Diagnosis of Unwanted Behaviours in Multi-Agent Systems

Lúcio S. Passos, Rosaldo J. F. Rossetti, and Joaquim Gabriel

Faculty of Engineering, University of Porto,
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
{pro09026,rossetti,jgabriel}@fe.up.pt

Abstract. Multi-Agent Systems (MAS) must increase their reliability in order to guarantee a solid and safe response to unexpected situations in complex domains. To address such an issue, both fault tolerance and software testing techniques are applied to ensure complying services; the foundation of these is the effectiveness in identifying and classifying fault events. Despite the advances in the field, most of fault-tolerant MAS rely on diagnosis mechanisms that demand for *a priori* knowledge of the system. Seeking to overcome this need for knowledge, we propose an extension of the Spectrum-based Fault Localization technique to be applied to MAS as it uses minimal information about the monitored system to detect agents presenting fault behaviours. For the moment, however, the proposed extension focuses on closed MAS and static environments. We successfully instantiate our approach and demonstrate its functionalities.

Keywords: Multi-agent System; Spectrum-based Fault Localization; Software Fault Diagnosis

1 Introduction

Agent-based solutions provide a set of desirable features to be deployed in real-world applications, though the body of knowledge in Multi-Agent Systems (MAS) still has not experienced an expressive growth regarding such deployments [12]. Pěchouček et al. [15] discuss some reason for such an issue and point as one of them the lack of reliability of MAS; in order to overcome this, it is necessary to include fault tolerance and testing techniques into the agent-development life-cycle coping with fault events on a sound basis. Both of these rely on the effective and correct recognition of an unforeseen even, known as *diagnosis*.

A diagnosis process demands for some knowledge about the system which may be built both from *a priori* models (given by designers), or information acquired by monitoring mechanisms; such model-based methods better suit deterministic domains. Therefore, it is crucial to tackle issues of diagnosing an exception in dynamic domains. To start this we first study the suggestion given by the fault tolerance field for an ideal diagnosis system: it must perform its duties regardless of any characteristic of the protected system as well as it must not produce overhead in order to avoid disruptions to the protected system's

performance. Still, agents add novel research challenges when identifying and diagnosing faults as they intensify some hazardous situations due to its non-determinism.

Current software diagnosis techniques demand for precise functioning models of the system and/or its components, or even have access to the source code where some marks might be added to control the execution flow. These needs, in the case of MAS, might be an issue as the access to the agent's source code is frequently impossible or undesirable due to several reasons. Thus another requirement to deal with agents is treat them as a black-box component. Indeed, multi-agent diagnosis should not rely on *a priori* knowledge being it structures or interactions, because this limits its capability to recognize failure in unforeseen contexts.

Endeavouring to fulfil these requirements, in this paper, we propose an Extended Spectrum-based Fault Localization approach to Multi-Agent Systems (ESFL-MAS) analysing the limitations of the current Spectrum-based Fault Localization (SFL) and thus expanding some characteristics so it supports diagnosis in a multi-agent system. In a nutshell, it assesses the overall and individual (i.e. per agent) performances during runtime while, by conveying such information, it diagnoses the most probable faulty agent; however, this first approach focus on closed MAS and static environments. Additionally, we instantiate the proposal in a grid-form environment where cleaning agents collect waste and drop it in the recycling station; with this illustrative experiment, we demonstrate that the ESFL-MAS is able to locate a faulty agent without any *a priori* knowledge of the multi-agent system.

The remainder of the paper is organized as follows. In Section 2 we discuss the related works in the field of fault diagnosis for MAS. Section 3 introduces a formalization of the spectrum-based diagnosis extension to MAS and, moreover, in Section 4 we instantiate the technique usage by presenting an illustrate example. In Section 5 we present conclusions as well as suggest future works.

2 Related Work

Effectively identify the origin and type of fault underlying a failure is the first step towards a more reliable multi-agent system; more formally, the diagnosis process is triggered by a detected deviation from the desired behaviour of a given system and further locates the responsible cause for the undesired event. Thus there are different techniques in the literature that focus on specific facets of agents while identifying unwanted events grounded upon two types of information: *fault-based* ones rely on experts to model all known faults in a given domain and *model-based* one depend on a system's model (i.e. its structure and behaviour) while any abnormality is classified as a fault [11].

The literature presents architectures based on fault description; for example, Shah et al.'s [19] proposed architecture detects deviations in runtime behaviour and diagnoses their underlying causes via heuristic classification techniques. Similarly, Dellarocas and Klein [2, 9] designed a domain-independent exceptions

handling for agent-based systems using a sentinel-based approach which diagnoses fault using heuristic classification to search in a pre-defined tree for possible causes based on detected symptoms; it introduces a knowledge base of generic exceptions. Such works are runtime dependent and impose previous knowledge of possible faults, to the contrary of what we aim with our proposal.

Some proposals, on the other hand, benefit from the specification model of the multi-agent system to identify unwanted behaviours. For instance, Roos et al.'s [17] approach diagnoses relying on spatially distributed models, i.e. each agent has a global view of the system; however, this method exchanges diagnosis precision for the ability of local minimal diagnosis. Furthermore, Letia et al. [10] presented a multi-agent monitoring and diagnosis scheme to supply these features to distributed applications; there are two types of diagnosis: first, it does in a local sphere and then in a cooperative one . Both of these endeavours centre on how to effectively merge knowledge and coordinate components whereas our work concentrates on how to minimize the knowledge usage.

Execution of plans is also an important matter for agent-based applications and thus Roos and Witteveen [18] introduce the concept of *plan diagnosis*. In later work, Jonge et al.[4] complement the previous concept with the *secondary diagnosis*; plan diagnosis defines the chain of actions towards a failure while secondary plan diagnosis determines causes for failure. Likewise, Micalizio [14] proposes an extended model of actions for partial observable plans which might be applied in the whole fault tolerance process. Opposite to diagnose failures in (previously defined) agents' plans, the detection of faulty behaviours is a context-dependent task and therefore should involve more components besides the plan analysis which the above works did not consider.

Another line of research is the *social diagnosis*, which seeks to recognize coordination failures within a team of agents. Kalech and Kaminka [8][7] model coordination among agents in a matrix structure and use it to detect the abnormal agent(s) and agents diagnose their teammates by internal model of them. Recently, the same authors (in [6]) improved identification efficiency to large-scale and more realistic teams. Also, Kafal and Torronis [5] approach diagnoses exceptions in contract execution which is a niche of applications. These social-diagnosis approaches have a team-oriented perspective, and yet overlook how local failures may impact in the global performance of a multi-agent system.

To the best of our knowledge, the assumption of *a priori* (either correct or faulty) knowledge is a constant in agent diagnosis endeavours; therefore, the accuracy of those diagnosis techniques depends on the specification completeness supplied by the design phase. However, our technique decouples the diagnosis process from given information as it collects dynamic spectra about the system during execution time.

3 Extended Spectrum-Based Fault Diagnosis for MAS

Spectrum-based Fault Localization is a statistical technique which analyses the software behaviour over multiples runs seeking reveal components that con-

tribute to an observed failure; it proved to be effective in software fault diagnosis (see e.g. [3]). This technique collects two essential types of information: *program spectrum* and *testing results*. The former is a set of runtime profiles that gives a specific view regarding the software behaviour [16]; the latter (per test case) informs the SFL whether the program behaves correctly (labelled as *passed*) or not (*failed*). Our proposed approach, inspired by SFL, aims to extend some of its characteristics to cover MAS applications.

Before presenting the technique, we define an agent and a multi-agent system based on [1] to cover only the fundamental aspects take into account by our approach. Agent and MAS are tightly interrelated as the first is embedded in the second and influences its performance as well.

Definition 1 *An* agent *is a tuple* $\langle \mathcal{S}, \mathcal{Y}, \mathcal{U}, p, h, s_0 \rangle$ *where:*

- $\mathcal{S}$ *is the internal state space of the agent.*
- $\mathcal{Y}$ *is the observation space of the agent; i.e. the set of possible observation the agent is able to perceive from the MAS.*
- $\mathcal{U}$ *is the action space available to the agent.*
- $p : \mathcal{S} \times \mathcal{Y} \longrightarrow \mathcal{S}$ *is the agent transition function, describing how the agent evolves as a result of its observations of the environment.*
- $h : \mathcal{S} \times \mathcal{Y} \times \mathcal{U} \longrightarrow [0, 1]$ *is the decision probability distribution of the agent, describing its behaviour.*
- s_0 *is the agent's initial internal state.*

Definition 2 *A* multi-agent system *is a tuple* $\langle \mathcal{A}, \mathcal{X}, f, \{\omega_i\}_{i \in \mathcal{A}}, p_i, \phi, x_0 \rangle$ *where:*

- $\mathcal{A}$ *is a set of agents,* $n = |A|$ *being their number.*
- $\mathcal{X}$ *is the environment state space.*
- $f : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \longrightarrow [0, 1]$ *with* $\mathcal{U} = \times_{i \in \mathcal{A}} \mathcal{U}_i$ *the joint action space, is the environment transition probability distribution, describing how the environment evolves as a result of the agents' actions.*
- $\{\omega_i\} : \mathcal{X} \times \mathcal{Y}_i \longrightarrow [0, 1]$, $i \in \mathcal{A}$ *are the observation probability distributions, describing how the n state of the environment is translated into agent observation.*
- $p_i : \mathcal{S}_i \times \mathcal{Y}_i \times \mathcal{U}_i \longrightarrow \mathbb{R}$, $i \in \mathcal{A}$ *is the agent performance, assessing how good an agent performs an action as result of its internal state and the observations of the environment.*
- $\phi : [p_1, \ldots, p_n] \longrightarrow \mathbb{R}$, *is the global performance of the system based on the single agent performance.*
- $x_0 \in \mathcal{X}$ *stands for the initial state of the MAS;*

We denote an agent in the set $\mathcal{A}$ simply by its index i and also indicate that an element belongs to agent i by subscript index i (e.g., S_i). In our interpretation, the agent's "mind" is composed by its state space $\mathcal{S}_i$ and the decision probability h_i while its sensory and actuating "body" is represented by the observation space $\mathcal{Y}_i$ and the set of available actions $\mathcal{U}_i$. The environment state, observation, internal state, and action space are considered discrete and finite in order to

match with our algorithm. The general notion of performance in this definition is given by the p_i and function ϕ; the element p_i is the agent performance seen by the system while ϕ function calculates the overall performance based on all individuals.

The Algorithm 1 emphasizes the dynamic nature of MAS to support our adaptation of the SFL. It can be interpreted in the following way. The variable k denotes the current value of the discrete time. The environment and states are components of a global state x_k. Agent i observes $y_{i,k} \in \mathcal{Y}_i$ at time k with probability designed by $\overline{\omega_i}(x_k)$ (line 3); then all agents update their internal states $s_{i,k+1} \longleftarrow d_i(s_{i,k}, y_{i,k})$ (line 4). The agents choose actions as their decision making function (line 5) and the environment evolves to the next state by $x_{k+1} = \overline{f}(x_k, u_k)$ (line 6). After this, all agents' performance are calculated by a function ψ_i that takes into account the action $u_{i,k}$ and the next state x_{k+1} of the environment; moreover, the function $\phi(p_{i,k})$ measures the overall performance, and then the cycles repeats all over again.

Algorithm 1: MAS evolution

1 $k \longleftarrow 0$
2 **for do**
3 $\quad y_{i,k} \longleftarrow \overline{\omega_i}(x_k)$, $\forall i \in A$
4 $\quad s_{i,k+1} \longleftarrow d_i(s_{i,k}, y_{i,k})$, $\forall i \in A$
5 $\quad u_{i,k} \longleftarrow \overline{h_i}(s_{i,k+1}, y_{i,k})$, $\forall i \in A$
6 $\quad x_{k+1} \longleftarrow \overline{f}(x_k, u_k)$, where $u_k = [u_{1,k}, \ldots, u_{n,k}]^T$
7 $\quad p_{i,k} \longleftarrow \psi_i(u_{i,k}, x_{k+1})$, $\forall i \in A$
8 $\quad p_{MAS} \longleftarrow \phi(p_{i,k})$, $\forall i \in A$
9 $\quad k \longleftarrow k + 1$
10 **end**

When analysing the original SFL proposed to object-oriented and procedural software development, we observe that its use in diagnosing faults in a multi-agent scenario is not straightforward; we must expand its capabilities to encompass important aspects of an agent. Hence, we below detail the implemented modifications focusing on each main component of the previous technique, namely: *test suite*, *spectra*, and *performance assessment*.

- **Test Suite.** It must be analysed in a twofold perspective. First, a simple software produces an output after a single execution whereas the MAS must be executed throughout several time steps so we can truly sees its behaviour. That is, for each test case, we must run the MAS during a time frame to take into consideration its changes. Second, a single execution per test case also limits our view from the MAS' behaviour as agents are autonomous and their activation paths (i.e. choices) might differ in each execution of the same test case; and, therefore, we must execute several rounds of the same test case to cover as many paths as possible to enhance coverage.

$$\underset{}{\mathcal{T}} \quad \underset{Ag_1\ Ag_2\ \cdots\ Ag_n}{X\left[T_\alpha, c_\beta\right]} \quad E\left[T_\alpha, c_\beta\right]$$
$$\begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_m \end{bmatrix} \times C \quad \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1K} & x_{2K} & \cdots & x_{nK} \end{bmatrix} \quad \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_K \end{bmatrix}$$

Fig. 1. Information of MAS P to ESFL-MAS

- **Spectra.** It is the crucial piece of information that bases the location of the faulty agent. However, the types of spectra so far proposed to object-oriented and procedural software development do not meet agent needs; for instance, if we choose a block hit spectra and apply it to our problem, the X matrix is going to be full of 1 because agents are persistent entities and are always active. As so we introduce a metric-based spectra, i.e. all agents are assessed by a metric and the 0s and 1s values indicate whether or not the agent is above a certain performance threshold respectively.
- **Performance Assessment.** ESFL-MAS should assess whether or not the MAS is in its normal functioning; we understand that this must rely on how effectively the overall system fulfils its requirements, in other words, how it performs the functions it was built to do. A direct solution is to evaluate performance using a domain-based metric that sets a performance threshold (e.g. a multi-agent system controlling an automotive shop-floor should produce 10 cars per hour). When the productivity is above this threshold the value represents that the run *passed*, otherwise it *failed*. In a more domain-independent approach, we intend to use a global utility function to measure the overall performance and, by comparing it with a certain threshold, determine the passed/failed states of the multi-agent system; this, however, is outside the scope of this paper.

After studying all required features to diagnose a multi-agent system, we propose the Extended Spectrum-based Fault Localization for Multi-Agent Systems; the Figure 1 presents the information of the multi-agent system P handled by the ESFL-MAS, where,

- $\mathcal{T}$ is a set of test cases: $\mathcal{T} = \{T_1, \cdots, T_m\}$, $m \in \mathbb{N}$ where m is the number of test cases.
- Test case T_i considers a set of $q \in \mathbb{N}$ environmental variable that all agents can perceive; that is $T = \{q_1, \cdots q_j\}$, $q_i \in \mathcal{X}$.
- $C \in \mathbb{N}$ is the number of executions per test case.
- K is total number of time step in a given execution.
- $X\left[T_\alpha, c_\beta\right]$, $0 \leqslant \alpha < m$, $0 \leqslant j < C$ is the spectrum matrix for the test case α and execution β. The whole tensor X has dimensionality of $m \times C \times K \times n$.
- E is a vector of boolean variables e_i, $0 \leqslant i < K$. A variable e_i represents the under performance of the system and its value is defined by the function

$$e_i = \begin{cases} 0 & \text{if } p_{MAS} \geq thr \\ 1 & \text{if } p_{MAS} < thr \end{cases}$$

where thr is a given performance threshold

The set of information of $\mathcal{T}$ test cases and C executions per test case constitute a 4-dimensional tensor ($m \times C \times K \times n$), where the metric-based spectrum is a matrix $X\,[T_\alpha, c_\beta]$; also its columns and lines correspond respectively to n different agents of a MAS and E time step per execution as we assume a discrete time. The information concerning in which time step an underperformance of the system was detected constitutes another vector named E. This vector represents a hypothetical part of MAS that is responsible for observing all failure events.

Spectrum-based diagnosis essentially consists in identifying the agent whose column vector resembles the underperformance vector the most; the similarity coefficient quantifies these resemblances and, assuming the high similarity with the underperformance vector indicates a high probability that the corresponding agent caused the detected "failure," it ranks agents with respect to their likelihood of containing faults. ESFL-MAS produces a tuple $\langle a_{ef}, a_{ep}, a_{nf}, a_{np} \rangle$ for each block of the multi-agent system P such that: a_{ef} and a_{ep} represent the number of test cases that covered the block and returns a failed and passed testing result respectively; whereas a_{nf} and a_{np} denote the number of test cases that do not execute the block and return a failed and passed testing result respectively.

In Algorithm 2 we present the diagnosis algorithm. First, the variables and vectors are initialized with the proper values (line 2-11); then it runs the MAS for the set of test cases and executions which builds the space of information and the error vector (line 12). For each agent and spectrum, the algorithm analyses the combination of agent's and overall performances to add in their respective counters (lines 13-29). At last, the similarity of each agent is calculated through the counter vector. The algorithm returns a sorted list with ascending order of the most probable faulty agents.

We must define some boundaries in the application of the above algorithm not to give the idea that it is the "silver bullet" for all MAS. This extension, in its current state, is only applicable to closed multi-agent systems that work within a static environment as it does not consider flow of agents and anything else, besides agents actions that make the environment change.

4 The Cleaning Agents Example

Demonstrating the different components of our technique and how they correlate demands for an illustrative example that shed light on each step of the process. For this purpose we used a closed cleaner agent scenario working within a static environment and at discrete time. The chosen scenario is composed by three cleaner agents, three wastes, and one recycling station. Every agent has as main goal to get a waste and bring it to the recycling stations. The actions the `Cleaner` agent can perform are: UP, $DOWN$, $LEFT$, $RIGHT$, $PICK$, and $DROP$; it is assumed that the agent has *a priori* knowledge about the recycling station's

Algorithm 2: ESFL-MAS Algorithm

Input: Multi-Agent System P, set of test cases $\mathcal{T}$, number of executions per test case C, and similarity coefficient s
Output: Diagnosis report D

```
1  begin
2      m ⟵ |T|
3      n ⟵ Get_NumOfAgents(P)
4      D ⟵ ∅
5      for j = 0 to n do
6          a11 (j) ⟵ 0
7          a10 (j) ⟵ 0
8          a01 (j) ⟵ 0
9          a00 (j) ⟵ 0
10         S[j] ⟵ 0 // Similarity s of agent j
11     end
12     (X, E) ⟵ Run_MAS(P, T, C)
13     for i = 0 to m do
14         for j = 0 to C do
15             K ⟵ Get_NumTimeSteps(X[i, j]) for k = 0 to K do
16                 for l = 0 to n do
17                     if x[i, j, k, l] = 1 ∧ e[i, j, k] = 1 then
18                         a11 (l) ⟵ a11 (l) + 1
19                     else if x[i, j, k, l] = 0 ∧ e[i, j, k] = 1 then
20                         a01 (l) ⟵ a01 (l) + 1
21                     else if x[i, j, k, l] = 1 ∧ e[i, j, k] = 0 then
22                         a10 (l) ⟵ a10 (l) + 1
23                     else if x[i, j, k, l] = 0 ∧ e[i, j, k] = 0 then
24                         a00 (l) ⟵ a00 (l) + 1
25                     end
26                 end
27             end
28         end
29     end
30     for i = 0 to n do
31         S[i] ⟵ s(a11 (i), a10 (i), a01 (i), a00 (i))
32     end
33     D ⟵ Sort(S)
34     return D
35 end
```

location. The environment is represented as a grid (where there is a bottleneck to get in the recycling station). We also consider that there is no communication between agents, yet they signalize their intended movement. Additionally, the test suite has three test cases as shown in Figure 2.

In our example one of the three agents has a fault which is known by us but unknown by the diagnosis system. The fault was a wrongly initialized belief, that

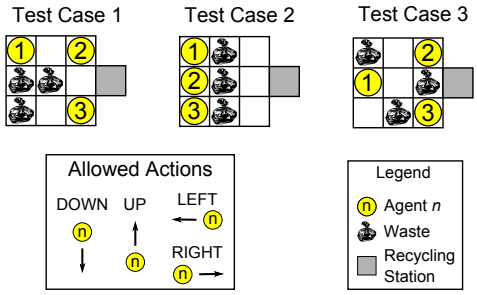

Fig. 2. Test cases and allowed actions in the example scenario.

is, in a normal condition an agent believes it has no waste; however, here agent 2 initially believes to have collected a waste. This leads to a failure in some of the test cases (i.e. underperformance), whereas it stays inactive in the others.

For obtaining the metric-based spectra, we define the minimal distance to reach the recycling station as well as to reach a waste; thus the chosen metrics are the following: *an agent must pick up a waste in four time steps* and *an agent must drop a waste in four time steps*. Both of them are very simple and effective for our purpose regardless their domain-dependent nature; we assume that external entities assess each agent. As for the runs, each test case was executed three times to cover more MAS's possible states as explained in Section 3; the discrete time of the spectrum matrix X was the time step.

Although the error detection is not the focus of our work, to instantiate our technique we had to implement it. The discussion on how to assess a multi-agent system requires a deeper look into the MAS definition. However, for the sake of illustration only, we adopted a simple metric for this example; therefore the chosen metric was *one waste must be received each 4 time steps*. Again this assessment vector of the system's performance was built for each execution per test case.

The similarity coefficient indicates the probability of a certain agent to be the faulty one and, for this illustrative example, we use a Ochiai [13] coefficient which is given by equation 1.

$$s_O = \frac{a_{11}}{\sqrt{(a_{11} + a_{01}) * (a_{11} + a_{10})}} \tag{1}$$

The obtained results were that the proposed technique was able to diagnose agent 2 as the faulty one since the vector returned by the algorithm was $D = \{agent\ 2, agent\ 1, agent\ 3\}$ with values of similarity being 0.304, 0.225, and 0.091 respectively. Nevertheless, there are two points we would like to highlights: first the similarity of agent 1 was closer to agent 2's which means that the diagnosis result can be jeopardized or even masked by a certain set of conditions; second, the built spectrum very much depends on the chosen metric which should be proposed by a specialist in the scenario.

5 Conclusion

Diagnosing faults in MAS is a very challenging task as several aspects should be taken into account in a timely fashion; another drawback is the reliance that current fault diagnosis techniques have on models provided by system designers. Our research aim is to propose a diagnosis technique that focuses on faulty behaviours in a multi-agent system relying on minimal given information. In this paper, we propose the extension of a known black-box diagnosis mechanism to identify agents jeopardizing the overall performance through misdirected reasoning. This proposal is called Extended Spectrum-Based Fault Localization that diagnose the cause of a failure (in our case an underperformance event) established by run-time spectrum from both agents' and system's performances; however this first appraisal addresses closed MAS and static environments. Also in this paper, we illustrate the ESFL-MAS in a cleaning agent scenario and as conclusion it was able to identify the faulty agent even with a small occurrence of under-performance situations; yet, it is important to highlight that the ESFL-MAS must be validated in more complex scenarios though we strongly believe that our approach can contribute to a more reliable, efficient, and deployable process to locate agents which decrease the overall performance of the system.

Several aspects of the technique should further developed to improve its response though. We first plan to deeply study the influence of different similarity coefficients and metrics on the accuracy of the ESFL-MAS. Another interesting branch of future research is to generalize the used metric aiming at a domain-independent proposal which would be based on individual and global utilities. Further explorations might take into account the social fabric in the identification process as agents can potentially lead others to faulty states, for instance by sending wrong perceptions of the environment. At last, a more immediate work is to investigate the cleaning scenario in a more realistic condition such as: larger environment, more agents, different test cases and executions in order to evaluate its robustness and scalability.

6 Acknowledgments

This project has been partially supported by FCT, the Portuguese Agency for R&D (PhD Scholarship grant SFRH/BD/66717/2009).

References

1. Buşoniu, L., De Schutter, B., Babuška, R.: Learning and coordination in dynamic multiagent systems. Tech. Rep. Technical Report 05-019, Delft Center for Systems and Control, Delft University of Technology (2005)
2. Dellarocas, C., Klein, M.: An experimental evaluation of domain-independent fault handling services in open multi-agent systems. In: MultiAgent Systems, 2000. Proceedings. Fourth International Conference on. pp. 95 –102 (2000)

3. Hofer, B., Wotawa, F., Abreu, R.: Ai for the win: improving spectrum-based fault localization. SIGSOFT Softw. Eng. Notes 37(6), 1–8 (Nov 2012), http://doi.acm.org/10.1145/2382756.2382784
4. Jonge, F., Roos, N., Witteveen, C.: Primary and secondary diagnosis of multi-agent plan execution. Autonomous Agents and Multi-Agent Systems 18(2), 267–294 (2009), http://dx.doi.org/10.1007/s10458-008-9045-x
5. Kafalı, Ö., Torroni, P.: Exception diagnosis in multiagent contract executions. Annals of Mathematics and Artificial Intelligence 64(1), 73–107 (2012), http://dx.doi.org/10.1007/s10472-012-9282-1
6. Kalech, M.: Diagnosis of coordination failures: a matrix-based approach. Autonomous Agents and Multi-Agent Systems 24(1), 69–103 (2012), http://dx.doi.org/10.1007/s10458-010-9144-3
7. Kalech, M., Kaminka, G.A.: On the design of coordination diagnosis algorithms for teams of situated agents. Artificial Intelligence 171(89), 491 – 513 (2007), http://www.sciencedirect.com/science/article/pii/S0004370207000483
8. Kalech, M., Lindner, M., Kaminka, G.A.: Matrix-based representation for coordination fault detection: a formal approach. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. pp. 162:1–162:8. AAMAS '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1329125.1329322
9. Klein, M., Dellarocas, C.: Exception handling in agent systems. In: Proceedings of the third annual conference on Autonomous Agents. pp. 62–68. AGENTS '99, ACM, New York, NY, USA (1999), http://doi.acm.org/10.1145/301136.301164
10. Letia, I., Cracium, F., Kope, Z., Netin, A.: Distributed diagnosis by bdi agents. In: Proceedings of the IASTED International Conference Applied Informatics. pp. 862–867. Innsbruck, Austria (February 14-17 2000)
11. Lucas, P.J.: Analysis of notions of diagnosis. Artificial Intelligence 105(12), 295 – 343 (1998), http://www.sciencedirect.com/science/article/pii/S0004370298000812
12. Mckean, J., Shorter, H., Luck, M., Mcburney, P., Willmott, S.: Technology diffusion: analysing the diffusion of agent technologies. Autonomous Agents and Multi-Agent Systems 17(3), 372–396 (Dec 2008), http://dx.doi.org/10.1007/s10458-008-9052-y
13. Meyer, A.d.S., Garcia, A.A.F., Souza, A.P.d., Souza Jr., C.L.d.: Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (zea mays l). Genetics and Molecular Biology 27, 83 – 91 (00 2004), http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1415-47572004000100014&nrm=iso
14. Micalizio, R.: Action failure recovery via model-based diagnosis and conformant planning. Computational Intelligence 29(2), 233–280 (2013), http://dx.doi.org/10.1111/j.1467-8640.2012.00444.x
15. Pěchouček, M., Mařík, V.: Industrial deployment of multi-agent technologies: review and selected case studies. Autonomous Agents and Multi-Agent Systems 17, 397–431 (2008), http://dx.doi.org/10.1007/s10458-008-9050-0
16. Reps, T., Ball, T., Das, M., Larus, J.: The use of program profiling for software maintenance with applications to the year 2000 problem. In: Jazayeri, M., Schauer, H. (eds.) Software Engineering ESEC/FSE'97, Lecture Notes in Computer Science, vol. 1301, pp. 432–449. Springer Berlin Heidelberg (1997), http://dx.doi.org/10.1007/3-540-63531-9_29

17. Roos, N., ten Teije, A., Witteveen, C.: Reaching diagnostic agreement in multi-agent diagnosis. In: Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on. pp. 1256 –1257 (july 2004)
18. Roos, N., Witteveen, C.: Models and methods for plan diagnosis. Autonomous Agents and Multi-Agent Systems 19(1), 30–52 (2009), http://dx.doi.org/10.1007/s10458-007-9017-6
19. Shah, N., Iqbal, R., James, A., Iqbal, K.: Exception representation and management in open multi-agent systems. Information Sciencies 179(15), 2555–2561 (Jul 2009), http://dx.doi.org/10.1016/j.ins.2009.01.034