A HYBRID DIAGNOSTIC-RECOMMENDATION SYSTEM FOR AGENT EXECUTION IN MULTI-AGENT SYSTEMS

Andrew Diniz da Costa, Carlos J. P. de Lucena
Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil

Viviane T. da Silva
Universidad Complutense de Madrid, Madrid, Spain

Paulo Alencar
University of Waterloo, Waterloo, Canada

Keywords: Multi-Agent Systems, Trust, Reputation, Diagnosis, Recommendation.

Abstract: Open multi-agent systems are societies with autonomous and heterogeneous agents that can work together to achieve similar or different goals. Agents executing in such systems may not be able to achieve their goals due to failures during system execution. This paper's main goals are to understand why such failures occurred and what can be done to remediate the problem. The distributed, dynamic and open nature of multi-agent systems calls for a new form of failure handling approach to address its unique requirements, which involves both diagnosing specific failures and recommending alternative plans for successful agent execution and goal attainment. In this paper, we discuss solutions to the main challenges of creating a system that can perform diagnoses and provide recommendations about agent executions to support goal attainment, and propose a hybrid diagnostic-recommendation framework that provides support for methods to address such challenges.

1 INTRODUCTION

Open multi-agent systems (Jennings and Wooldridge, 1999) are societies with autonomous and heterogeneous agents that can work together to achieve similar or different goals (Boella and Torre, 2004). In many cases, the agents are unable to attain their goals due to failures during system execution. When an agent tries to attain its desired goal, but faces execution failures that prevent achievement, it becomes relevant to understand why such failures occurred and what can be done to remediate the problem.

This paper focuses on the diagnosis of failures and on the recommendation of alternative plans for successful agent execution and goal attainment. Some proposals that have recently appeared in the literature suggest different ways for agents to diagnose system execution failures. Li et al. (Li et al., 2004) present a decentralized system to monitor and diagnose the agents' behavior. Although this is an interesting idea, in open multi-agent systems it is not applicable because when the execution of an agent is monitored its privacy is violated.

Another interesting work, which was proposed by Horling et al. in (Horling et al., 2000), examines how an independent domain for diagnoses can behave in multi-agent systems and compares the hoped result of an execution with the result obtained. However, the approach does not offer a large data set to define the expected behaviors, and there is no control of the reputation of the agents when an agent is detected as guilty of some failure.

In this paper we describe a new hybrid diagnostic-recommendation system of agents' executions that does not violate the agents' privacy and that defines a set of facts that can be used to provide information about the reasons for failing to achieve a goal. Diagnosis is assumed as the process of determining the reason why agents do not achieve

Diniz da Costa A., J. P. de Lucena C., T. da Silva V. and Alencar P. (2008).
A HYBRID DIAGNOSTIC-RECOMMENDATION SYSTEM FOR AGENT EXECUTION IN MULTI-AGENT SYSTEMS.
In *Proceedings of the Third International Conference on Software and Data Technologies - PL/DPS/KE*, pages 159-168
DOI: 10.5220/0001879401590168

their goals. Recommendations are provided on how to achieve the desired goals that agents have failed to achieve.

A diagnostic system must be able to analyze different sets of information related to the agents' executions and point out the (main) problem that has occurred. Recommendations are provided based on the diagnosis and indicate alternatives to the agent's execution to try to achieve the same goal. A recommendation system can recommend, among others, the use of another resource, the execution of another plan and the interaction with other agents. In order to recommend other partners to interact with the agent, the recommendation system bases its choice on the agents' reputations. The reputation of an agent is evaluated according to its past behavior. Several reputation systems have been proposed to collect, distribute, and aggregate feedback about agents' past behavior. In this paper we use the Governance Framework (Silva et al., 2007) to provide agent reputations.

The hybrid diagnostic-recommendation framework called DRP-MAS (Diagnosing and Recommending Plans in open Multi-Agent Systems) proposed in this paper can be instantiated to perform different kinds of diagnoses and to provide recommendations (advices) to help agents achieve their desired goals. This framework uses the Governance Framework to receive the agents' reputations that are used to offer advice about partners with which to interact.

This paper is structured as follows. In Section 2 we discuss some of the main difficulties of diagnosing and providing alternative execution strategies for agents to achieve their goals. In Section 3 we provide an overview of the DRP-MAS framework. Since our framework uses the Governance Framework to represent the reputation concept, in Section 4 we briefly explain how it was used. Section 5 illustrates the applicability of the framework through a case study and Section 6 presents some related work. Finally, in Section 7, conclusions and future work are discussed.

2 DIFFICULTIES OF PROVIDING DIAGNOSES AND RECOMMENDATIONS

In this section, we describe some of the challenges and requirements related to the process of performing diagnoses and providing recommendations to help agents to achieve their goals. These challenges and associated requirements include:

1. Deciding how to Analyze the Behavior of the Agents

The first challenge was to determine an appropriate way to analyze the behavior of the agents. Two solutions could be adopted. In the first, the execution of each agent is monitored. Since we are working with open multi-agent system environments and with heterogeneous agents, one of our requirements is to not violate the agents' privacy. In the second possibility, each agent analyzes its own execution. By using such an approach, the agent's privacy is not violated and the information stored in past analyses can be used by the agent in future ones. Due to these two reasons, our approach adopted the second solution.

2. Selecting Data for Diagnosing

One major challenge was to define which data is necessary to perform diagnoses related to the execution of agents. To perform the diagnoses, the following information can be used: problem which occurred due to limited memory space, the list of resources used and the ones that the agent tried to use, etc. In this paper we are considering a predefined list of information to be used in the framework composed of the plan executed, desired goal, norm violated, roles of the agents and the agents that provided some information during the execution of the plan, among others. Since different domains can require different information, such a list can be extended.

3. Determining Strategies for Diagnoses

Different domains can require different strategies to provide diagnoses. The challenge was to define services or strategies that could be used by different domains and to make available an infrastructure that could be extended to accept new strategies.

4. Determining Trustworthy Agents

The reason for the failure regarding the execution of plan can be related to the behavior of the partner with whom the agent has interacted. For instance, the partner may have provided a bad service or inadequate information. Therefore, when a diagnosis is formulated and it is verified that a specific agent was responsible for the unsuccessful execution, the recommendation system will try to avoid selecting such agent in the next advice. To solve the problem of distinguishing whether an agent is good or bad with respect to some criteria we are making use in this paper of agents' reputations.

5. Providing Recommendations
We meant to create a framework that could provide alternative ways of execution to achieve the same goal. Therefore, the big challenge was to define a strategy, which could be used in different domains and an infrastructure that could be extended to accept new strategies.

6. Representing Profiles of Agents
The same diagnostic can be associated with two different recommendations, depending on the characteristics of the agents that will receive such advices. The challenge is to define how to represent profiles of agents and how they could influence the recommendations provided by the proposed framework. The framework makes available a basic agent profile that specifies the minimum global reputation of partners to be considered in advices.

3 THE DRP-MAS FRAMEWORK

In this section, we describe the DRP-MAS framework that performs diagnoses about the failure to achieve the goals; moreover, it provides recommendations for agents about how to achieve their goals. Initially, the general idea of the framework is presented, followed by its architecture, and in the sequel we discuss the central concepts on which it is based.

3.1 The General Idea

The DRP-MAS framework is used when an agent does not achieve one of its goals after the execution of one of its plans. The agent of the application, the *Requester agent*, requests to the *Mediator agent* a *Diagnostic agent*. When the *Mediator* receives the message, it creates a *Diagnostic agent* (responsible for providing diagnoses) and a *Recommendation agent* (responsible for providing recommendations) and sends a message to the *Requester* informing which *Diagnostic agent* will work for it (Figure 1).

Subsequently, the *Requester* requests diagnoses to the *Diagnostic agent* in order to receive advices from the *Recommendation* agent to achieve the desired goal. For this purpose, it sends a message to the *Diagnostic agent* with the values of a set of attributes that can help it in the analyses, such as: plan executed, goal not achieved, the agents used in the negotiations with their played roles, its profile, and a number that represents the quality of the execution performed (details in Section 3.3). This idea of quality was based on the works (Horling et al., 2000) and (Horling et al., 2007).

When the *Diagnostic agent* receives the message, it tries to find the reason (s) why the *Requester agent* was unable to achieve the desired goal. At the end of the analysis, it provides the diagnosis to the *Recommendation agent*. Even if a diagnosis could not be provided, the *Diagnostic agent* sends a message to the *Recommendation agent* informing that it was not possible to detect the reason for the *Requester agent* not to have achieved the desired goal. In this case, when the *Recommendation agent* receives the message informing that it was not possible to meet a diagnosis, it simply selects another plan that achieves the desired goal.

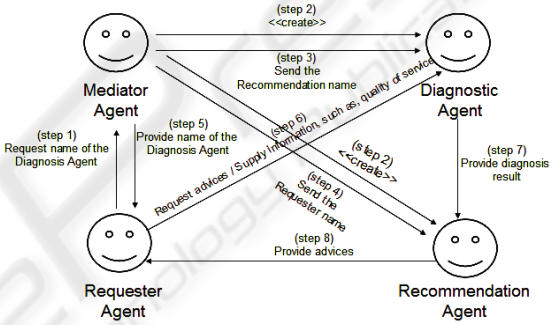

Figure 1: Conceptual Model for requesting the name of the diagnosis agent.

In the case that some diagnosis is met, the *Recommendation* searches alternative plans to achieve the goal (details in Section 3.4) by considering the data in the diagnosis. When the diagnosis indicates a problem with agent interaction, an analysis is made to decide which other agents could be used to perform the interactions (analysis performed based on the roles played).

From the set of agents that can perform the same roles in those interactions, the *Recommendation agent* uses the agents' reputations to select the "best" agents, i.e., the agents with the top reputations. The profile of the *Requester agent* can be an important piece of information to define which agents should be selected. When the execution of the *Recommendation agent* ends, a message to the *Requester agent* is provided with the selected recommendations.

To help in representing agents' reputations, we are making use of the Governance Framework (Silva et al., 2007), which is based on testimonies provided by witnesses about facts or events that they know are related to norms that have been violated. Since

agents know the application's norms (laws), they can judge whether an agent violated a norm. Besides, it is possible to attribute a reputation to each agent of the system (details in Section 3.2).
Before a *Requester* requests some recommendation from the DRP-MAS agents, the possible plans that the *Recommendation agent* can recommend to an agent must be defined by the application. Not only the plans themselves but also their related data (such as resources to be used in the execution and the roles of the agents with whom the agent executing the plan will need to interact) must be provided. Such plans are stored in a plan base that the *Recommendation agents* can access in order to perform the advices.

3.2 Architecture

In this section we describe the architecture of our approach that is composed for two layers: DRP-MAS and Reputation. The DRP-MAS is composed of four models: Mediation, Diagnosis, Recommendation and Artificial Intelligence Toolset.

The mediation module is responsible for providing the *Mediator agent*, which creates a *Diagnostic agent* and a *Recommendation agent* for a *Requester agent* defined in the Application, as described in Sub-Section 3.1. The Diagnosis module performs the process of diagnosis, while the Recommendation module aims to provide recommendations to achieve some desired goal. The Artificial Intelligence Toolset module defines an API (Application Public Interface) interface called BIGUS (Bigus, 2001), which allows using different kinds of reasoning algorithms to perform the processes mentioned: forward chaining, backward chaining and fuzzy logic.

The Reputation layer supplies reputations to the DRP-MAS and can also supply them to the Application layer, when requested. In the current implementation, we are using the Governance Framework to implement the Reputation layer. The framework defines three modules: judgment, reputation and punishment. The judgment module is responsible for receiving the testimonies and for providing a verdict to the punishment module, i.e., for verifying whether an agent violated a norm. The module can make use of different strategies to judge the violation of the different norms. Such strategies may use the reputation module to help in providing the decision about the violations.

The reputation module is responsible for calculating the reputation of the agents and provides them to the judgment module and to other application agents. The reputations are updated based on the testimonies provided by the judgment module about violated norms. This module already offers calculations to provide the reputations. In addition, the instances of the framework can define new calculations. The final module, punishment, is responsible for determining the penalties applied to agents that have violated the norms of the environment.

For better comprehension of the DRP-MAS framework, two key concepts are elaborated as follows: how to perform diagnoses and how to provide recommendations.

3.3 Performing Diagnoses

As was already explained in Sub-Section 3.1, the diagnosis is performed by the *Diagnostic agent* offered by the proposed framework. Such analyses are performed based on a set of information provided by the *Requester agent*. The set is used in different diagnosing processes and in different forms of recommending alternative ways to achieve some goal. The information provided in the set encompasses:

1. Resources and associated problems - In (Horling et al., 2000) it is defined that resources are important data to support diagnoses. In some situations the reason why an execution cannot be successfully performed could be the absence of some resource, or perhaps an insufficient amount of resources used to perform something. Therefore, the diagnostic agent should receive information about the used resources (their identification) and the amount used.
2. Norm violated - The violation of a norm could be a reason for not achieving a goal. Thus, the diagnosis may depend on the norm violated. Note that a norm can be considered a law that must be followed by one or more agents. Some data are provided about the norm violated: (i) the agent responsible for the failure and (ii) a value that represents the importance of said violation from the agent's point of view, called degree of violation.
3. Quality of service – The quality of service should be defined based on the TAEMS model (Horling et al., 2000) (Horling et al., 2007). The model represents a goal/task language that provides an explicit representation for goals and the available sub-goal pathways that are able to achieve them from methods (plans). Each branch in the tree can have an expected quality based on the execution of the plans. Therefore,

to verify whether a goal was achieved, the quality of the execution of the plan must be at least equal to the minimum acceptable degree. In order to represent this idea, each plan contains the following data: (i) maximum degree of quality related to the execution of the plan; (ii) minimum acceptable degree of quality to achieve the goal; and (iii) the degree attributed after the execution of the plan.

4. Goal - The execution of an agent plan is always associated with a goal that the agent wants to achieve (Silva et al., 2003). To know the goal that the agent is trying to achieve it is fundamental not only to provide a diagnosis but also to make advices. The advices about other plans to be executed will be provided considering the goal the agent was trying to achieve
5. Plan executed - To know the plan executed by the *Requester agent* it is important to understand the reason of the failure and to provide alternative execution to achieve the goal that was not achieved.
6. Agents with whom the agent interacted - The diagnosis can indicate that an agent is guilty of having provoked the failure on the execution of a plan. For this reason, it is important to know the agents with whom the *Requester* has interacted during the execution of such plan.
7. Roles - The roles played by the agents that have interacted with the *Requester* can be important to update reputations, and to serve as recommendations for other agents that play the same roles.
8. Profile - Agents can have profiles that represent some of their characteristics. A profile can, for instance, stipulate the minimum acceptable degree of reputation of the agents that provide information to the *Requester agent*. This information can be useful in the process of providing recommendations, especially when there is a need to advise another partner to interact with the *Requester*.
9. Problems met by the Requester - The information that can be provided by the *Requester* is not limited to the set mentioned previously. The *Requester* can also send domain-dependent information that will be used by the domain-dependent strategies to perform diagnoses and providing recommendations.

The DRP-MAS framework defines the performance of diagnoses as a hot spot (or flexible point) (Fayad et al., 1999) that can be implemented by applications to provide domain-dependent strategies. Therefore, different applications can define different strategies to deal with the domain-independent and domain-dependent sets of information provided by the *Requester agent*. Nevertheless, the framework makes available a default strategy to provide diagnosis based on the domain-independent information set.

In order to illustrate a situation when some failure of a goal happens, let's focus on the domain of making coffee. An agent has a goal to make coffee for its friends and to achieve this goal it executes a specific plan. Suppose that the agent noticed that the coffee is not good but does not know why. There are several reasons leading to making bad coffee: the quality of the coffee powder is poor, the water used was cold, the quantities of coffee and water were not adequate, etc. To find out what has happened, the *Requester agent* should send to the *Diagnosis agent* information about its goal (to make coffee for three persons), the plan it has executed to make the coffee, the coffee itself, the quantity of water, the temperature of the water and the quality and the description of the coffee powder used. The *Diagnosis agent* must know how much water and coffee powders are required to make a cup of coffee, the ideal temperature of the water and which coffee powders are good. One possible simple strategy combines the information the agent has received with the beliefs of the *Diagnosis agent* related to make coffee. The more information the strategy receives, the more precise the diagnosis will be.

To help defining domain-dependent strategies three different algorithms (backward chaining, forward chaining and fuzzy logic) are available in the Artificial toolset module defined in the framework. The strategies can use the BIGUS API to access such algorithms. In Section 6 an example of a strategy that used the forward chaining algorithm is presented.

3.4 Providing Recommendations

The *Recommendation agent* incorporates the process of advising alternative ways to achieve a goal. The process is composed of three steps: to select plans, to verify if the plan requires that agents request information, and to choose good agents.

The first step is executed when the *Recommendation agent* receives the diagnosis from the *Diagnostic agent*. It first verifies which plans can be used to achieve the desired goal. Second, the *Recommendation agent* uses the diagnosis and the information sent by the *Requester agent* to select a plan. If no plan is encountered, then a message is

sent to the *Requester*. Otherwise, the second step is executed.

The second step verifies if the selected plan needs the assistance of agents in order to request information. If it is not necessary, then the process is concluded and a message with the recommended plan is sent. Otherwise, the reputations of the agents are requested using the reputation module offered by the Governance Framework. The third step is executed after receiving all reputations (control performed by the DRP-MAS). In the third step, the *Recommendation agent* selects the agents to be used by the chosen plans according to their reputations. At the end, the selected plans and agents are provided to the *Requester agent*.

3.4.1 Selecting Plan

The step *Selecting Plan* is responsible for choosing alternative plans to achieve the desired goal. This task is a domain-dependent one since the selection of a plan may depend on the domain-dependent information provided by the *Requester agent*. Therefore, each application that uses the DRP-MAS can define its own strategy to select plans. The task for selecting plans is defined in the DRP-MAS as a flexible point that should be extended by the application.

The application should provide the possible plans that the agents can execute and the expected configuration that each one has. These expected configurations are available in a plan base that can be accessed by the application's *Recommendation agents*. Each plan can have the following data associated with it: resources used during the execution, desired goal, profiles of agents that accept executing the plan, quality of service that determines how the previous execution of the plan was performed, related diagnoses, roles played by agents in the execution of the plan, and a collection of possible problems that the plan can resolve. Note that the set of data used to configure a plan is the same set that comprises the information described in Sub-Section 3.3, i.e., the information provided by the *Requester agent*.

Although the selection of a plan may be domain-dependent, the framework provides a default strategy for selecting a plan based on the domain-independent set of information that the *Requester agent* can provide. As a default strategy, the framework provides plans that achieve the same desired goal, excluding the plan used by the *Requester agent*.

3.4.2 Verifying Selected Plans

After the selection of the alternative plans, it is verified whether some plan needs to interact with other agents. To perform this analysis, each plan must have been associated with a list of the roles to be played by agents with whom the agent executing the plan may interact. If the list of roles in a plan is empty, it means that no communication is needed between agents while executing the plan. In the case the lists of all plans are empty, a message can be sent to the *Requester agent* with the recommended plans. Otherwise, the *Recommendation agent* must decide which agents should be used in the interactions. This decision is based on the reputations of agents that will be selected as partners to play the roles. The *Recommendation agent* requests the reputations of all agents that can play the roles identified in the plans from the Governance Framework. Although we propose the use of the Governance Framework to provide the agents' reputation, any other approach that is able to provide the reputation of agents while playing a role can be used. After receiving all reputations, the third step is executed.

3.4.3 Choosing Agents

As in the two previous steps, the strategy in this step is also a flexible point of the framework and different kinds of strategies can be used. However, the framework offers a default strategy that selects the agents based on the minimum acceptable reputation defined in the *Requester agent* profile.

Note that a profile can specify other information that can also be useful when choosing agents. Multi-agent systems can have heterogeneous agents with different behaviors and characteristics that can define several different profiles. We stimulate the use of profiles to help on deciding about which plans should be executed, and which agents the *Requesters* accept to interact with.

Consider the application of buying and selling goods to understand how the profiles of agents can influence the selection of agents. If a buyer desires to buy a given product, it can determine that it will negotiate only with sellers that have good reputations. Therefore, the buyer can determine a minimum reputation in order to select the acceptable sellers to future transactions.

After the selection of the advised agents, a message is sent to the *Requester agent* with the recommendations. For each plan, the possible agents and the resources to be used are defined.

4 THE USE OF GOVERNANCE FRAMEWORK

As mentioned in Section 3.1, the DRP-MAS uses the Governance Framework in order to represent reputations. The judgment module is used to update reputations, while the reputation module is used to request agents' reputations These situations are better explained as follows.

1. Updating reputations - To change the reputation of selected agents, the *Diagnostic agent* of the DRP-MAS can send testimonies to the judgment module. The testimonies point out, according to the information in the diagnoses, the agents that have violated norms. Since the testimonies provided by such agents are always truth testimonies, the judgment module does not judge them. For this reason, when the judgment module receives the testimonies, the reputations of the accused agents are automatically modified by the reputation module.
2. Using reputations - When the *Recommendation agent* needs to meet agents to provide information about a plan, it requests the reputation of the selected candidates from the Reputation Module. When the *Recommendation agent* receives all reputations requested, it performs the analyses and decides which agents are good or bad from the negotiation point of view.

5 INTELLIGENT HOME

The example used to instantiate the DRP-MAS framework is the intelligent home that also is used in (Horling et al., 2000). From a set of possible cases about the intelligent home, two were chosen for illustration: dishwasher and to make coffee.

5.1 Dishwasher

In one of the analyzed scenarios of the intelligent home, an agent representing a dishwasher receives hot water from another agent representing a water heater. The water heater is also able to provide hot water to the shower of a person. Suppose that while the dishwasher is on, a person starts to take a shower. Since the dishwasher needs hot water to work properly, the dishwasher should adapt its behavior by choosing one of the following options: (i) to wait for the water heater to be free and provide hot water again, (ii) to search for another available water heater, if any, or (iii) to wash the dishes with cold water. However, in this latter situation the agent does not achieve the desired goal since the dishes are not properly washed.

Let us suppose that the dishwasher has chosen the latter option because there no water heaters are available and that it is programmed to save energy. When the dishwasher finishes its work, it notices that the dishes are not properly washed. When this happens, the dishwasher agent decides to request a *Diagnostic agent* from the *Mediator agent*. In sequel, the *Requester agent* (dishwasher) provides six different pieces of data to the *Diagnosis agent* about the execution performed: (i) the quality of service on the plan, (ii) its profile, (iii) the norm violated, (iv) the agent used during the execution, (v) the role played by such agent and (vi) the temperature of the water used. The first five pieces of information are pre-defined by the framework (Section 5.1), and the sixth is defined by the application. On the profile of the agent it is informed that only agents with reputations higher than 0.8 can provide information to the *Requester*.

When the *Diagnostic agent* receives the message supplied by the *Requester*, it begins to perform the diagnosis. To perform it, we chose to use the well-known Forward Chaining algorithm offered by the Artificial Intelligence Toolset module of the framework. This algorithm uses inference rules from a set of available data in order to extract more data while seeking an optimal goal. Therefore, we had to create a rule base with all the possible rules that allow meeting the diagnoses from the data provided by the *Requester agent*. These rules are shown in Figure 2.

The rule base uses six attributes: quality_service, violated_norm, role_agent_used, temperature_water, conclusion and problem. The values of the first four attributes are provided by the *Requester agent*, while the values of the two last ones are automatically attributed by the forward chaining algorithm, which tries to infer new data (conclusion and problem) from available ones (quality_service, violated_norm, role_agent_used and temperature_water). The quality_service attribute represents the quality of the execution performed by the plan. The violated_norm informs the norms violated during the execution of the plan. The role_agent_used informs the role played by the partner agent (water heater) that interacted with the *Requester agent*, and the temperature_water attribute informs the temperature of the water used to wash the dishes. The conclusion attribute will inform if the dishwasher used hot water during the washer, and the problem attribute

will point out the final diagnosis. The rules presented in the rule base lead to only two possible diagnoses: the dishwasher did not succeed in washing the dishes because the communication with the water heater has failed, or some unknown problem occurred.

Let's suppose that the data provided by the *Requester agent* were: quality_service=5, norm= to_wash_dishes_with_hot_water, role_agent_ used=water_heater and temperature_water=30. After applying the rules, the attributes *conclusion* and *problem* receive the data without_hot_water and problem_communication_waterheater, respectively, indicating that the water heater did not provide hot water correctly.

After meeting the desired diagnosis, a message with the diagnosis is sent to the *Recommendation agent*, in order to search for alternative executions to the *Requester agent*. The *Recommendation agent* analyzes the diagnosis and concludes that it is necessary to select another agent to provide hot water. The conclusion is that the selected plans need to request hot water from a water heater agent. For this reason, the *Recommendation agent* requests the reputation of the water heaters, and therefore decides which of them have reputations higher than 0.8 (defined in the profile supplied by the *Requester*). After the analysis process, the selected agents and plans are sent to the *Requester agent.*

Problem_Communication_WaterHeater:
IF conclusion=without_hot_water AND
violated_norm=to_wash_dishes_with_hot_water
AND role_agent_used=water_heater AND
quality_service <10 THEN
problem= problem_communication_waterheater
Problem_Unknown_in_the_Plan:
IF conclusion=com_agua_quente AND
quality_service <10 THEN
problem= problem_unknown_in_the_plan
With_Hot_Water:
IF temperature_water>39 THEN
conclusion=with_hot_water
Without_Hot_Water:
IF temperature_water<40 THEN
conclusion=without_hot_water

Figure 2: Rule base of the domain Dishwasher.

5.2 To Make Coffee

Another scenario chosen was the coffee maker, whose goal is to make 20 cups of strong coffee. While an agent represents the coffee maker, another one represents a tester, which is responsible for testing whether the coffee was made correctly. Initially, the coffee maker executes a plan to make the coffee. When the coffee is ready, a message is sent to a *Tester agent*. It analyzes the coffee and sends a response message informing that the coffee is not good. For this reason, the coffee maker decides to request recommendations from the Analysis Module.

The first step performed by the coffee maker is to request a *Diagnostic agent*, and then to request the recommendations, informing some data about its execution: quality of the execution of the plan (provided by the framework), amount of the water and amount of coffee powder used (provided by the instance), which are the resources used by the plan.

As in the case of the dishwasher presented previously, we have also used the Forward Chaining algorithm to make the diagnoses. Part of the rule base defined in this example is shown in Figure 3. Six data were defined: *amount_water*, *amount_powder*, *quality_service*, *conclusion_ coffee, conclusion_cups* and *problem*. The first data represents the amount of water used by the plan to make the 20 cups of coffee. The second data is the quantity of coffee powder used, while the *quality_service* represents an assigned degree to the execution of the plan. If the value attributed is lower than 10, then some problem occurred during the execution. Another data used was the *conclusion_coffee* that informs whether the coffee that was made used too little or too much coffee powder, while the *conclusion_cups* verifies whether the correct amount of water was used to make 20 cups. The *problem* attribute will represent the diagnosis.

Let's suppose that the *Requester agent* (coffee maker) provided the following data: quality_service=0, amount_water=600 (mL), and amount_powder=20 (grams). As the goal of the coffee maker is to make 20 cups of strong coffee, we can see that applying these values in the rule base, the problem met is *problem_amount_powder_and_cups*. In other words, the amount of powder and the amount of water were incorrect to make 20 cups of strong coffee. For this reason, the quality of service came with a value lower than 10.

Problem_Strong_Coffee _20_Cups:
IF conclusion_coffee= weak_coffee AND
conclusion_water = Coffee_Incorrect_Water
AND quality_service <10 THEN
problem= problem_amount_powder_and_cups
Weak_Coffee:
IF amount_powder <30 THEN
conclusion_coffee= weak_coffee
Strong_Coffee:
IF amount_powder >29 THEN
conclusion_coffee= strong_coffee
Coffee_Correct_Water:
IF amount_water=1000 THEN
Conclusion_water= **Coffee_Correct_Water**
Coffee_Incorrect_Water:
IF amount_water!=1000 THEN
Conclusion_water= **Coffee_Incorrect_Water**

Figure 3: Rule base for the domain of coffee making.

After reaching the diagnosis, a message is sent to the *Recommendation agent*. The strategy adopted in this case was to verify the amount of necessary resources to make 20 cups of strong coffee, and to search other plans with the same goal. With the selected plans, the correct quantity of powder and water to make the coffee is informed to the plans. Later, it is verified that the plans need a *Tester agent* to test the coffee. As there is only one available tester (defined by the application), this one is chosen. Finally, the recommendations are provided to the coffee maker (the *Requester agent*).

6 RELATED WORK

In this section, we describe some related work and make a comparison with the proposed DRP-MAS. In particular, we consider works reported in (Li et al., 2004), (Horling et al., 2000) and (Roos et al., 2002).

6.1 Application of MAS in Control and Fault Diagnosis Systems

In (Li et al., 2004), a decentralized system is proposed in order to perform diagnosis and monitoring. Each component has a monitor (Monitoring Agent), which is responsible for collecting information about it. When obtained, the data are provided to agents offered by the proposed system, which are responsible for working together in order to find the diagnoses.
One of the drawbacks of this approach is that it violates the privacy of the agents. For this reason, the DRP-MAS does not create monitors, but waits for the agents of the application to request for diagnoses.

6.2 Diagnosis as Part of Adaptability

The authors in (Horling et al., 2000) examine the use of domain-independent diagnoses in multi-agent systems. They argue that the initial step is to make available information describing the correct, or at least expected, behavior of agents. They state that useful method execution and goal achievement information can be succinctly encoded in a domain-independent way with a goal/task decomposition language called TAEMS.

In the DRF-MAS the methods defined by the TAEMS in order to achieve the desired goals are represented by plans that are used to attain goals. Each plan defines a set of possible related information, such as resources used and their expected amount, desired goal, expected quality, etc., as described in Section 3.3. If a plan has a problem, it is possible to verify the causes of the failure.

Comparing with (Horling et al., 2000), our approach offers a bigger information set, making it possible to perform more and different diagnoses. Another distinguishing characteristic of the DRP-MAS is the use of agents' reputations, which helps during the selection of future partners.

6.3 An Analysis of MAS Diagnosis

In (Roos et al., 2002), the authors define an information set to be used by a global system to provide diagnoses. This set is:

$$S = (C, M, Id, Sd, Ctx, Obs)$$

where C is a set of components, M is a specification of possible fault per component, Id is a set of identifiers of points that connect components, Sd is the description of the system, the Ctx is a specification of input values of the system that are determined outside the system by the environment, and Obs is a set of observed values of the system. DRP-MAS follows a similar idea by extracting the necessary information to perform diagnoses from the set of information used by the process of diagnosis presented in Sub-Section 3.3.

7 CONCLUSIONS

In the present paper we have outlined the main challenges and associated requirements as well as

the design strategy to create a hybrid diagnostic-recommendation system for agent execution in open multi-agent systems. This system helps to perform diagnoses and to recommend alternative ways for executions to achieve goals. The intelligent home domain was presented as a case study to illustrate the applicability of our approach.

Two important lessons were learned in the process of analyzing and developing the proposed system. The first lesson relates to the diagnosis process. We have realized that to define a universally efficient solution to perform diagnoses in different domains is very difficult, because some domains have particular characteristics that influence the result of the diagnoses.

The second lesson relates to the use of the reputation concept. Depending on the situation, to adequately select the agents that will be used to request some information can be important, because some provided information can determine the success or failure of some execution.

Our plan for future work is to focus on case studies involving ubiquitous computing, because it can present complex situations where we can perform diagnoses and provide recommendations.

REFERENCES

Bigus, J., Bigus, J., 2001. Constructing Intelligent Agents Using Java, 2nd edition.

Boella, G., Torre, L., 2004. Regulative and Constitutive Norms in Normative Multi-Agent Systems. *In Proc. of 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning*. California.

Fayad, M., Johnson, R., Schmidt, D., 1999. Building Application Frameworks: Object-Oriented Foundations of Framework Design (Hardcover), Wiley publisher. 1st edition.

Horling, B., Lesser, V., Vincent, R., Bazzan, A. Xuan, P., 2000. Diagnosis as an Integral Part of Multi-Agent Adaptability*, DARPA Information Survivability Conference and Exposition, DISCEX'00*, Proceedings, Volume 2, pp. 211-219.

Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., Garvey, A. 1999. The TAEMS White Paper. http://dis.cs.umass.edu/research/taems/white/. Last access in November, 2007.

Jennings, N., Wooldridge, M., 1999. Agent- Oriented Software Engineering. *In Proc. of the 9th European Workshop on Modeling Autonomous Agents in a Multi-Agent World*: *Multi-Agent System Engineering* Vol. 1647, Springer-Verlag, pp. 1-7.

Li, T., Peng, Y., Zhao, H., Li, K., 2004. Application of Multi-Agent in Control and Fault Diagnosis Systems. *In Proc. of the Third Int. Conf. on Machine Learning and Cybernetics*, Shanghai, pp. 26-29.

Roos, N., Teije, A., Bos, A., Witteveen, C., 2002. An Analysis of Multi-Agent Diagnosis, *AAMAS'02*.

Silva, V., Cortês, M., Lucena, C., 2004. An Object-Oriented Framework for Implementing Agent Societies, MCC32/04. Technical Report, Computer Science Department, PUC-Rio. Rio de Janeiro, Brazil.

Silva, V.; Duran, F.; Guedes, J., Lucena, C., 2007. Governing Multi-Agent Systems, *In Journal of Brazilian Computer Society*, *special issue on Software Engineering for Multi-Agent Systems*, n. 2 vol. 13, pp. 19-34.

Silva, V.; Garcia, A.; Brandao, A.; Chavez, C.; Lucena, C.; Alencar, P., 2003. Taming Agents and Objects in Software Engineering" In: Garcia, A.; Lucena, C.; Zamboneli, F.; Omicini, A; Castro, J. (Eds.), Software Engineering for Large-Scale Multi-Agent Systems, Springer-Verlag, LNCS 2603, pp. 1-26.

Vicent, R., Horling, B., 2000; Experiences in Simulating Multi-Agent Systems Using TAEMS, *Proceedings Fourth International Conference on MultiAgent Systems*, Volume, Issue, pp. 455-456.