A Step towards Fault Tolerance for Multi-Agent Systems

Katia Potiron⋆†, Patrick Taillibert⋆, and Amal El Fallah Seghrouchni†

⋆Thales Systèmes Aéroportés
2 avenue Gay Lussac
78852 Elancourt – FRANCE
{katia.potiron,patrick.taillibert}@fr.thalesgroup.com
†LIP6
104 avenue du Président Kennedy
75016 Paris – FRANCE
{amal.elfallah}@lip6.fr

Abstract. Robustness, through fault tolerance, is a property often put forward in order to advocate MAS. The question is: What is the first step to be fault tolerant? Obviously the answer is: to know faults. The claim of this paper is that existing fault classification suitable for distributed systems does not fit completely MAS needs because of autonomy, the main characteristic of their components. Actually autonomy is the very distinctive concept of agents and has unquestionable worthwhile properties. But do these properties have no compensation?
After a short presentation of the fault classification which prevails in fault tolerance community, the paper will show that autonomy induces a need for significant extension to this classification. It will then make a special review of this extension and present some expectations with regard to the programing of fault tolerant MAS.

Key words: Fault tolerance, MAS design, autonomy

1 Introduction

Autonomy is one of the major characteristics of agents, and one issue of the MAS domain has been to precisely define that concept. In this paper, we consider a point that most definitions have in common: autonomy allows agents to take their decisions on their own, see for example [dL96,Hex01,CF03].
Agents, taking their decisions on their own gain some independence with regard to other agents. They can go on an dthus survive even if some other agent is not available. This aspect of autonomy makes agents more robust.
But, as a consequence of this decision making, the behavior of an agent is not completely foreseeable for the agents interacting with it. The question here can be: How to interact with autonomous agents? Agents eventually have to make some assumptions about the behavior of the other agents. But what is to be done if an agent does not fulfill these expectations? In brief, agents taking their

decisions by themselves can, voluntarily or not, be responsible for faults which other agents will experience.
From a software engineering point of view, autonomy can be perceived as the fact that the agent designer does not know exactly the specifications and internal laws of other agents or their internal state during their execution. Even if MAS are distributed systems, this is the break point opposing their design.

Faults are a concern for MAS designers, especially because agents are interacting with unpredictable agents. To deal with faults, the designer must have some precise information on the faults MAS are subject to. But what is a fault? Faults are generally defined as judged or hypothesized causes of an error. They are not dangerous when handled properly because only errors have direct effect on the system. Nonetheless, since not all faults can be detected during the system design and tests, their handling must be viewed as a natural feature of computing systems. Some works on MAS were done into this direction.
For example, some research on exception handling for MAS [KD99,PSH06] deals with *exceptional situations*. In this researches exceptions are detected as situations not suitable with the expectations of the agent. More practically they are defined as the messages that materialize fault detection as it creates an error [Lap85].
Another line of research in MAS is the replication of agents, a well-known fault tolerance method which deals particularly with physical faults [FD02,GFB05]. "Prevention of harmful behaviors" [CFST06] which deals with the emergence of harmful behaviors of agents and "fault tolerant agents communication language" which deals with crash failure detection [DG06] are other approaches.

Our final goal is to find a way to build MAS where fault tolerance is naturally a property of agents and system (achievable without a specific effort of the designer). To obtain such a good property, fault tolerance must be "made for MAS" and hence takes into account specificities of MAS, particularly autonomy of agents. The fault classification presented in this paper is our first step as it gives a first tool for designers to specify systems and agents. The paper will be organized as follow. The first section presents fault classification issue and method. The second section explains our contribution on classification of MAS faults. A study of the usefulness of such a classification is presented in the next section and the last section concludes our paper and presents our perspectives.

2 Conventional fault classification

The seminal work in fault classification is the work done by [Lap85,ALRL04,ACD+06] that began in the early 80's. Authors studied a wide group of faults to make their classification, including faults like short-circuits in integrated circuit, programmer's mistakes, electromagnetic perturbations or inappropriate man-machine interactions.

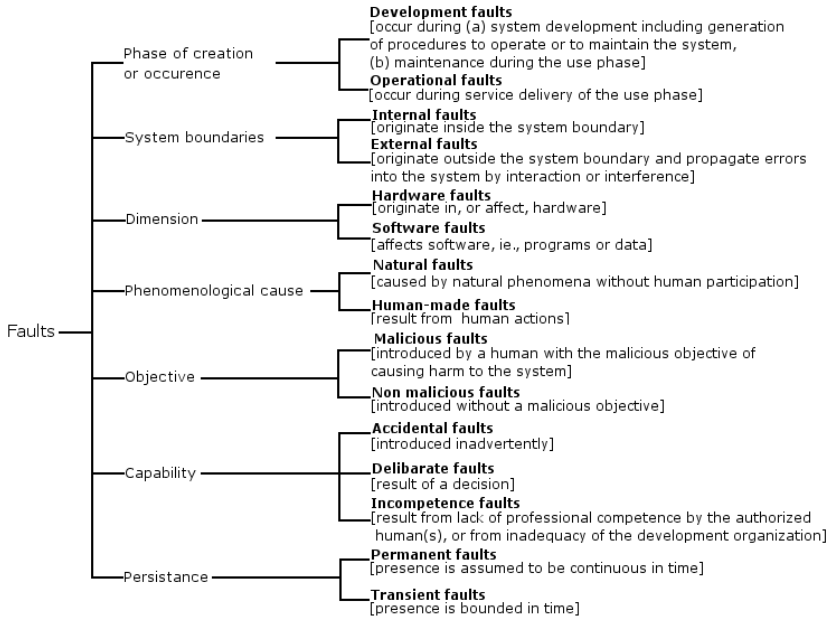

Fig. 1. Attributes to describe faults and their values

In [ACD+06], faults are classified according to seven attributes[1] (Phase of creation or occurrence, system boundaries, dimension, phenomenological cause, objective, capability, persistence). Each attribute has a set of exclusive values (for example values related to the attribute phase of creation are: development faults or operational faults), as shown on Fig.1.

A fault is described as a complete assignment of a single value to each attribute. Doing so the seven attributes and their values give 192 possible combinations. Not all these combinations were kept by the authors, because not all combinations are relevant. Thus a fault cannot be malicious and accidental at the same time. The 25 faults remaining, that are named DSFaults (Distributed Systems Faults) in this paper, are illustrated in the fault classification tree presented on Fig.2.

After this formalization of faults classification, some illustrative examples are given on the boxes at the bottom of Fig.3. The faults are also shown to belong to three *major non-exclusive groups* representing some practical points of view (two first lines of Fig.3) and described as follow:

[1] In [ALRL04], there were eight attributes as capability was separated into: 1) intent with deliberate and non-deliberate faults as values and 2) capability with accidental and incompetence faults as values. But the intent viewpoint appeared redundant.

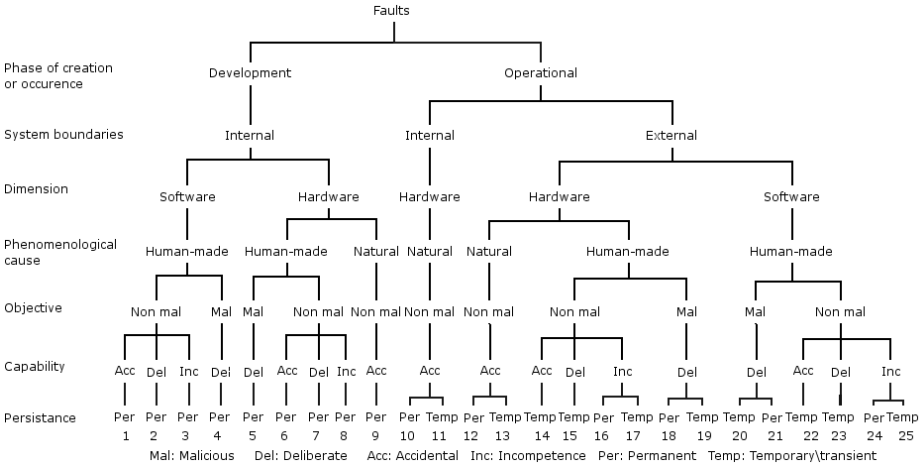

Fig. 2. Fault classes combinations

- **Development faults**: faults occurring during development.
- **Physical faults**: faults affecting hardware.
- **Interaction faults**: external faults.

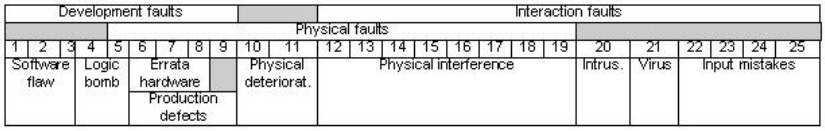

Fig. 3. Groups of fault examples

The knowledge of all possible fault classes allows the system designer to decide, during the system specifications, which fault classes should most be taken into consideration.

The next section will investigate this fault classification using a MAS point of view. As matter of fact, the fault classification presented here is in particular suitable for distributed systems (which MAS are) but, as explained in the introduction, MAS have some specificities (in particular the autonomy), which motivate further investigations on the fault classification. Our goal is to emphasize the pertinent faults for MAS (belonging to Fig.2) and to demonstrate that new specific faults are needed for MAS. This is the purpose of the next section.

3 MAS faults classification

MAS are separated components interacting each other to achieve some goal as in the case of distributed systems. What introduce the following findings:

- Since they are made of programs, MAS are vulnerable to all faults grouped as development faults;
- Since they are distributed programs, MAS are vulnerable to all faults grouped as physical faults;
- Since they are composed of interacting programs, MAS are vulnerable to all faults grouped as interaction faults.

The classification tree presented (Fig.2) is entirely relevant for MAS.

But, since MAS are composed of autonomous components we argue that an extension to the DSFaults classification is necessary. For doing so, a new value is given to the first attribute of the DSFaults classification. Specific faults of MAS are investigated and classified from the agents point of view and from an "external to the MAS" point of view.

3.1 A new value for the first attribute "phase of creation or occurrence"

The autonomy of the agents is the most salient difference between MAS and classical distributed systems. It is perceived by agents as the impossibility to predict the behavior of other agents. This unpredictability is the point studied here as possible fault source.
When considering these faults for the first time, we tried to classify them with one of the two existing values of the first classification attribute "the phase of creation or occurrence"[2]. These attributes presented into (Fig.1) are "Development faults" what means faults occurring during system development (they occur before the execution of the considered "program") and "Operational faults" what means faults occurring during service delivery of the use phase (they occur when executing the considered system interacting with programs or human beings). But the result was not what we expected:

1. These faults can not be considered as development faults because autonomy is a natural feature of agents.
2. These faults can not be considered as operational faults because they do not occur because of *service delivery* but because agents have an *autonomous behavior* what is not related.

This makes us consider autonomy as a new value of the attribute "phase of creation or occurrence" as shown in Fig.4. *Autonomy value* will represent faults occurring during the "autonomous behavior" of an agent. When employing: "autonomous behavior" we mean all actions that autonomy allows to the agents, as for example:

[2] Phase of creation or occurrence is related to the moment when the fault is made.

– Not to respond to a request ("the power to say no") or respond negatively whether or not it is include into the interaction protocol.
– To make a fault in order to incapacitate another agent.
– Not doing what was promised.

Concerning the three non-exclusive fault groups (development faults, physical faults or interaction faults) presented on Fig.3, faults which first attribute is valued as "autonomy" cannot be considered as belonging exclusively to anyone. We named these faults: *Behavioral faults* and consider it as a fourth non-exclusive group of faults presented in Fig.8.

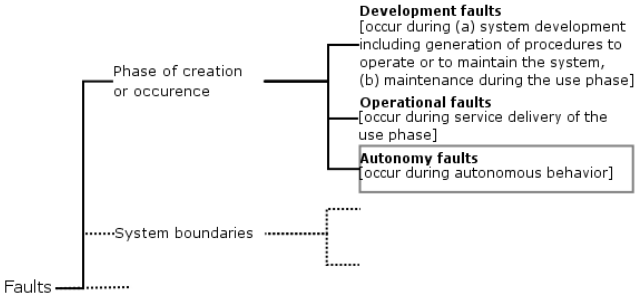

Fig. 4. Autonomy as a value of the attribute "phase of creation or occurrence"

The introduction of a new value for an attribute creates 96 new possible combinations among which the following analysis shows that 12 correspond to relevant new fault classes. We will present these new fault classes using two different points of view: agent centered (section 3.2) and "external to the MAS" centered (section 3.3).

3.2 Agent centered analysis

The frame of the study from the point of view of the agent is defined as follows:

– System: agent (named B on next examples)
– Environment: other agents or a human user (we have intentionally excluded the environment in the MAS sense)
– System boundaries: communication means of the agents
– User: other agents or a human user

A behavioral fault, on an agent-centered point of view, is equivalent to the "freedom" that autonomy gives to other agents and their unpredictability. This implies that the considered act is a fault only for the agent in interaction with the agent having an autonomous behavior not for the second agent. We give six examples to illustrate some corresponding situations.

1. An agent A from time to time voluntarily commits a fault to interfere with an agent B. For example sending a wrong message because it have chosen not to follow a correct interaction protocol (if the protocol was not correct it would be an operational fault).
2. Same as example 1, but permanent.
3. An agent A evaluates that it has no time to respond and so agent B does not receive any answer (duration of faults is time bounded and link to the agent a context).
4. Same as example 3, but not bounded in time.
5. Physical attacks between agents like temporary spam.
6. Same as example 5 but permanent.

For these faults, the values of the attributes are:

- **Phase of creation or occurrence:** *Autonomy*.
- **System boundaries:** *External*; because its source is in the other agent (an "internal to the agent" fault would be a development fault).
- **Dimension:** *Software*; autonomy comes from the agent implementation (examples 1 to 4) or *Hardware*, autonomy cannot come from hardware but can influence it (examples 5 and 6).
- **Phenomenological cause:** *Natural*; autonomy does not allow a human being to dictate its behavior to the agent.
- **Objective:** *Non-malicious* (examples 3 and 4) or *Malicious* (examples 1, 2, 5 and 6).
- **Capability:** *Deliberate* fault; results from the decision of an agent.
- **Persistence:** *Transient*; if the decision context is bounded in time (examples 1, 3 and 5) or *Permanent* (examples 2, 4 and 6).

This classification is represented by the tree of fault classes number 32 to 37 on Fig.5, for a more global view see Fig.7.

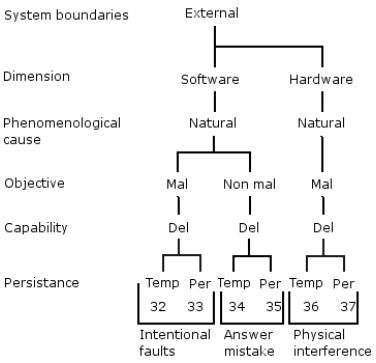

Fig. 5. External behavioral faults

3.3 System centered analysis

The frame of the study from an external point of view is defined as follows:

- System: MAS.
- Environment: other MAS or a human user.
- System boundaries: MAS interface.
- User: other agents, MAS or a human user.

A behavioral fault in the "external MAS"-centered point of view is comparable to the incompetence to handle the autonomy of other agents. This refers to how an agent can handle the autonomous behavior of the agents it is in interaction with. Faults are observable for an "external MAS" point of view only if an agent is incompetent to handle some autonomous behavior. We give six examples to illustrate some corresponding situations.

1. An agent overloads the network creating temporary problems considering messages transmission time.
2. Same as example 1 but not bounded in time.
3. An agent is incompetent to realize its goal because of another agent reaction (request refusal) and temporarily has no other way to realize his goal.
4. Same as example 3 but permanent.
5. An agent creates voluntarily a temporary fault to prevent another agent from accomplishing his goal.
6. Same as example 5 but permanent.

For these faults the values of the attributes are:

- **Phase of creation or occurrence:** *Autonomy.*
- **System boundaries:** *Internal*; because its source is into the MAS.
- **Dimension:** *Software*; autonomy comes from the agent implementation (examples 3 to 6),or *Hardware*, autonomy cannot come from hardware but can influence it (examples 1 and 2).
- **Phenomenological cause:** *Natural*; autonomy does not allow a human being to dictate his behavior to the agent.
- **Objective:** *Non-malicious* (examples 3 and 4) or *Malicious* (examples 1, 2, 5 and 6).
- **Capability:** *Incompetence* fault; results of an agent incompetence to adapt itself to the non-expectable behavior of the other agents or to changes in the environment.
- **Persistence:** *Transient*; if the decision context is bounded in time (examples 1, 3 and 5), or *Permanent* (examples 2, 4 and 6).

This classification is represented by the tree of fault classes number 26 to 31 on Fig.6, for a more global view see Fig.7.

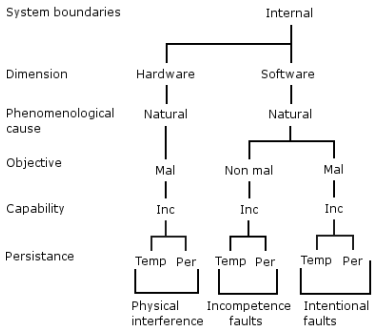

Fig. 6. Internal behavioral faults

3.4 Faults review

To begin a review of new faults introduced, named AAFaults (Autonomous Agent Faults) in this paper, Fig.7 shows the new fault classification tree.
As shown at the bottom of Fig.5 and Fig.6 (and summarized on last line of Fig.8), some faults can be gathered into fault classes examples. Malicious software fault group is named intentional fault group, as they are faults committed intentionally by agents. External non-malicious deliberate fault group is named answer mistake, as they are faults committed without bad intention. Internal non-malicious incompetence fault group is named incompetence fault group as this faults result from the agent incompetence in front of other agents autonomy. Hardware fault group is named physical interference as they are close to this example class. Moreover some behavioral faults can be classified as development,

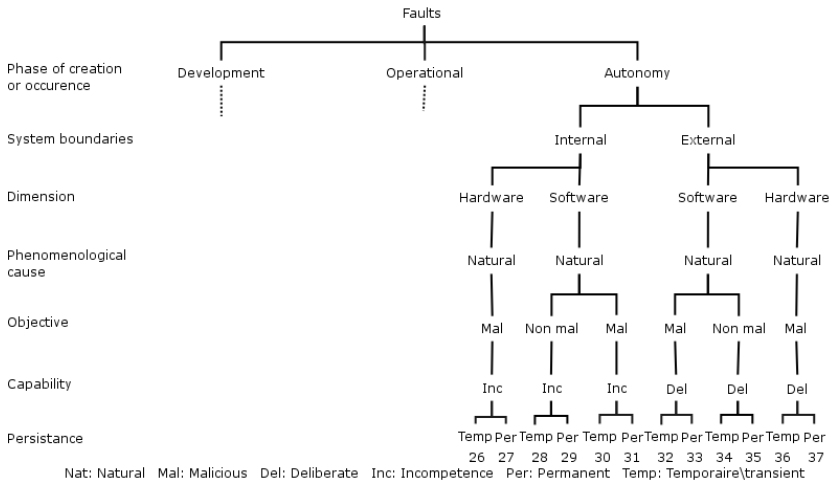

Fig. 7. New fault classes combinations

physical or interaction faults like shown on Fig.8.
Faults 26 to 31 are development faults as these faults come from agents or system incompetence to handle autonomous behavior of agents. They can create some service outage and can force system in a degraded mode or it can, at worst, stop its execution.
Faults 32 to 37 are interaction faults because these all are external to the considered system. They can create some local service failure (not always observable from an external point of view).
Some of these faults can also be viewed as physical faults (faults 26, 27, 36 and 37) because of their influence on hardware.

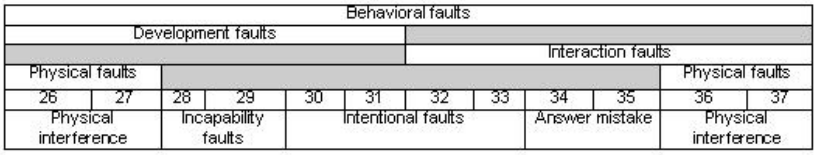

Fig. 8. Behavioral faults example classes

4 Validity of our approach

4.1 Faults comparison

In order to analyze the relevance of the behavioral faults (named AAFaults) we propose, we made a comparison to evaluate their similarity with regard to the pre-existing DSFaults. To do this, we compute a similarity measurement representing the number of common values of the attributes describing two faults, and defined as:

$$Similarity_{ab} = \sum_{ij} \delta_{ij}$$

With i (resp. j) in the set of values describing fault a (resp. b) and δ_{ij} the Kronecker symbol [3].

For example, the similarity measurement of faults 20 and 32 is shown on Fig.9. Fault 20 is described as: "Operational, External, Software, Human-made, Malicious, Deliberate, Temporary", and fault 32 is described as: "Autonomy, External, Software, Natural, Malicious, Deliberate, Temporary", their similarity score is equal to 5. The biggest possible *similarity score* is six since the first attribute ("phase of creation or occurence") always have a different value between DSFaults and AAFaults. Results are presented on Tab.1, lines are AAFaults,

[3] $\delta_{ij} = 0$ if i $\neq$ j; $\delta_{ij} = 1$ if i = j

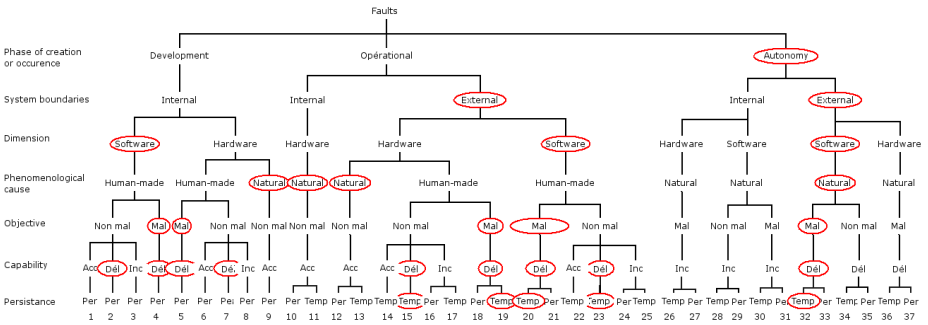

Fig. 9. Comparison of fault 32 to DSFaults

columns are similarity scores and the values are the DSFaults having with AAFault the corresponding similarity score.

AAFaults/Similarity	3	4	5
26	5, 8, 9, 13, 17, 19	11	
27	3, 4, 6, 7, 12, 16, 18	5, 8, 9, 10	
28	8, 9, 13, 17, 22, 23	3, 11, 25	
29	4, 6, 7, 12, 16	10, 9, 8, 1, 2, 24	3
30	3, 4, 11, 20, 25		
31	1, 2, 5, 8, 9, 10, 21, 24	3, 4	
32	4, 13, 15, 18, 22, 25	19, 21, 23	20
33	2, 5, 12, 19, 23, 24	4, 18, 20	21
34	2, 11, 12, 14, 17, 19, 21, 24	13, 15, 22, 25	23
35	1, 3, 4, 7, 9, 10, 13, 18, 22, 25	2, 12, 16, 21, 23, 24	
36	5, 11, 12, 14, 17, 21, 23	13, 15, 20	19
37	4, 7, 9, 10, 13, 15, 16, 20	5, 12, 21	18

Table 1. Faults comparison

General review: Tab.1 shows that some DSFaults are very similar to AAFaults. But the first observation is that there are no similarity score better than 5. This tends to confirm the necessity of introducing AAFaults as they are not redundant what would have been the conclusion if a similarity score was 6.
Another observation that can be done is that fault number 3 (a development fault) is most present as similar fault. It is not possible to make some conclusions right now but some questions can be raised. Did this means that autonomy is, in term of software development, near to the capability to violate specifications, what is the characteristic of fault 3? Did the fact this fault is viewed four times implies that MAS are more subject to some kind of faults than classical distributed systems?

As different fault classes can lead to very similar errors, comparison of faults is a good mean to improve MAS fault tolerance. If behavioral faults can be considered similar to other faults in the given classification, we can think that being tolerant to these faults gives some tolerance to the other faults.

4.2 Analysis of the difference between DSFaults and AAFaults

The similarity score study we made underlines some similarities between DSFaults and AAFaults but also some differences that are now explained.

On natural faults: All faults introduced by our study are natural faults since autonomy is a natural component of the agents. Difference is that, for natural faults in DSFaults, phenomenological cause being natural does imply only non-malicious and physical faults. Some explanation comes when taking into account the fact that agents are programs allowed to be malicious; they can act not only at physical level but also at interaction level. The consideration we made on autonomy would tend to unify humans-made faults and faults made by agents.

On malicious faults: Malicious faults are now not anymore human-made. Autonomy and, more generally agents, introduces into the MAS entities that have some specificities usually belonging to humans (autonomy and independence). Particularly in open MAS, agents are not always cooperative, they are able to decide to make malicious actions towards other agents.

One interesting point to emphasize here is behavioral faults prevention. A simple way to prevent all non-malicious behavioral faults is to send a preventing message. For example if an agent has too many messages to consider, it can send (without reading the messages to save some time) a cancel message or if it cannot deliver a result in time, ask for a delay. Prevention messages does not increase the number of messages exchanged since they are only used for exceptional situations and since if they did not exist it would be required some fault handling mechanism. However this is not well suitable to malicious faults, because the prevention can be useless if a fault is made with a malicious objective.

On interaction faults: Half of AAFaults faults are interaction faults. These behavioral faults are really close to DSFaults interaction faults; one outstanding point is that they differ according to their classification as natural (Fig.7). The closeness of DSFaults and AAFaults drives us towards the conclusion that, at least for interaction faults, some DSFault tolerance methods can be used, or adapted to MAS in order to handle groups of interaction faults belonging to DSFaults and AAFaults. These adapted methods could be able to handle interaction faults due to autonomy as well as those due to operational context.

All these observations lead to the consideration that faults related to agent autonomy can be treated partly as human interaction faults and partly as development or physical faults because their effects will be very similar.

Conclusion We have seen first that the autonomy of the agents is source of faults. But theoretically, with fault comparison and connection it seems possible to consider that if agents are tolerant to behavioral faults they can be tolerant to some other faults, particularly development and interaction faults making it possible to factorize the processing necessary to tolerate these faults. It seems that, contrary to what would be expected, fault tolerance methods used for distributed systems cannot guaranty the handling of behavioral faults because of the fundamental difference of assumptions made by interacting with autonomous agents. But some practical tests must be done to confirm these expectations.

Since the aim of this paper is to study fault tolerance suitable for MAS we will next present some perspectives based on the classification and factorization of faults we introduced.

5 Prospect about building fault tolerant MAS

As we said before our final goal is a fault tolerant method adapted to agents. A first direct use of the classification is to determine what faults must be handled by agents to be able to interact with autonomous agents in a dependable way. Designing MAS with autonomous agents does not allow to ignore these faults. A first step for a MAS designer would be to choose which faults the system (all the MAS including the agents and the platform) must be tolerant to. Especially a significant piece of work must be done on defining which faults must be handle by the platform or by the agents. The classification will then facilitate MAS specifications. Folliwing are some examples of specification of faults depending on MAS specificities:

- The platform would have to deal with all physical faults 5 to 19 plus 26, 27, 36 and 37 (because aware of hardware problems). But agents will have to handle the interaction faults not contained into physical faults as 20 to 25 plus 28 to 35. And development faults must be handled at their corresponding level (development faults occurring respectively into the platform/agent handled by the platform/agent level).
- In a closed MAS, if there are no malicious agents there are no reason that faults 26, 27, 30 to 33, 36 and 37 occur.
- For some other MAS it will not be considered that agents can commit physical faults. So faults 26, 27, 36 and 37 have no reason to be taken into consideration by agent designer.

We will consider next the issue of diagnosing faults. Effectively, as other programs, MAS would have to make some assumptions on faults they face because of the difficulty to diagnose exactly the faults.

Classification of the handlers : In the domain of fault handling, diagnosis is the first step to be able to choose the fault corresponding handler. For practical purpose, general classification methods exist. Some are particularly based on

temporal duration of faults [LS90] and evaluate faults as permanent, intermittent or transient to evaluate appropriate handler.
Principal existing methods for fault diagnosis are:

- Bayesian classification [LS90] and other classification techniques as k-nearest neighbor or neural networks are, for example, discussed in [CCT04].
- Comparison between an expected behavior, with some methods for detecting deviations from those expectations and faculties for diagnose these deviations [HLV+00].
- Social monitoring comparing own state (beliefs, goals, behavior) with socially similar peers' state [KT98].
- Model based diagnosis applied to software debugging [FFJS04].

After fault diagnosis an agent or a sentinel [Häg96,KD99] would have to choose some handlers to manage exceptions. This could be a way of using this classification. It would be possible to classify handlers with the same value and same attributes as the faults they can handle. Making so, the choice of the handler can be done by matching diagnosed properties of a fault and classification of the handlers.

Illustration of our expectations: the ReSend protocol. ReSend protocol is a protocol presented in [PTFS07]. It was designed for agents to handle some interaction faults based on the argument that a retry method can be used in a cooperative way. The agent can obtain by this way some information useful at its knowledge level, presumptions also made by [DG06]. When an agent thinks that it should have received a response to a message it sent before, it can send another message encapsulating the previous message to explain the issue to the other agent. The message encapsulation makes this method different from a retry.
We classified this protocol with regard to the values of the attributes of the faults it can handle. These faults are[4]: *external*, *time bounded* and *non-malicious* as faults: 13, 14, 15, 17, 22, 23, 25, 28 and 34.
But in the case of behavioral faults this protocol can be viewed as a method to point out the necessity of the previous message to have a response. So it can also be useful with some malicious behavioral faults since they can change agent decision like for 32 and 36.

6 Conclusion

After a summary of the existing fault classification which prevails in fault tolerance community, this paper has shown that autonomy induces a need of significant extension to this classification. To do so, it studied one of the consequence of autonomy, the most salient property of agents, that the behavior of an agent is not completely foreseeable for the other interacting agents. It implies that agents

[4] Since some attributes does not look discriminating as "phenomenological cause" we did not enumerate them.

taking their decisions by themselves can, voluntarily or not, be responsible for faults which other agents will experience.
Then the paper has pointed out the pertinent faults and demonstrated which specific faults were possible for MAS. Autonomy was added as a value of "phase of creation" attribute, representing faults occurring during an autonomous behavior. We defined a new group of faults named *Behavioral faults*. It has also made a special review of these 12 faults and presented some expectations since the fault classification presented in this paper is a first tool for designers to specify systems.
Finally this paper pointed out the use of the classification to determine what faults must be handled by agents to be able to interact with autonomous agents in a dependable way.

We would conclude citing [ALRL04]: "More combinations may be identified in the future".

References

[ACD+06] Jean Arlat, Yves Crouzet, Yves Deswarte, Jean-Charles Fabre, Jean-Claude Laprie, and David Powell. Tolérance aux fautes. In I.Comyn-Wattiau J.Akoka, editor, *Encyclopédie de l'Informatique et des Systèmes d'Information*, pages 241–270. Vuibert, 2006.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. In IEEE computer society, editor, *IEEE Transactions on dependable and secure computing*, pages 11–33, 2004.

[CCT04] M.Y. Cheng, S.C. Cheung, and T.H. Tse. Towards the application of classification techniques to test and identify faults in multimedia systems. In *Proceedings of the 4th International Conference on Quality Software (QSIC 2004)*, pages 32–40. IEEE Computer Society Press, 2004.

[CF03] Cristiano Castelfranchi and Rino Falcone. From automaticity to autonomy: the frontier of artificial agents. In Kluwer Academic, editor, *Agent Autonomy*, pages 103–136. Hexmoore H., Castelfranchi C., Falcone R, 2003.

[CFST06] Caroline Chopinaud, Amal El Fallah-Seghrouchni, and Patrick Taillibert. Prevention of harmful behaviors within cognitive and autonomous agents. In *European Conference on Artificial Intelligence*, pages 205–209, 2006.

[DG06] Nicola Dragoni and Mauro Gaspari. Crash failure detection in asynchronous agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 13(3):355–390, November 2006.

[dL96] Mark d'Inverno and Michael Luck. Understanding autonomous interaction. In W. Wahlster, editor, *12th European Conference on Artificial Intelligence*, pages 529–533. John Wiley and Sons, 1996.

[FD02] Alan Fedoruk and Ralph Deters. Improving fault-tolerance by replicating agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 737–744, Bologna, Italy, 2002. ACM Press.

[FFJS04] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumptner. Consistency-based diagnosis of configuration knowledge bases. In *Artificial Intelligence*, pages 213–234, 2004.

[GFB05] Zahia Guessoum, Nora Faci, and Jean-Pierre Briot. Adaptive replication of large-scale multi-agent systems: towards a fault-tolerant multi-agent platform. In *Proceedings of the fourth international workshop on Software engineering for large-scale multi-agent systems*, pages 1–6, St. Louis, Missouri, 2005. ACM Press.

[Hex01] Henry Hexmoor. Stages of autonomy determination. In IEEE computer society, editor, *IEEE Transactions on Systems, Man, and Cybernetics*, pages 509–517, 2001.

[Häg96] Staffan Hägg. A sentinel approach to fault handling in multi-agent systems. In *Second Australian Workshop on Distributed AI in conjunction with the Fourth Pacific Rim International Conference on Artificial Intelligence*, pages 181–195, 1996.

[HLV+00] Brian Horling, Victor Lesser, Régis Vincent, Ana Bazan, and Ping Xuan. Diagnosis as an integral part of multi-agent adaptability. In *DARPA Information Survivability Conference and Exposition*, volume 2, pages 211–219, 2000.

[KD99] Mark Klein and Chrysanthos Dellarocas. Exception handling in agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 62–68, Seattle, WA, USA, 1999. ACM Press.

[KT98] Gal A. Kaminka and Milind Tambe. What is wrong with us? improving robustness through social diagnosis. In *fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 97–104. American Association for Artificial Intelligence, 1998.

[Lap85] Jean-Claude Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *15th IEEE Symposium on Fault-Tolerant Computing (FTCS-15)*, pages 2–11. Vuibert, 1985.

[LS90] Tein-Hsiang Lin and Kang G. Shin. A bayesian approach to fault classification. In *ACM SIGMETRICS Performance Evaluation Review archive Volume 18 , Issue 1*, pages 58–66. ACM Press, 1990.

[PSH06] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. A definition of exceptions in agent-oriented computing. In Gregory O'Hare, Michael O'Grady, Oguz Dikenelli, and Alessandro Ricci, editors, *Engineering Societies in the Agent World'06*, 2006.

[PTFS07] Katia Potiron, Patrick Taillibert, and Amal El Fallah-Seghrouchni. Gestion des exceptions dans les conversations entre agents autonomes. In *JFSMA'07 (to appear)*, 2007.