not to operate any central element, we decided to create a network connecting all Airports. This network is called *AA network*. So, agent can find the Airport with desired services, if it exists and is on-line. We prefer this approach to store the list of all other Airports because the *AA network* consists only of the Airports currently on-line. The communication traffic is smaller when a new Airport is connecting to the *AA network* and when an agent is looking for an accessible Airport. Of course, some traffic is necessary for *AA network* updating.

Note that, *AA network* is only logical network and its topology has no relation to the physical network connecting the host computers.

When a newly installed Airport wants to connect to the *AA network*, it needs to know at least one address of any Airport already connected. It is used for searching other Airports and connect into the *AA network*. It is important

to create a network which will satisfy requirements for attainability of all other Airports even if several Airports will shutdown. Therefore, we specify these two main requirements on created network: maximize the number of Airports which can shutdown at one moment and minimize the network traffic during creating and updating the network.

This problem can be converted to the graph theory problem of creating and maintaining connected directed graph. In the graph, the nodes represent Airport platforms and the set of edges is defined by the relation 'is connected with'. In the best case, the created graph is k-connected, where k is number of Airports which is every Airport connected with. It ensures that the graph remains connected even as any $(k-1)$ Airports will shutdown at one moment. There has not been found an algorithm for creating and maintaining such a graph in distributed (*AA network* has no central element) and dynamic (Airports are continually connecting and disconnecting) environment. Therefore, we designed distributed stochastic algorithm keeping the *AA network* connected.

Our algorithm keeps with the list of neighborhoods also information about all their neighborhoods. When an adjacent Airport disconnects, some of his neighborhoods are chosen for his replacement. From time to time, the list of adjacent Airports is consolidated not to contain needless platforms (e.g. that ones, which are neighborhoods of other adjacent Airport).

3.3   Security Features

In AA infrastructure, strong security for the host [6, 7] is implemented.

The security model is based on the information fortress model [3] where the host is protected by maintaining a closed system accessed through well defined and controlled interfaces. In *AA platform*, all local resources are maintained by modules. The modules know the *access rules* for the restrictions of agent access to the resources. User can define specific rules for every known agent (agent, whose classes has been already installed) and default rules for other agents.

When newly coming agent needs to access other services, he has three main possibilities how to achieve it. Firstly, he can try to negotiate direct access with the system agent administering this service. Further, he can request another agent for facilitation of undirect access. Or finally, he can request the user to permit the access to desired services; but it is possible only if agent has access to the user interface.

Protection of other agents uses basic Java feature, that programs have no direct access to the memory. Therefore agent can not affect other agent even though there are implemented as threads of one application.

As a protection of CPU overloading by hostile agent runs in a single thread created by the kernel during agent creation. Agents can not create other threads.

3.4   Agent's Mobility

Mobility is the agent's ability to autonomously migrate from one MAS platform to another and to use their distributed resources locally. This approach reduces

network traffic and agent's response can be faster then usage of far access to the data.

Mobility is almost necessary in dynamic environment. Consider, for example, an agent which needs to be still on-line. In our platform, an agent receives a system signal when the system is going to shutdown. Therefore, the agent can save his work and safely move to another Airport.

AA infrastructure offers agents an easy way to be transported.

The process of agent's migration consists of two phases. In the first phase, the agent asks his current Airport for moving him to the target Airport. Current Airport ask target Airport whether it will accept this agent, or not. If the target Airport accepts him, the transport of agent's data begins. It is necessary to transport agent's *Kept Connections* (see 3.6), private data, sometimes also agent's classes, etc. During this process, agent is still running and finishes his work, e.g. processes unread messages in mailbox, stores uncompleted work. If something happens agent can still decide to not move. Otherwise, agent confirms his decision to move. After his ending, his private files and identity are moved.

AA infrastructure warrants that agent will be at exactly one Airport after the moving. Even if communication error occurs at any time, or any Airport will shutdown.

There is not implemented any security for the agent against the host because agent decides where he want to move and moreover, only migration within AA infrastructure is currently implemented.

3.5   Agent Life-Cycle

An Agent's life begins by his creation. New unique (in the whole system) ID is generated and assigned to the agent during the creation. We use proprietary ID semantics different then the FIPA name. The ID does not change during whole agent's life.

The agent can freely migrate trough the system or fall asleep waiting any system signal. Weak persistency is warranted during these actions.

When agent finishes his work and decides to die, his identity and all his private files are removed from the system.

3.6   Agent Communication

Only message based communication between the agents is implemented in AA infrastructure. It is implemented in two ways.

Firstly, it is common agent communication protocol compliant with FIPA ACL standard [5]. Agent can simply send a message specifying receiver's unique ID and receiver's current Airport (only ID is necessary if the sender and the receiver are on the same Airport).

Second type of communication solves a problem known from distributed systems with migration of components. Problem is how to find the agent I have

been communicated with, a while ago, if he is not at his previous position (he moved away)? One possible solution is the *tracking*. The *signs* with new location are put when any agent leaves local platform. Following these *sings*, starting in the last known position, allows to find the desired agent. This solution does not work in dynamic environment, because any host in the agent's track can be currently off-line. Therefore we designed and implemented another solution of this problem called *Kept Connection*. It enables two agents to create a communication channel between them. This connection is kept between them even as they migrate trough the *AA network*.

Kept Connection allows agent also to address his partner only by his ID. No additional information about his (current or previous) location is necessary. It also ensures that the message will be delivered if the receiver is accessible.

In the Airport kernel, there is an *address book* associated with every currently presented agent. It contains all agents, which agent has *Kept Connection* established with.

Agent advices to the platform, that he would like to establish new *Kept Connection* with an agent, by adding his current address to the address book. The connection does not become active unless the second agent also adds the address of the first one to his address book. Once the connection is created, the platform itself updates addresses of agents in the address book.

When the agent wants to move to other platform, all address books of agents in his address book are notified about it. They tag appropriate connection to be temporarily unavailable. Messages sent through this connection are blocked until the migration is over or the specified time is out. End of moving is notified from old platform, if the moving has not succeed, or from the new platform if the moving has succeed.

Additional network communication load during moving can be a disadvantage of this solution mainly if it is used within a group of agent which moves a lot (e.g. group of search agent looking for a resource). In this case, it is better to use an immobile agent, which agents can communicate with using common messages, because *Kept Connection* is not obligated.

Any agent can cancel his *Kept Connection* with other agent. This way, he can estop the harassment and tracking (see [7]) by the other agent.

Both types of communication ensure that the agent named as message sender has really sent this message, because agent cannot change this information. Therefore, agent cannot send a message and make target agent believe, that the message has been sent by another agent.

Messages are stored in the agent message-box queue and the agent is responsible for managing this queue, e.g. in the cases of moving or shutting down. The message-box allows to rigidly separate the kernel and the agent's body. It seems to be important because of the security.

3.7   Facilitators

In [5], there are presented two facilitator agents, which are also often used in MAS: Directory Facilitator (DF) and Agent Management System (AMS).

In our system, DF is presented by *Yellow Pages Agent* which should be at every Airport and allows each agent to register to his database of locally running agents. Every agent can also make public some of his services which he want to offer other agents. *Yellow Pages Agent* has always the alias 'YellowPages' associated with him. Therefore, newly created agent can send him a message without any other a priori knowledge about the system, e.g. ID of any agent. He simply fill in the 'YellowPages' string as the receiver ID.

In Airports terminology, AMS is *White Pages Agent*. He manages list of agents which could be possibly created on local Airport. In Java terminology, he knows which agent classes are installed and can be used for agent creation.

We recognize another type of facilitator agent. *Black Pages Agent* manages a list of agents, which has been present on the local Airport and moved away, with information where they have moved to. It enables tracking of agents. But only in the case that every Airport in the track is currently on-line. Better and transparent solution to this problem is offered by the *Kept Connection* (see 3.6).

All of these facilitators are ordinary agents with a small exception of *Yellow Pages Agent*, which has its alias in the Airport and this alias is the same everywhere. All three services provided by these agents are currently implemented in one agent.

Facilitator agents know only registered agents and types. These registrations are voluntary, therefore agent can make himself invisible to other agents. He can use it as defense against harassment (see [7]).

4   Comparison to the Related Work

In this section we will try to point the main differences between our infrastructure and some other MAS.

4.1   JADE

JADE [8] is often used as undistributed MAS, even if it enables to be distributed. JADE does not offers any possibility how to find another JADE platform. Therefore, if we need to keep the system connected, agent or user has to carry about it. AA keeps the hosts connected in the *AA network* and the user has to provide only one address of on-line Airport after the installation.

Agents in JADE system can define behaviors, which are evoked when appropriate message arrives. AA infrastructure uses simpler and possibly faster way to handle messages. Every agent has his queue, where all his messages are stored. This simple solution can be expanded to implement JADE's behaviors or other more sophisticated solution.

JADE Agents can run their own threads. AA platform does not allow it because of the security against hostile agents, which wants to overload the system (no other solution in Java has been found).

4.2 Aglets

The agents in Aglets platform [2, 10] are programmed using event-driven scheme, as in windows system programming. They implement only handlers which are evoked when appropriate event occurs. It enables to simply create a reactive agent which responds to incoming messages. But the creation of more sophisticated algorithm using some communication protocol can be more difficult. In comparison, in AA platform, every agent has the message-box where all incoming messages are stored. Agent can pick out a message anytime. It enables to simply create agent implementing some communication protocol, although the creation of a simple reactive agent can be more difficult then in Aglets platform.

Aglets platform solves security problems similar to the problems solved in AA platform. Authors of Aglets platform defined several supported local resources and solved the security policy for them [9]. The Airport kernel does not contain any security for the local resources access. It allows agents only to access several main services. The modules are responsible for the access policy for all local resources, e.g. filesystem, network, etc. This solution seems to be more flexible, because everybody can write the module with specific security policy or the module accessing specific resource (possibly currently unknown), and no changes to the Airport kernel has to be made.

5 Future Work

In present time, implementation of the next version of CPlanT project [4] is created using AA infrastructure.

Our further work will be focused to several main direction: cooperation with other MAS, *Enhanced Kept Connection* and enhanced security model.

Cooperation with other FIPA compliant MAS infrastructures (e.g. JADE [8] or Agentcities [1]) is considered at two layers. Firstly, communication with the agents residing there. It is already designed in current architecture, only implementation of some support functionality is necessary. Secondary, migration between both infrastructures. Only logical migration of agents is considered (migration without agent code).

Enhanced Kept Connection, which implements contact transmission (one agent gives the second one the contact to the third one) and other services, could be fully functional replacement of addressing using receiver address.

Enhancing the security to include the protection of agents against the host and the protection of underlying network. Better security against CPU overloading should not be possible without also changing the Java virtual machine.

Also improving of algorithm keeping the *AA network* connected is a permanent challenge for us.

6   Conclusion

Open dynamic distributed MAS infrastructure with high security level has been presented. It supports agent mobility and weak form of persistency.

To the best of our knowledge, *Airports for Agents* infrastructure contains two features which have not been presented ever before in MAS context: *Kept Connection* and *AA network* (ensuring infrastructure connectivity).

The *Kept Connection* has been presented as an arbitrary replacement of agent tracking. It is transparent, simple to use and cannot be abused for harassment or tracking by a hostile agent.

Connectivity of dynamic *AA network* is ensured using distributed stochastic algorithm. It allows agents to find any Airport with desired services if any is currently on-line.

References

[1] Agentcities: http://www.agentcities.org.   381
[2] Aglets: http://www.trl.ibm.com/aglets/.   381
[3] Blakley B. (1996): The Emperor's Old Armor. *Proceedings of New Security Paradigms Workshop.*   377
[4] CPlanT: Multi-Agent System for Planning Humanitarian Relief Operations. http://agents.felk.cvut.cz/cplant.   381
[5] FIPA: Foundation for Intelligent Physical Agents. http://www.fipa.org.   378, 380
[6] Gray R. S. (1996): Agent Tcl: A Flexible and Secure Mobile-Agent System. *Proceedings of Fourth Annual Tcl/Tk Workshop (TCL 96).*   374, 377
[7] Greenberg M. S., Byington J. C., Harper D. G. (1998): Mobile Agents and Security. *IEEE Communicitaions.*   374, 377, 379, 380
[8] JADE: Programmer's Guide. http://sharon.cselt.it/projects/jade/.   380, 381
[9] Karjoth G., Lange D. B., Oshima M. (1998): A Security Model for Aglets. *Mobile Agents and Security*, pp. 188–205   381
[10] Lange D. B., Oshima M. (1997): Java Agent API: Programming and Deploying Aglets with Java, *Addison-Wesley.*   381