# Diagnosis as an Integral Part of Multi-Agent Adaptability *

**Bryan Horling, Victor Lesser, Régis Vincent, Ana Bazzan, Ping Xuan**
Department of Computer Science
University of Massachusetts

UMass Computer Science Technical Report 99-03

**Abstract**

Agents working under real world conditions may face an environment capable of changing rapidly from one moment to the next, either through perceived faults, unexpected interactions or adversarial intrusions. To gracefully and efficiently handle such situations, the members of a multi-agent system must be able to adapt, either by evolving internal structures and behavior or repairing or isolating those external influenced believed to be malfunctioning. The first step in achieving adaptability is diagnosis - being able to accurately detect and determine the cause of a fault based on its symptoms. In this paper we examine how domain independent diagnosis plays a role in multi-agent systems, including the information required to support and produce diagnoses. Particular attention is paid to coordination based diagnosis directed by a causal model. Several examples are described in the context of an Intelligent Home environment, and the issue of diagnostic sensitivity versus efficiency is addressed.

**Content Areas: Intelligent Agents : Tasks or Problems : Multi-Agent Communication or Coordination or Collaboration; Intelligent Agents : Tasks or Problems : Learning and Adaptation**

## Overview and Motivation

Designing systems utilizing a multi-agent paradigm offers several advantages. They can easily utilize distributed resources, work towards multiple goals in parallel, and reduce the risk of a single point of failure. One common failing of multi-agent systems, however, is brittleness. In the face of an adverse or changing environment, assumptions which designers commonly make during the construction of these complex networks of interacting processes may become incorrect. For instance, the designer may assume a certain transfer rate over the local network, or that a particular method may produce results of a given quality, both of which can easily change over time. In today's networked environment, it is also quite possible for adversarial intrusions to occur, capable of producing an even wider range of symptoms. Failure to act on incorrect assumptions can lead to degraded performance, incorrect results, or in the worst case even bring the entire system to a halt.

One solution is for the designer to be extremely paranoid, and essentially over-engineer each aspect of the system to reduce the chance of failure (e.g. always over-coordinate, execute redundant plans). Although such precautions certainly have their place in computing systems, they can also increase overhead to undesirable levels. This overhead then decreases the overall level of efficiency, as the system essentially wastes effort avoiding failures which would not have occurred in any case.

To improve the robustness of multi-agent systems, without unduly sacrificing efficiency, the system should be designed with adaptability in mind. An agent working under changeable conditions must have knowledge of its expected surroundings, goals and behavior - and the capacity to generate new, situation-specific coordination plans when these expectations are not met. The key to initiating adaptive behavior, therefore, is detecting such failed expectations, and determining their cause so they may be corrected. We believe that this need can be satisfied, and thus adaptability promoted, by incorporating domain and coordination independent diagnosis capabilities into the individual agents within the multi-agent system.

In this paper we will discuss how diagnosis plays a role in the adaptability of an intelligent home multi-agent system (Lesser *et al.* 1999). In this environment, major appliances, such as the dishwasher, waterheater, air conditioner, etc., are each controlled by an individual autonomous agent. The intelligent home provides

* Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number F30602-97-1-0249 and by the National Science Foundation under Grant number IIS-9812755 and number IRI-9523419. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, National Science Foundation, or the U.S. Government.

us with interesting working conditions, allowing the creation of scenarios involving constrained resources, conflicting objectives and cooperative interactions. We believe the issues raised by the intelligent home model can be sufficiently generalized to apply to multi-agent systems operating in other domains.

To overview the role diagnosis may play in a multi-agent system, consider the following scenario. A dishwasher and waterheater exist in the house, related by the fact that the dishwasher uses the hot water the waterheater produces. Under normal circumstances, the dishwasher assumes sufficient water will be available for it to operate, since the waterheater will attempt to maintain a consistent level in the tank at all times. Because of this assumption and the desire to reduce unnecessary network activity, coordination is normally unnecessary between the two agents. Consider next what happens when this assumption fails, perhaps when the owner decides to take a shower at a conflicting time (i.e. there is an assumption that showers only take place in the morning), or if the waterheater is put into "conservation mode" and thus only produces hot water when requested to do so. The dishwasher will no longer have sufficient resources to perform its task. Lacking diagnosis or monitoring, the dishwasher could repeatedly progress as normal but do a poor job of dish-washing, or do no washing at all because of insufficient amounts of hot water. Using diagnosis, however, the dishwasher could, as a result of poor performance observed through internal sensors or user feedback, first determine that a required resource is missing, and then that the resource was not being coordinated over. By itself, this would be sufficient to produce a preliminary diagnosis the dishwasher could act upon simply by invoking a resource coordination protocol. After reviewing its assumptions, later experiences or interactions with the waterheater, it could refine and validate this diagnosis and perhaps update its assumptions to note that hot water should now be coordinated over, or that there are certain times during the day when coordination is recommended. It should be clear that even simple diagnostics capabilities can provide a fair measure of robustness under some circumstances.

This work represents the continuation of our previous work on diagnosis for single agent (Hudlická & Lesser 1987) and multi-agent systems (Hudlická *et al.* 1986; Bazzan, Lesser, & Xuan 1998; Sugawara & Lesser 1993; 1998). The emphasis of this new work is to show that diagnosis can be exploited for a wider set of issues than indicated in our earlier work and more importantly that this diagnosis can be done in a domain-independent manner. The use of diagnosis is also a recent theme in the work by Kaminka and Tambe (Kaminka & Tambe 1998). Our work differs in the set of issues we are interested in diagnosing. For example, in Kaminka, collaborative agents use one another reflectively based on plan recognition, as sources of comparative information to detect aberrant behavior due to, for example, inconsistent sensing by different agents of environmental conditions. Our work in contrast is mainly concerned with the performance issues surrounding the use of situation-specific coordination strategies. In this view of coordination, the strategy used for agent coordination must be tailored to specifics of the current/expected environment and the coordination situations that an agent will encounter. Over or under coordination can be harmful, respectively, because of the expenditure of resources to implement a coordination strategy that doesn't contribute sufficiently to a more efficient use of resources or the lack of appropriate coordination that leads to suboptimal use of resources. Implicit in this discussion is that there are a set of assumptions about agents behaviors and availability of resources that is the basis for effective situation specific coordination. Detection in this case involves recognizing when a monitored performance based assumption is no longer valid; diagnosis is then taking this triggering symptom and determining the detail set of assumptions about agent and resource behavior that is responsible for the symptom in the current context.

Several facets of agent diagnosis will be covered in this paper by detailing a diagnosis system that we have implemented in the Intelligent Home environment. What sort of information is needed? How can the diagnosis be produced? How does the diagnosis affect adaptability? These questions will be addressed using the diagnosis framework we have created, which focuses on the resources, coordination and activities associated with the agent. In the following section, the use of assumptions and organizational knowledge, along with fault detection techniques, will be discussed. Our diagnosis-generating framework, incorporating and utilizing this information, will then be introduced. Several examples detailing diagnosis in the Intelligent Home will provide further motivation and functional details about our system. We will then provide a more quantitative analysis of detection and diagnosis sensitivity tradeoffs, and conclude with directions for future research.

## Information Requirements

For diagnosis to function, some sort of knowledge about expected behavior must be known a priori, to serve as a basis for comparison. The exact information which is needed depends on the diagnosis capabilities needed by the agent, but the data typically fits into one of three categories: knowledge about the agent's expected operational behavior, including environmental assumptions; methods for detecting deviations from those expectations; and faculties for diagnosing these deviations when they are found.

### Expectations and Assumptions

For diagnosis to function, some information must be available describing what the correct, or at least expected, behavior of the agent should be. We have found that a great deal of useful method execution and goal

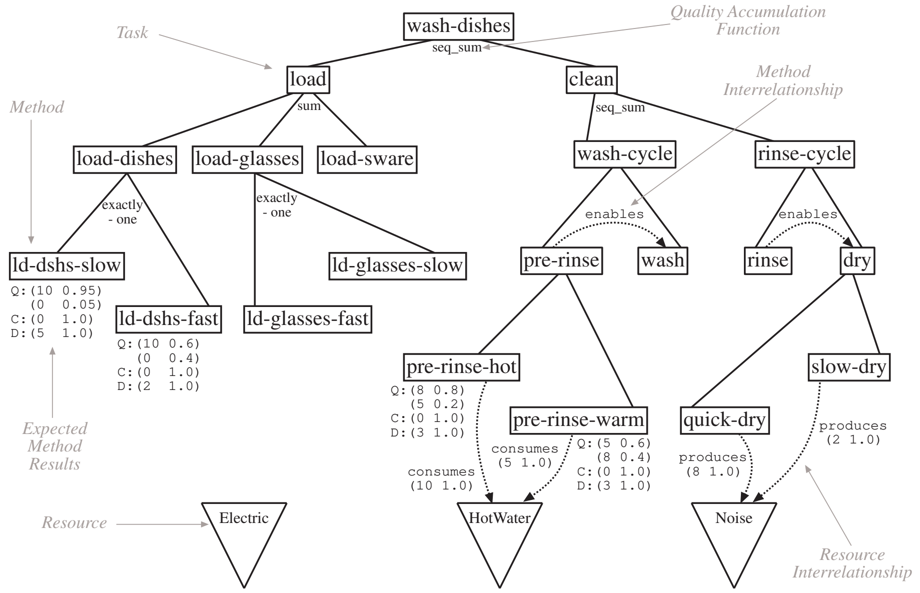


Figure 1: Abbreviated TÆMS task structure example for the dishwasher agent

achievement information can be succinctly encoded in a domain-independent way with a goal/task decomposition language called TÆMS (Decker & Lesser 1993). A graphical example of the dishwasher's TÆMS task structure can be seen in Figure 1. TÆMS provides an explicit representation for goals, and the available subgoal pathways able to achieve them; each branch in the tree terminates at an executable method. Expected method behavior and interactions between other methods and resources may also be represented explicitly. Associated with each method is a distribution-based description of its expected quality, cost and duration measures. These descriptions are represented by the Expected Method Results in the figure, where each possible method result along each trait consists of expected value-probability pair. Similarly, the effects of hard ($A$ enables or disables $B$), soft ($A$ facilitates or hinders $B$) and resource ($A$ produces or consumes resource $B$) interrelationships are also quantitatively described with distributions.

In our discussion so far we have not focused on the underlying coordination architecture. Our initial thinking was that diagnosis needed to be strongly tied to the specifics of this architecture, so initial efforts (Bazzan, Lesser, & Xuan 1998) were thus bound to the GPGP coordination architecture (Decker & Lesser 1995). However, our stance has changed on this matter. We now feel that so long as TÆMS is a faithful representation of the expected local agent behavior and its interaction with other agents and resources, and that information is provided about what aspects of TÆMS were used in the current coordination context, and what was the actual schedule and results of activities executed, diagnosis can be done largely in a domain and coordination independent structure.

Another set of information, describing pertinent external assumptions, is needed for the diagnosis system to reason about the agent's interactions with its environment. Such characteristics currently used in our agents include network behavior (e.g. bandwidth, latency), entities thought to exist external to the agent (e.g. entities one might interact with), and resource characteristics (e.g. low/high bound, usage patterns), each of which has a direct correlation with how the agent should coordinate with other agents in the system. Organizational knowledge, the information an agent has about where and how it fits into its society, is a subset of this category. It may be useful, for instance, for the agent to have a record of the types of agents it is expected to interact with, and what sort of interactions should take place. In the the introductory example, this sort information let the dishwasher know that there was a prior assumption that coordination over hot water was unnecessary.

## Detecting Possible Failures

Once descriptive information and models are incorporated into the agent, the process of using that information to detect possible inconsistencies becomes relevant. Consider the simple case of detecting abnormal method results. Within the TÆMS structure, the expected cost, quality and duration outcomes are described for each method. Armed with this information, the diagnosis system can use a light-weight comparison monitor to determine when a deviation from expected behavior might have occurred. The dishwasher used the method-resource relationship description in its TÆMS task structure, for instance, to determine that hot water was necessary for its dish-washing task to successfully complete. Performance threshold assumptions can

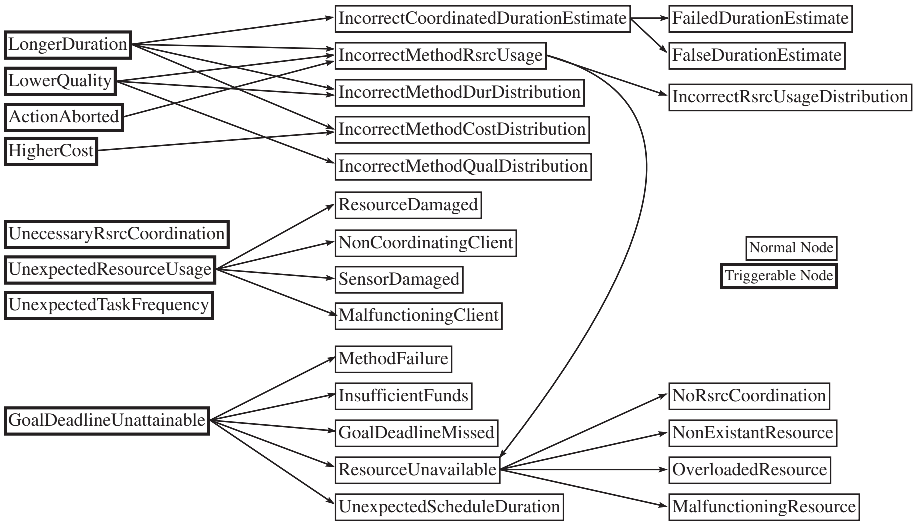


Figure 2: Example causal model structure for diagnosis in the Intelligent Home

then be used to determine the severity of the deviation, which would help determine the correct diagnosis and response later on. On-line learning of models can also be used to track long term trends in behavior, which can help determine if a deviation is caused by a fundamental shift in the environment or is just a one-time aberration.

Using knowledge about method interactions, also found in the TÆMS model, the diagnosis system can determine if a failure might have been brought about by some other method which had or had not successfully executed. If the method's source is local, an activity log can be checked for more evidence. If the offending method's source is remote, the list of known external agents can then be used to track down the culprit. A mixture of model based comparisons, combined with directed evidence gathering, provides a good base for detecting possible failures.

## Performing the Diagnosis

The agent can now use its expectation information and failure detection methods to begin the actual process of diagnosis. Diagnosis is a well-researched field, with many different methods and techniques already available to the system designer (W. Hamscher 1992). Our goal was to use a technique that offered great flexibility in the information it could use and the diagnoses it could generate, without sacrificing subject scope or domain independence. Recent work in the field of diagnosis (Kaminka & Tambe 1998; Toyama & Hager 1997) has shown that viable new technologies are still being developed. It is not clear, however, that any single diagnostic technique is suitable for the entire range of faults exhibited by multi-agent systems. It was therefore desirable to use a system or framework capable of incorporating different diagnostic techniques, i.e. make use of different specific methods for the types of failures they best address.

Expanding on work first researched in (Sugawara & Lesser 1993), we chose to organize our diagnostic process using a causal model. The causal model is a directed, acyclic graph organizing a set of diagnosis nodes. Figure 2 shows such a graph, constructed to address issues brought up by examples in this paper, which we have implemented for the Intelligent Home agents; complete graphs addressing broader topics can be found in (Bazzan, Lesser, & Xuan 1998). Each node in the graph corresponds with a particular diagnosis, with varying levels of precision and complexity. As a node produces a diagnosis, the causal model can be used to determine what other, typically more detailed, diagnoses can be used to further categorize the problem. Within the diagnosis system, the causal model then acts as a sort of road map, allowing diagnosis to progress from easily detectable symptoms to more precise diagnostic hypotheses as needed.

The causal model in Figure 2 focuses on coordination, behavior and resource issues. Within the diagram, several nodes have bold-faced outlines. These nodes are *triggerable*, meaning they periodically perform simple comparison checks to determine if a fault may have occurred. This trigger-checking activity is a primary mechanism for initiating the diagnostic process. The runtime usage of the causal model is shown when considering the diagnostic activity of the agent in the introductory example. In the example, the dishwasher has performed an activity, but achieved a lower than expected amount of quality in the results. Using the given causal model, the LowerQuality node would be triggered. The resulting diagnosis would activate the child nodes of LowerQuality: IncorrectMethodRsrcUsage, IncorrectMethodDurDistribution, and IncorrectMethodQualDistribution. The first node attempts to encapsulate the idea that something went wrong with the resources expected to be used by the method, while the latter two address possible discrepancies in the method's expected duration and quality. IncorrectMethodRsrcUsage would then produce a diagnosis, activating IncorrectRsrcUsageDistribution and ResourceUnavailable. The resource was not used, so ResourceUnavailable would be triggered, activating its four child nodes. Of these, NoRsrcCoordination (possibly in combination with OverloadedResource) would then determine the exact problem.

The causal model addresses each of the design goals previously mentioned. The automatic flow of diagnoses through the graph structure allows the designer to add or remove subgraphs and nodes at will to increase or decrease the diagnostic specificity offered by the model. Thus, although the model shown in this paper is targeted towards a specific set of faults, the causal model in general allows one to create a diagnostic system for any range of faults or intrusions who's set of symptoms that can be observed or deduced are differentiable from one another. In our implementation, each node in the model corresponds to an individual persistent diagnostic object, capable of passively listening for data or actively gathering evidence. This means that, within the causal framework, individual nodes are free to use whichever diagnostic technique is needed, offering a great deal of flexibility to the system designer. The persistent nature of the diagnosis object also allows for direct control over the amount of evidence required for a diagnosis to be triggered and produced, since diagnostic analysis can continue through several episodes. We will come back to this notion of diagnosis sensitivity and confidence in a later section.

Faults involving multiple symptoms (or lack thereof) can be handled elegantly by the causal model, by incorporating a single node for the fault which uses diagnostic evidence from several other symptom-verifying nodes. For instance, a node diagnosing incorrect local method behavior descriptions could be derived with a diagnosis from another node detecting methods which take too long to execute, in conjunction with a lack of diagnoses from nodes diagnosing competing theories. A detected fault may also be caused by the cumulative effects of several other deviations, which did not merit diagnosis individually. This can occur, for instance, when a goal deadline is missed because each in a series of method invocations took slightly longer than expected. Individually, the methods' durations were within their respective distribution ranges; an expected deadline produced with the mean of each of these distributions could very well be missed under these circumstances.

The designer is also free to make nodes as domain independent as possible, so with a carefully thought out structure it is possible to transport the model from one system to the next with little modification. In addition to being domain independent, we also hypothesize that such a structure may be coordination independent; accurate diagnoses can be formed about coordination activities independent of the protocol's details. The fact that the framework scales so easily in scope means that a common, domain independent core may also be used among a group of agents, which augment the model with more specific nodes to meet their individual diagnostic needs.

## Example Diagnosis Generation

In this section we will go over several multi-agent scenarios from the Intelligent Home domain, and how diagnosis based adaptation plays a vital role in its successful behavior. The first example gives a complete agent-by-agent description of a malfunctioning agent scenario, while the remaining examples will sketch out the usefulness of diagnosis under other circumstances.

### Detecting Software Failure

This scenario contains three agents, each with its own set of capabilities, goals and knowledge (see Figure 3), and a set of water pipes which connect them. The central figure in the scene is the waterheater, which produces hot water at a rate dictated by requests posed by the agents it serves. The owner of the house, being quite thrifty, has set the waterheater's goal to produce the minimum amount of hot water possible, thereby minimizing cost. The waterheater will therefore keep no minimum amount of water in the tank, forcing any agent needing hot water to coordinate over its usage (i.e. to explicitly schedule the production of hot water at a specific time). Elsewhere in the house is some other hot water-using appliance, which has a generic goal it needs to complete by a certain time. It is functioning normally, coordinating with the waterheater as needed.

The third agent, a dishwasher, has as its goal to wash the load of dishes currently in its possession. It has successfully started this operation, but through some sort of malfunction has arrived in a state where it is endlessly executing the method "pre-rinse-hot", a method which uses hot water (see Figure 1). The coordination component in the dishwasher still functions however, and therefore is also repeatedly coordinating over the hot water being used by the cyclic execution. As it happens, the dishwasher has also been set with a higher priority level than the other generic appliance working towards its goal. This has the end result of overwhelming the waterheater, forcing it to reject the hot water requests of the other appliance.

Lacking diagnosis, this scenario would end in failure. The dishwasher would never complete its load, the waterheater will produce much more water than would normally be needed, and the other appliance will fail to meet its goal deadline. Each agent, however, is equipped with information and a diagnosis model similar to that shown in Figure 2. Figure 3 should give the reader some notion of the characteristics each agent has which are relevant to this situation.

The dishwasher is first to detect a problem. The UnexpectedTaskFrequency node in its causal model is triggered, which proceeds to gather evidence on the current situation. Using the local activity log, schedule,

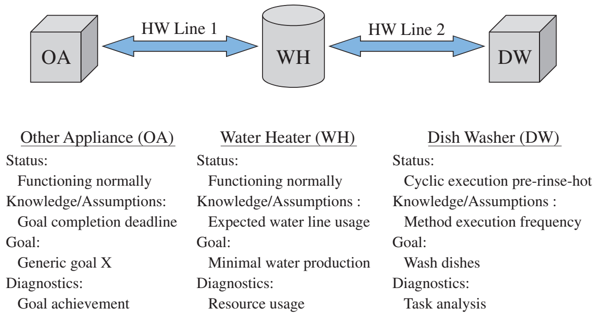


Figure 3: Software failure scenario overview

and assumptions about expected task frequency, it produces a diagnosis with a high confidence value. For this example, we will assume the dishwasher has no repair mechanisms for this problem, so no further actions are taken.

Eventually, the diagnosis component at the waterheater is also triggered. A root node UnexpectedResourceUsage is activated, which uses knowledge about expected hot water usage to determine a failure may be occurring. This node then activates each of its children in the causal model: ResourceDamaged, NonCoordinatingClient, SensorDamaged and MalfunctioningClient. Analysis of coordination logs, and sensor data from both the tank and output pipes rule out the first three possibilities. MalfunctioningClient proceeds to contact those agents using water line two - only the dishwasher in this case. The dishwasher sends the waterheater its diagnosis of UnexpectedTaskFrequency, which acts as supportive evidence for the MalfunctioningClient diagnosis. The waterheater, acting on this diagnosis, can either reduce the priority of the dishwasher's coordination requests, or cut off the flow of water into water line two.

The other appliance, unaware of the problems being handled by the waterheater, only knows that its coordination attempts have been consistently refused for quite a while. Realizing that its ability to meet its deadline requirement is in question, the diagnosis node GoalDeadlineUnattainable activates its child nodes: MethodFailure, InsufficientFunds, GoalDeadlineMissed, UnexpectedScheduleDuration and ResourceUnavailable. The ResourceUnavailable node can use the attempted schedule, local TÆMS task structure and coordination logs to determine that the hot water resource is unavailable. This in turn activates the child nodes NoRsrcCoordination, NonExistantResource, OverloadedResource and MalfunctioningResource. OverloadedResource gains evidence by querying for and receiving the UnexpectedResourceUsage and MalfunctioningClient diagnoses from the waterheater. At this point, the other appliance can reschedule with the knowledge that the hot water resource is overloaded. Further analysis by the OverloadedResource node could also detect when the problem had been resolved, allowing the other appliance to continue using hot water as it becomes available.

Slight modifications to this example demonstrate how diagnosis can be used to detect aggressive intrusions into the multi-agent system. In the altered scenario, the dishwasher's logic has been compromised in such a way that it continually uses hot water without coordination. The activity exhibited by the remaining two agents could remain much the same, with the exception that the dishwasher would produce a NonCoordinatingClient diagnosis in lieu of MalfunctioningClient (since it is unlikely that the compromised dishwasher would admit to malfunctioning). The waterheater could unilaterally react to this diagnosis by cutting off the water supply to line two, a reasonable short term repair for this problem.

### Over-coordination

One interesting efficiency scenario is that of over-coordination. A spectrum of coordination models is possible in multi-agent systems, ranging from fully explicit, verbose communication to "well-known" assumptions or implicit agreements. Clearly, it is more efficient to reduce inter-agent communication if possible, but how can an agent know when it is safe to do so? One method, similar to a technique described in (Toyama & Hager 1997), makes use of a persistent diagnostic process to monitor tested changes.

In this example, the UnecessaryRsrcCoordination node begins its work by monitoring the coordination which takes places over the system's resources. If it detects that requests for a particular resource are always being satisfied, it forms the hypothesis that it is not necessary to coordinate over that resource, either because it is automatically replenished (e.g. a low-bound maintaining waterheater) or a common resource lacking contention (e.g. electricity under normal circumstances). The problem solving component in the agent could then react to this diagnosis by ceasing coordination over that resource. Over time, the persistent diagnosis object then monitors this resource, to see if methods requiring it are affected, and adjusting the diagnosis as needed. This diagnosis can then provide the feedback necessary for the agent to maintain the situation-specific assumptions needed for it to be efficient in its environment. A more complicated diagnostic process could also further classify coordination relations, based on coordination type, temporal cycles or sensitivity to requested resource amounts.

### Method Outcome Discrepancies

A key measure of adaptability is how the agent responds to unexpected results. If a method fails, does the system blindly reschedule, or does it take into account the reasons for the failure? How does the agent react when method performance varies within what is considered normal ranges?

The introductory example demonstrates a problem of this type. In the example, a NoRsrcCoordination diagnosis was used to target the perceived fault, which allowed the agent to repair its original schedule rather than generate a new one which may have made the same mistake. A more interesting scenario involves an agent's reaction to different levels of discrepancies - when should an agent adapt to, ignore or repair a problem? The TÆMS task language allows the designer to explicitly encode the expected behavioral ranges a method may exhibit, and learning algorithms can be employed to maintain the structure as time passes. Results falling within these ranges are expected, and should not trigger diagnosis. The remaining issue requires more information to discriminate enough to make an intelligent choice. More detailed organizational knowledge about method behavior can be used to determine thresholds, allowing the agent to discriminate between acceptable and unacceptable variations in long term performance deviations. Diagnosis can then use this information and other available evidence to provide the specific problem description seen in other examples, which the problem solver can use to pick the appropriate course of action.

Consider an example involving method duration. An agent executes method $X$, expecting it take between 10 and 15 clicks to complete (a click being an arbitrary unit of time), as encoded in its task structure. In addition, the designer has specified in the organizational knowledge that durations up to 40 clicks are within "acceptable" tolerances, which is designed to take into account such things as network activity fluctuations or noisy sensor data. If $X$ were to complete in 25 to 35 clicks, the agent would take note of the event and modify performance characteristics in TÆMS after determining the deviation was not caused by a fault other than inaccurate expectations (such as missing resources, hindering method interrelationships). Instead, $X$ takes 100 clicks to complete, clearly outside of its expected range. This value is also well outside of the acceptable tolerances, so the agent should not adapt its expectations to this new situation. The aberration would then initiate diagnosis activity, which would monitor future behavior of this method. We will assume that the operating conditions match no other diagnosis' symptoms. Over time, as $X$ is executed again, a clearer picture of its current performance could be generated, which can help determine if the failure was a single event, or the first instance of a software or hardware fault, or an intrusion.

## Detection and Diagnosis Sensitivity

While diagnosing problems in a multi-agent setting is an interesting problem in its own right, it is also important to examine the effect of detection and diagnostic frequency on overall system behaviors. Specifically, one may wonder what the appropriate level of "aggressiveness" is for detection and diagnosis. On one hand, if the process is very sensitive, effort may be wasted monitoring behaviors operating normally, or adapting to faults that don't exist. On the other hand, a more skeptical diagnostic system may ignore triggers signifying larger problems, or spend so much time gathering evidence and improving confidence that the eventual adaptation comes too late.

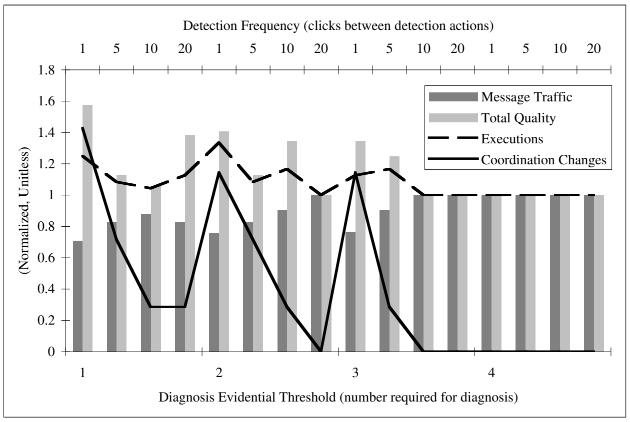


Figure 4: Results from the over-coordination scenario

The notion of inappropriate coordination will be used to more closely examine this continuum. In the experiment, a water-using appliance and the waterheater interact. The water-using appliance executes a schedule using different amounts of water, at different times. The waterheater attempts to maintain a pre-specified level of water in the tank, and so will react to unanticipated usage. The heater is also able to coordinate over water usage, thus ensuring that the water is available when it is needed. The challenge here is to determine when the water should be coordinated over: should the agent expend energy to ensure the needed water is present, or should it simply use the water and accept the fact that there is some probability of failure due to lack of resources?

To diagnose the situation, the water-using appliance examines the coordination pattern with the waterheater, and the resulting effects when coordination is stopped. Thus, if the agent determines it is almost always receiving positive responses from its coordination requests, it may opt to stop coordinating with the hypothesis that the waterheater can reactively keep up with its requests. If, however, this is not the case, negative execution results will prompt the agent to begin coordinating again.

Two parameters are varied in the different experiments: the frequency at which the coordination and result data is examined, and the amount of evidence required for a diagnosis to be produced (e.g. for a change in coordination to take place). The scenario is allowed to run for a set amount of time, after which the number of coordination messages, overall quality, number of method executions, and number of coordination changes are recorded. An optimal agent would thus minimize the number of coordination attempts and maximize quality. The number of coordination changes and executions is not good or bad in and of itself, but is indicative of the rate and results of adaptation the agent has exhibited.

The results of the experiment can be seen in Figure 4. The lower X axis measures the amount of evidence required for the agent to create the diagnosis which would effect coordination decisions. The upper X axis should be viewed as four separate cases, each of which measure the effects of decreased detection frequency. The data shown is otherwise unitless and normalized, to help accentuate the observed trends.

The overall trend of the graph indicates that as more pieces of diagnostic evidence are required, or if the detection frequency is decreased, the agent will be more conservative in its coordination decision. To see this trend, look at the rate of coordination change first along the lower axis, representing increases in required evidence, then in each of the four subsections indicated by the upper axis, which shows decreases in detection frequency. In each case, the rate of coordination change decreases.

The effects of this coordination decision are shown by the bars in the graph. At higher rates of change (e.g. the left side of the graph and sub-graphs), the agent tends to communicate less, while at the same time producing more quality. This may at first seem contradictory - one would expect that an agent which coordinated more would have a higher quality, since it is supposedly increasing the probability that the required water will be available. What is not considered, however, is the time cost of coordination. When coordinating, the agent both wastes time waiting for coordination responses, thereby delaying the start of execution, and may also decide not to execute at all, when its coordination request is denied. These two items combine to reduce the rate of execution under coordination (shown by the Executions line), which in turn reduces the total amount of quality. Thus, this data suggests that under these environmental and behavioral circumstances, the diagnosis component should be restricted to a narrow window of data, which is continuously analyzed, allowing the agent to quickly react to changes.

The results shown here are meant to persuade the reader that an important continuum exists in the sensitivity of diagnostic components, rather than claim that any one point is reasonable for all, or even this particular type of fault. Clearly the optimal point in this range is dictated both by the techniques being employed and the circumstances under which they are used. Designers of diagnosis-capable agents should keep this tradeoff in mind, so as to avoid counteracting the benefits of adaptability with unnecessary overhead.

## Current Implementation & Future Work

The diagnosis architecture described in this paper has been implemented, and used in several coordination and behavior fault based scenarios. We have over a dozen agents operating in the Intelligent Home, with varying levels of diagnostics ability and ad hoc adaptability. The causal model shown in this paper has also been implemented, and extensions to other software, hardware and intrusion based faults are being considered.

In the future, we plan on more closely addressing the eventual reaction to diagnosis: adaptation and repair mechanisms. Specifically, the implementation, domain independence and quantitative analysis of these mechanisms will be considered. We also plan to more thoroughly analyze the efficiency cost of adapting to non-fault conditions.

## Conclusion

Adaptation can be an important part of any computational system, but the susceptible of multi-agent systems to broad classes of computational, behavioral and adversarial faults make it especially vital. We believe that a robust and flexible diagnostic component, coupled with informative models and data, is a necessary part of this adaptive capability. Agents capable of self and remote diagnosis will play an important role in making multi-agent systems both robust and efficient.

## References


Bazzan, A. L.; Lesser, V.; and Xuan, P. 1998. Adapting an Organization Design through Domain-Independent Diagnosis. Computer Science Technical Report TR-98-014, University of Massachusetts at Amherst.

Decker, K. S., and Lesser, V. R. 1993. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management* 2(4):215–234. Special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".

Decker, K. S., and Lesser, V. R. 1995. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, 73–80. San Francisco: AAAI Press.

Hudlická, E., and Lesser, V. R. 1987. Modeling and diagnosing problem-solving system behavior. *IEEE Transactions on Systems, Man, and Cybernetics* 17(3):407–419.

Hudlická, E.; Lesser, V., P.; J.; and Rewari, A. 1986. Design of a distributed diagnosis system. UMASS Department of Computer Science Technical Report 86-63.

Kaminka, G. A., and Tambe, M. 1998. What is wrong with us? improving robustness through social diagnosis. In *in Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*. AAAI.

Lesser, V.; Atighetchi, M.; Horling, B.; Benyo, B.; Raja, A.; Vincent, R.; Wagner, T.; Xuan, P.; and Zhang, S. X. 1999. A Multi-Agent System for Intelligent Environment Control. In *Proceedings of the Third International Conference on Autonomous Agents*. Seattle,WA, USA: ACM Press.

Sugawara, T., and Lesser, V. 1993. Learning control rules for coordination. In *Multi-Agent and Cooperative Computation '93*, 121–136.

Sugawara, T., and Lesser, V. R. 1998. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*. To appear.

Toyama, K., and Hager, G. D. 1997. If at first you don't succeed... In *in Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*. AAAI.

W. Hamscher, L. Console, J. d. K., ed. 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann.

## Appendix: Diagnosis Implementation Details

To allow the reader to better understand how diagnosis currently functions in the agent, the implementation details of several of the causal model nodes will be discussed in this appendix.

### LongerDuration

As can be see from Figure 2, the LongerDuration node is triggerable, meaning it periodically checks simple observable traits which might be symptoms of a larger problem. The node uses two pieces of information to perform this check: the list of currently executing methods and the local TÆMS task structure. During the trigger-check phase, the node will compare the current running time of each executing method to the expected time indicated by the task structure. If the current time exceeds the mean of the expected time plus some given threshold, the node becomes *activated*.

Once activated, a new instance of node is created which more closely monitors that single method whose duration seems to be too long. A different, presumably looser, standard is used to determine excessive duration at this point. This allows the triggering to be somewhat sensitive, while the actual diagnosis can allow for a wider range of performance. If this new threshold is also passed, a diagnosis will be generated noting the problem. If the method does not pass the new threshold, the instance will silently deactivate itself.

### IncorrectMethodQualDistribution

As this node is not triggerable, it depends on other nodes for activation. Typically, this node will be activated by the LowerQuality node, which itself would be triggered by a method whose resultant quality is lower than expected (determined by a method similar to that described in the previous section).

IncorrectMethodQualDistribution, once activated, acts primarily as an interface to a local long-term learning component. The learning component will independently monitor all method invocations, and track their execution characteristics. As results are obtained, the learning component is able to build its own local version of the task structure, which can be used to verify or contradict the expected values present in the agent's actual task structure. If a sufficient amount of evidence has been gathered by the learning component, a chi-squared test is made to determine the significance of the differences (if any) between what it has learned and what was expected. If the difference is significant, a diagnosis is produced, otherwise the node silently deactivates itself.

The two nodes IncorrectMethodDurDistribution and IncorrectMethodCostDistribution work similarly.

### NoRsrcCoordination

This node, typically activated when a method results differ from expectations, is used to determine if the symptoms could have been caused by a missing resource which was uncoordinated over. The diagnosis activating the node will contain a reference to the method which has misbehaved. By examining the agent's local task structure, the node can determine if the method in fact used resources by searching for the method-to-resource interrelationships which conventionally indicated such usage.

Agents in our environment actually posses two copies of its task structure, which are differentiated as subjective and conditioned views. The subjective view represents what the agent believes to be the true working conditions imposed by the environment. Interrelationships included in this model indicate that relationships between methods and resource do exist, and will have impact on one another (so far as the agent knows). The conditioned view, however, represents only those relationships which the agent deems necessary to coordinate over. Thus, a method-to-resource relationship in the subjective view indicates the the agent is aware a certain method will interact with the resource; the presence of the same relationship in the conditioned view indicates the the agent has actively decided to coordinate over that interaction. Thus, the conditioned view is used as a coordination-independent view of the coordination activities the agent will exhibit at runtime.

By examining the dichotomy between the subjective and conditioned view, the NoRsrcCoordination node can determine both when resource interactions are taking place, and whether or not coordination took place over that interaction. When a poorly performing method is delivered to the node, it first determines if an interaction took place by looking for relationships in the subjective view. If such relationships don't exist, the node simply deactivates because there was no coordination that could have taken place. If the relationship does exist, however, and the corresponding relationship is not present in the conditioned view, the NoRsrcCoordination may pose a diagnosis indicating that poor performance may be a result of needed resources which were not coordinated over (and therefore may have not been available).

### UnecessaryRsrcCoordination

This node is also triggerable, searching for possible instances where coordination may be unnecessary. The triggering activity in this case is whenever coordination over a resource is performed for the first time, which is determined by listening to the event stream emanating from the coordination component. Once a coordination event is observed, the node proceeds to set up long term monitoring facilities to track the activity and results of this coordination.

Once in place, the monitors count the number of times a resource is attempted to be coordinated over, and the number of acceptances and rejections arising from these attempts. Once a certain amount of evidence (coordination attempts) have been made, the node examines the gathered data to calculate the probability of a coordinate attempt being accepted. If this ratio is above a certain threshold, a diagnosis is created indicating that coordination may be unnecessary.

If no coordination changes are made by the agent, the node will continue gathering evidence and updating its diagnosis when changes are observed. A sliding window of evidential data is maintained which helps prevent the node's diagnosis from being unduly affected by historical events. If, however, a coordination change is made, the node will response by throwing out all of its evidence, and begin listening to the NoRsrcCoordination node. Because no coordination is taking place, the monitors previously put in place will offer no new evidence as time passes. Instead, The results of the action will be the new, albeit indirect, source of evidence. By monitoring the NoRsrcCoordination node, UnecessaryRsrcCoordination can determine if its diagnoses adversely affects the activity of the agent, and can revise its diagnosis as necessary, which should allow the agent to know when and if coordination should be restored.

### FailedDurationEstimate

This node is responsible for diagnosing when an agent performing a previously-coordinated over task fails to meet local and remote expectations (as opposed to FalseDurationEstimate, which attempts to determine when a remote agent misrepresents the duration). A good example of this is when an acting agent fails to achieve the duration estimate it submitted in a contract accepted through a contract-net protocol because of a local execution error (a required resource might not have been available, or the local distribution description might be incorrect).

The parent node, IncorrectCoordinatedDurationEstimate, has presumably already determined that the method in question was both coordinated over and exceeded the agreed upon duration. If this is the case, and the observed duration was greater than the agreed upon duration (if any), diagnosis takes place. FailedDurationEstimate node uses a form of distributed diagnosis to gather evidence by querying the executor for pertinent diagnoses it may have generated. If the executing agent reports a diagnosis concerning the method in question and its duration, the requesting node can infer that an execution error occurred, which resulted in the method taking longer to complete. A diagnosis could be produced based on this information.