A NEW PERFORMATIVE FOR HANDLING LACK OF ANSWERS OF AUTONOMOUS AGENTS

Katia Potiron[1,2], Patrick Taillibert[1] and Amal El Fallah Seghrouchni[2]
[1]*Thales Systèmes Aéroportés, 2 avenue Gay Lussac, 78852 Elancourt, France*
[2]*Laboratoire Informatique de Paris 6, 104 avenue du Président Kennedy, 75016 Paris, France*

Keywords: Agent communication, Fault tolerance, Agent design, Autonomy, Fault handler.

Abstract: An agent can send a message and never receive a response, this is what we name the ”empty mailbox problem” this paper is concerned with. The causes of the problem can lie in low level layer as, for instance, in communication links, but also in the behavior of the autonomous entity the agent is interacting with, which can choose not to respond. The task is not easy, for the agents developer, to find what is to do in such cases. The proposed solution consists in a performative and the associated meta-protocol. This results into a generic method to handle the empty mailbox problem in the case of temporary faults. Some prospects are given to handle permanent faults.

1 INTRODUCTION

The *Empty Mailbox Problem* (EMP), in other words the problem to find what is to be done when an expected information is not received, is not specific to the MAS domain.
In distributed systems the fact that an expected message is not received, may result from: a development fault (a bad implementation of the interaction protocol), a problem with the communication link (loss or deterioration of a message), a hardware failure (a computer crash), a bad specification on the interaction protocol, a malicious user...
But for MAS, which are composed of autonomous agents, another cause can be identified. If an autonomous agent chooses, for any reason, not to follow the interaction protocol used during a conversation with other agent; the other agent waits for a message but its mailbox remains empty.

More rigorously, the EMP is an error that programs communicating by message passing can experiment due to several faults (since faults are defined as the causes of errors). In order to define the faults that cause of the EMP, and hence get clues to find an adapted handler, we first analyze fault classification results, obtained so far.

1.1 The Faults Leading to the EMP

A conventional fault classification (Avizienis et al., 2004), in particular suitable for distributed systems (which MAS are), presents a study of all the faults that can affect a system, fig.1 presents the resulting fault classes tree. A study of the faults leading to the EMP into distributed system reveals that the faults can potentially belong to any of the number 1 to 25 classes. These include development (a mistake made into the code of the agent), interaction (a wrong specification of the interaction protocol) or operational (a network cable unplugged) level faults.

But because of MAS specificities (in particular autonomy and proactiveness), we showed in (Potiron et al., 2007) that an extension to the classification was necessary. We introduced the value *autonomy* into the existing classification to represent the faults occurring during the *autonomous and proactive behavior* of an agent. We named this class of faults: *Behavioral faults*.
These faults are the consequence of the impossibility for agents to predict with certainty the behavior of other agents as well as a consequence of the need for an agent to adapt itself to this unpredictability. This corresponds to six new classes represented in fig.1 by number 26 to 31 on fig.1. A behavioral fault, on an agent-centered point of view, is a consequence of the freedom, ”the power to say no”, that autonomy gives to other agents making them possibly unpredictable. This corresponds to six new classes represented by the fault classes number 32 to 37 on fig.1. Here again the faults corresponding to the EMP into MAS are potentially resulting from any of the number 26 to 37 classes.

Potiron K., Taillibert P. and El Fallah Seghrouchni A. (2009).
A NEW PERFORMATIVE FOR HANDLING LACK OF ANSWERS OF AUTONOMOUS AGENTS.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 441-446
DOI: 10.5220/0001511704410446

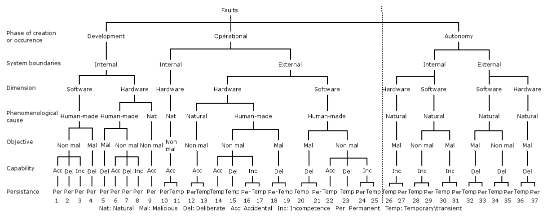

Faults
Phase of creation or occurence
Development Opérational Autonomy
System boundaries
Internal Internal External Internal External
Dimension
Software Hardware Hardware Hardware Software Hardware Software Software Hardware
Phenomenological cause
Human-made Human-made Nat Nat Natural Human-made Human-made Natural Natural Natural Natural
Objective
Non mal Mal Mal Non mal Non mal Non mal Non mal Non mal Mal Mal Non mal Mal Non mal Mal Mal Non mal Mal
Capability
Acc Del Inc Del Del Acc Del Inc Acc Acc Acc Acc Del Inc Del Del Acc Del Inc Inc Inc Inc Del Del Del
Persistance
Per Per Per Per Per Per Per Per Per Per Temp Per Temp Temp Temp Per Temp Per Temp Temp Per Temp Temp Per Temp Temp Per Temp Per Temp Per Temp Per Temp Per Temp Per
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
Nat: Natural Mal: Malicious Del: Deliberate Acc: Accidental Inc: Incompetence Per: Permanent Temp: Temporary\transient

Figure 1: Fault classes combinations.

This results on the following conclusion: the EMP is an error potentially resulting from any of the 37 fault classes presented before, which means: for all the possible faults occurring in MAS. When message passing is the only mean of communication for agents, the EMP is a serious issue and finding an adapted handler is a difficult task for the developer. Therefore, making handlers adapted to a specific situation (for a specific interaction protocol in a specific system) and for a specific class of faults cannot lead to an easy handling of the EMP.

This paper will be organized as follow. Section 2 is a view of the related work. Section 3 presents our proposed handler. Section 4 proposes an overview of the integration of our solution into the behavior of agents and experimental results. Section 5 proposes discussions related to the proposed method and the last section concludes and presents some perspectives.

2 RELATED WORK

Two main line of research in the MAS domain have similar goals than our work (we exclude here the ad hoc adaptation of interaction protocols and non reusable methods), they are presented next.

One line of research for agents fault tolerance is to include the fault tolerance directly into the languages representing the behavior of the agent. The creators of Cool (Barbuceanu and Fox, 1995) or AgenTalk (Kuwabara, 1996) in their papers introducing their respective languages also present some prospects on mechanisms to handle faults into the conversations. They propose to use a meta script to control the execution of the behavioral scripts, the meta scripts being started with messages, representing the exceptions, sent from the behavioral script to the meta script. The authors do not provide any general meta script of control and assume that they will be built in a suitable way for each exception. They conclude saying that the meta scripts can maybe be grouped together using exception classes.
More recently, Dragoni and Gaspari (Dragoni and Gaspari, 2006) proposed a fault tolerant agent communication language that deals with crash failure of agent, providing fault tolerant communication primitives and support for an anonymous interaction protocol. The knowledge level of the FT-ACL is based on giving for each speech act a success continuation and a failure continuation.
These methods are focused on conversational agents and on the handling of faults perceived into the communications but the realization of the suitable handler is the responsibility and the burden of the developer. The more serious issue of such a handler creation is the risk for the developer to introduce new faults into its handler. As shown after, our EMP handling proposition can be seen as a general meta script or a general failure continuation method.

Another line of research in MAS, on exception handling for MAS (Klein and Dellarocas, 1999) uses sentinels to diagnose (from the analysis of the received and sent messages) and handle exceptions for agents. This approach is different from the approaches presented previously since the diagnosis and decision about the handler adapted to the fault is outside the agent. The choice of the handler can be done inside a knowledge-base containing specific handlers (Klein and Dellarocas, 2000). Besides the creation of the external entity, it requires some special conception of the agents to allow an external entity to change its internal state. Our EMP handling proposition can be viewed as a handler that sentinels can choose to manage a diagnosed fault.

3 A HANDLER FOR THE EMP

3.1 Looking for a Fault Handler

The first observation, that must be done is that the proper handling of a fault depends on the fault being temporary or permanent since methods adapted to the first case are not relevant to the second and methods adapted to permanent faults (and then potentially for temporary faults) are too costly or too drastic. So, persistence is the more discriminating attribute for the handling of the faults during program execution. The work presented in this paper has been particularly focused on temporary faults but we study some prospects about the handling of permanent faults. Being concerned by temporary faults is a restrictive working hypothesis but since they are present when the system is running and they represent half of the fault classes, it is not too restrictive.

Even if, we consider that it is free to evaluate whether the fact that the message it expects to receive has not yet arrived is a ”normal” or ”exceptional” situ-

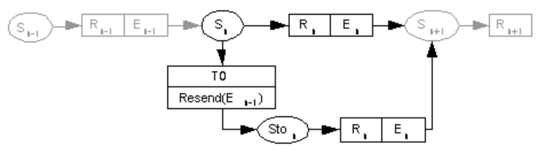


Figure 2: Behavior of an agent using the Resend method.

ation due to some form of autonomy. We assume that, on each state of an interaction protocol corresponding to the waiting of a specific response or event, the agent has a fault detection mean (for example a timeout). This allows detecting a fault before the agent does any other action into this interaction protocol. Furthermore, it seems natural that the agent can also evaluate the usefulness of a fault handler or another. This results on the following consideration: the agent must have handlers adapted to its knowledge level and some knowledge about how to chose and use them.

3.2 *Resend*: A Handler for the Agents

3.2.1 Key Idea

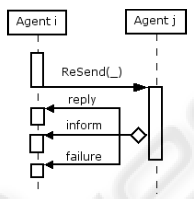
Resend corresponds to a performative and a protocol, initially suggested in (Potiron et al., 2007) as an example. It was designed for agents to handle some behavioral faults based on the argument that a time redundancy method can be used in a cooperative way and can be adapted to the knowledge level of the agents.

The key idea can be illustrated as follows:

- When your browser writes ”The server is down, please try again later” you refresh the page: you just use the time redundancy method sending the same request to the server;
- But when you send a mail and have no response you would rather send a new mail saying ”In a previous mail, I asked you this question and I am really bothered that I had no response about it.”

Thus, resend is used when an agent thinks that it should have received a response (or perceived a result expected after the sending of a message). It consists in sending a new message encapsulating the previous one to explain to the other agent that the expected response was not received. The behavior of an agent using the resend method is depicted on fig.2. When the agent considers an exception in state2 it sends the resend message and waits the result in a new state corresponding to a deviation of its behavior, state2 bis. If the expected result is perceived in state2 bis the agent continues its nominal behavior making the action2.

Table 1: Resend description.

Name	Resend
Description	An agent *i* tells an agent *j* that *i* desires that *j* processes the expression ϕ because *i* had not perceived any realization.
Message	The Resend performative contains the expression ϕ corresponding to the performative of the message.
Formal model	$\langle i, resend(j,\phi)\rangle \equiv \langle i, inform(j, U_i\phi \wedge I_i\phi)\rangle$ $FP: I_i\phi \wedge U_i(B_jI_i\phi \vee B_j\phi)$ $RE: B_jI_i\phi$
Protocol	 Agent i Agent j ReSend(_) reply inform failure

3.2.2 Formal Description

The performative, ”expresses a mental state to the other agent”, it corresponds to an ”expressive” speech act as defined in (Searle, 1969). A formal description of the resend performative, using FIPA-ACL formal logic (FIPA, 2001), and the associated protocol are given by tab.1.

The description of the Resend performative for KQML (Finin et al., 1997), would be: Normally the **receiver** of a performative will deliver its response as soon as a response is generated. The *resend* performative that takes a **¡performative¿** as its content, acts like a modifier on the usual order of affairs. It is a request to the **receiver** to handle the embedded performative as it could do because no response was perceived by the **sender** the first time it sends the embedded performative. The performative_name may be any of the performatives that requires a response.

3.2.3 Effectiveness

The use of resend allows managing the fault, in the case of an operational fault, in the same way the retry method does and with the same hypothesis (because they are time redundancy methods). But one advantage of the resend is that it is adapted to the knowledge level of the agents what allows them to decide whether to use it or not.
Moreover, the fact that the internal state of the agent is not the same when receiving ”resend” avoids repetition of the effects on the behavior of the agent what corrects a disadvantage of the retry method. This, plus the fact that the resend message has a different treatment than the original message, implies that

the original message and the resend message are not processed with the same lines of code. This might be compared to N-version programming and in some way allows the Resend method to handle some development faults which are permanent faults.
Another point concerning the handling of the behavioral faults is that, the performative allows the other agent to change its mind. Because the resend message has a specific meaning, insisting on the fact that the agent needs a response, if the agent has chosen not to respond to the first message it can, if possible or if interested, change its mind.
Another advantage to be noticed is that the message encapsulation makes our method not confusable with a stutter that is the repetition of the same message potentially due to a development fault.
Finally, a cooperative diagnosis is possible because the encapsulation allows the agents to identify the moment when the fault appeared and then eventually diagnose it with their common knowledge. For example, if an agent receives no response to a request and uses our method to handle this fault the other agent can respond that it has already answered to the request and that the first message may have been lost or ignored. Cooperative diagnosis is not investigated further in this paper.

3.2.4 Implementation

Practically, one part of the resend protocol must be applied after the detection of a fault; the other one into the conversation state where the resend is received.
After the detection the agent that wants to use the resend protocol must implement the agent i part of the protocol, see tab.1.
The difficulty of the resend method is: to understand when an agent can receive a resend message, because resend messages corresponds to exceptional situations. Resend message treatment, agent j part of the protocol, can be the same that when receiving the nominal message but it can be useful to provide a different one if important treatment are involved. Since the exact method for developing agents using our resend method depends a lot on the model and language used for the agent, we will present in the next section the method we use for our agents.

4 RESEND INTEGRATION

To present the integration of the resend protocol into the behavior of the agent we will separate the behavior of agent i (the one experiencing the EMP) and agent j (the one receiving the resend performative). The nominal behavior of agent i in coordination with agent j is depicted at the left part of fig.3, the right part depicts agent i and agent j behaviors using the resend.

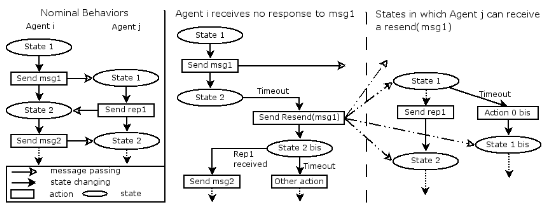


Figure 3: Resend and agents behavior.

4.1 Agent i

The integration of the resend protocol is presented only for the state 2 of the behavior of the agent i, in the middle of fig.3. The agent behavior is the following:

- When the agent i detects the EMP it sends the appropriate resend message to the agent j that changes its internal state for the state 2bis;
- In this state agent i waits for: the previously expected response or a resend message and has a detection means (like in any state);
- If the agent i receives a response it continues its normal behavior as if it never used the resend (the received answer may differ from the first one);
- If the agent i receives no response it considers that the fault is a permanent fault and then must search for another fault handler.

4.2 Agent j

The integration of the protocol is difficult because the resend message can be received at different states of the conversation. They are represented on the right part of fig.3 by the white arrows. For a given conversation state the agent may receive the resend message corresponding to:

1. The nominal waited message, example: the nominal message sent by agent i is ignored.
2. The previously received message, example: the response sent by agent j to agent i was ignored.
3. The following message into the protocol, example: the loss of the response sent by agent i.

The way the resend message is processed depends on when it is received. It comes under the responsibility of the developer of the agent because it is context dependant. It is also the case of the next state the receiving agent will reach.

4.3 Resend Implementation

The implementation of the resend into the behavior of the agent can be a quite complex task for the developer (like some meta-protocol proposed by FIPA). Fortunately, in MAS when agents use predefined interaction protocols the resend protocol can be implemented once for all with this protocols. This means that it can be given to the final developer of the agent a skeleton of the interaction protocol with the resend protocol. The developer has only the responsibility to complete the protocols with decisions methods and calculation operations but need not to wonder about how to use the resend method.

4.4 Tests

The implementation was done with the conversation based language presented in (Devèze et al., 2006). The language allows the description of the behavior of agents as a set of plans where states correspond to the waiting of a message. Tests were made with lost messages because, like explained in introduction, a lot of different faults can lead to the error corresponding to the EMP. Losing a message and having an agent choosing not to respond are part of them.

The results of the experimentation are that the handler succeeds when the fault does not last when the resend message is sent. For example the resend method does not allow handling the case when the message sent by the agent i is ignored by the agent j because it is overloaded and the corresponding resend message is received before the agent j is no more overloaded and so ignored too.

This shows the importance of the used detection means, for example the duration of the timeout, but it cannot be investigate further into this paper.

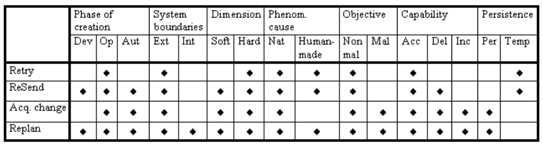
5 DISCUSSION

This section will investigate some points and some statements not developed about the resend handler.

5.1 Only Once

The reasons why we chose to send the resend message only once, and why we do not apply the resend method recursively, are the following:

- We propose a high level protocol so it is not designed to handle low level communication faults, moreover message loss are negligible.



	Phase of creation			System boundaries		Dimension		Phenom. cause		Objective		Capability			Persistence	
	Dev	Op	Aut	Ext	Int	Soft	Hard	Nat	Human-made	Non mal	Mal	Acc	Del	Inc	Per	Temp
Retry		◆		◆			◆	◆	◆	◆		◆				◆
ReSend	◆	◆	◆	◆		◆	◆	◆	◆	◆		◆	◆			◆
Acq. change		◆	◆	◆		◆	◆	◆		◆	◆	◆	◆	◆	◆	
Replan	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	

Figure 4: Evaluation of the attributes of the handled fault classes.

- If the same message is sent more than one time it become confusable with a stutter what is something we want to avoid.
- Sending more messages can overload the agent.
- The other agent is autonomous, so it is not obliged to answer and, moreover, it has maybe a good reason not to answer (overloaded for instance).
- The other agent might be dead or buggy so it becomes useless to waste more time trying to communicate with it.

5.1.1 Prospects about Permanent Faults

The resend method is adapted to temporary faults and so cannot guarantee that permanent faults will be handled (even if some development faults can be handled). In case of failure to manage the fault using the resend, the agent has other possibilities adapted to its knowledge level:

- To change its acquaintance: the agent can replace the agent with which it is making a transaction to obtain the needed result.
- To adapt its plan to the unexpected situation or create a new plan to obtain the needed result in a different way.

These methods are reconfiguration methods and so are adapted to permanent faults. But they involve a higher cost because they imply some explicit redundancy: to change its acquaintance the agent needs to find an available agent capable of delivering the wanted service; to change its plans the agent needs find an available agents capable of delivering the required new services. An evaluation of the fault classes handled is presented fig.4.

5.2 Formal Verification

Formal verification of the entire behavior of the agents is a difficult task, and formal verification of the resend protocol is meaningless. Nonetheless, considering the resend protocol ”encapsulated” into an interaction protocol permits to use verification methods and to provide a robust interaction protocol with guaranteed properties, concerning the handling of temporary faults.

We make the formal verification of our interaction protocols with timed automatas realized with UPPAAL (Bengtsson et al., 1995). For example deadlock verification can be checked , as the fact that it is always possible for the agents to reach the last state of the protocol.

6 FUTURE WORK AND CONCLUSIONS

This paper introduced and addressed the empty mailbox problem presenting its causes, and particularly the faults related to the specificities of MAS. After this, the paper focused on the adaptation from an existing low level handler used into distributed systems to our generic handler that fits agents knowledge level and then presents the handled faults and the prospects about the handling of permanent faults.

In future work the authors will investigate the diagnosis possibilities for the agents using the resend method. As it was underlined in the paper, the agent sending a resend message and the agent receiving it have a partial view of the situation. The possibility of diagnosis depends a lot on the state of the agent when it receives the resend message.

A lot of work has also to be done with regard to the effectiveness of fault tolerance methods, usually used for distributed systems, to MAS. Particularly, since MAS are compound with a low level entity (the platform) and high level entities (the agents), the sharing of the fault tolerance among the platform and the agents must be defined precisely to eventually allow the adaptation of other handlers to the agents.

REFERENCES

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. In computer society, I., editor, *IEEE Transactions on dependable and secure computing*, pages 11–33.

Barbuceanu, M. and Fox, M. S. (1995). Cool: A language for describing coordination in multiagent systems. In Lesser, V. and Gasser, L., editors, *Proceedings of the First International Conference oil Multi-Agent Systems*, pages 17–24, San Francisco, CA, USA. AAAI Press.

Bengtsson, J., Larsen, K. G., Larsson, F., Pettersson, P., and Yi, W. (1995). UPPAAL - a tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243.

Devèze, B., Chopinaud, C., and Taillibert, P. (2006). Alba: A generic library for programming mobile agents with prolog. In *PROMAS*, pages 129–148.

Dragoni, N. and Gaspari, M. (2006). Crash failure detection in asynchronous agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 13(3):355–390.

Finin, T., Labrou, Y., and Mayfield, J. (1997). KQML as an agent communication language. In Bradshaw, J., editor, *Software Agents*, Cambridge, MA. MIT Press.

FIPA, D. T. (2001). FIPA communicative act library specification.

Klein, M. and Dellarocas, C. (1999). Exception handling in agent systems. In Etzioni, O., Müller, J. P., and Bradshaw, J. M., editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 62–68, Seattle, WA, USA. ACM Press.

Klein, M. and Dellarocas, C. (2000). A knowledge-based approach to handling exceptions in workflow systems. *Computer Supported Cooperative Work*, 9(3/4):399–412.

Kuwabara, K. (1996). Meta-level control of coordination protocols. In *Second International Conference on Multi-Agent Systems*, pages 165–172.

Potiron, K., Taillibert, P., and Fallah-Seghrouchni, A. E. (2007). A step towards fault tolerance for multi-agent systems. In *Languages, Methodologies and Development Tools for Multi-Agent Systems First International Workshop, Revised Selected Papers*. Lecture Notes in Computer Science , Vol. 5118.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.