# Adapting an Organization Design through Domain-Independent Diagnosis[*]

Ana L. C. Bazzan, Victor R. Lesser, and Ping Xuan
Department of Computer Science
University of Massachusetts, Amherst, MA 01003

**Abstract**: Coordination is an essential technique in multi-agent systems. However, sophisticated coordination strategies are not always cost-effective in all problem-solving situations. This reinforces the need of situation-specific control rules, in which the benefits and the costs of coordination for the current situation are taken into account. We focus here on situation-specific control based on a long-term understanding of the frequency of tasks occurring, the characteristics of these tasks in terms of their resource and coordination needs, and the available resources. Another way of describing this type of long-term situation-specific control is an organizational design. In this paper, we assume that we are given an initial organizational design. However, as the characteristics of the environment change, this organizational design needs to be adapted. We discuss a domain-independent approach to detecting that some important aspects of the environment may have changed and to diagnosing/explaining exactly what these changes are. This work is done in terms of the GPGP/TÆMS domain-independent coordination architecture.

## 1 Introduction

In multi-agent systems, an agent's control decisions based only on its local view of problem-solving task structures may lead to inappropriate decisions. From the perspective of the model of Decker and Lesser (1995) for instance, each of the agents makes decisions based on a subjective view of its own and other agents' activities and its view of available resources. In certain situations this subjective view will lead to an agent taking ineffective or inappropriate actions because certain non-local activities or the non-local resources have not been appropriately taken into account. However, *acquiring* and *exploiting* non-local views, so that an agent could make more informed choices, is not cost-effective in all situations due to limited computational resources. Therefore, for specific problem-solving situations, it may not be worthwhile to acquire such additional views or fully exploit the current view; some level of non-coherent activity may be the most effective strategy.

An additional motivation to pursue situation-specific coordination comes from past research’s conclusion that, in agents as well as in human organizations, there is no one best approach to organizing and controlling computational activities for all situations when the computational and resource costs of this control reasoning is

[*] Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0249. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.
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), Rome Laboratory or the U.S. Government.

taken into account. Instead, each agent should be able to tailor, to the specifics of the current situation, all aspects of control reasoning to balance the resource requirements of this reasoning against the gains achieved by more coherent agent behavior.

We focus here on situation-specific control based on a long-term understanding of the problem-solving environment including the frequency of specific tasks occurring, the character of these tasks in terms of their resource and coordination needs, and the available resources. Another way of describing this type of long-term situation-specific control is an organizational design. In this paper, we assume that we are given an initial organizational design which includes not only situation-specific coordination strategies associated with different agents, and their associated tasks, but also some of the assumptions about the environment which were the basis for the specific organizational design chosen.

In a complex and open operating environment, the characteristics of the environment can either change gradually so that at some point the environmental characteristics are quite different from the initial assumptions motivating the original organizational design, or can change abruptly when for instance a key resource goes off-line, or its performance characteristics change, or the frequency of specific tasks occurring changes markedly, or a task's performance characteristics (in terms of resource usage, character of its output, and frequency of specific types of outputs change markedly. These types of changes require the system to rethink what is the most appropriate organization it should be in. An understanding of the detail character of these changes in the environment and how they are effecting performance can then be used to either modify the organizational design in some local manner or redesign significant parts of it.

In this paper, we discuss a domain-independent approach to detecting that some important aspects of the environment may have changed and to diagnosing/explaining exactly what these changes are. This work is done in terms of an extended version of the GPGP/TÆMS domain-independent coordination architecture (Decker, 1995; Decker and Lesser, 1995; Decker, 1996; Lesser et al., 1997), which permits situation-specific control to be represented. In this version, an organizational designer can focus coordination activities to reason only about a subset of possible coordination relationship, which exist among agents. This focusing can be further selective so that it is done on a task and situational perspective (such as the current loading of the agent), can also indicate where default assumptions about other agent activities and availability of resources should be used in making coordination decisions, and how much resources should be spent in making coordination decisions. This focussing leads to what we call a *conditioned* agent view (by the organization) of local agent activities and their relationships to other agent activities that an agent will use to make its coordination decisions. This conditioned view is in confront to a *non-conditioned* agent view which is the agent view prior to it being modified by organizational knowledge. Finally there is an objective agent view of the present situation which includes a complete and accurate view of local activities and all their interactions with non-local activities.

The current research aims at developing a framework to achieve situation specific control in *changing environments*, which, given the computational costs of finding out about the activities of other agents (non-local views), is able to acquire *just those pieces of information needed to perform a given problem-solving* *process*. This is especially important in changing environments, as changes are likely to require modifications in the coordination strategies as well.

As we want to focus on systems which can survive in changing environments, we present in this paper detection and diagnosis technologies which are needed in order to detect and explain changes in the behavior of agents and environments. Diagnosing inappropriate behavior of the problem-solving system is a crucial step to correct the system's domain knowledge about the characteristics of agents problem-solving, and available resources, which then allows a reorganization of the society of agents to permit effective processing. This, in turn, is the key to survivable systems. Lesser (1998) discusses an agent architecture that includes such a diagnosis component. To reiterate, we feel that degraded system performances caused by changing task and resource characteristics, by hardware or software failures and intrusions can be handle within our framework. The unifying theme among these many different causes for degraded performance is that they either directly or indirectly cause assumptions that were the basis of current agent organization to no longer be valid.

In the three next sections of this paper we briefly discuss the related research, introduce the representation and coordination tools, TÆMS and GPGP, and discuss some approaches to diagnosis, respectively. After that we discuss a scenario in which the behavior of the multi-agent system is not as expected because the environment has changed. Section 5 returns to TÆMS by discussing some extensions needed in order to adapt it to diagnosis. We then present our approach to detecting and diagnosing such situations (section 6), and discuss in detail with respect to the scenario laid out early (section 7).  The last section presents the main conclusions and discusses our research directions.

## 2 Related Work

Our development of a detection and diagnosis component as part of an organizational designer builds on the following previous research. Hudlicka and Lesser (1987) developed causal models to reason about the behavior of problem-solving systems in order to diagnosis faults in their behavior due to faulty sensors or communication channels. They saw diagnosis as a way to discover an inappropriate control setting, which they called *problem-solving control error*. This is particularly interesting for us since such a setting can be based on a (partially or totally) wrong assumption for instance.

As in their work, we also see some issues arising from the lack of standards for the correct behavior, from the potential combinatorial explosion of possible explanations and data to analyze, and thus the need to somehow avoid an extensive search. This has been indeed one of the main motivations for model-based diagnosis (e.g. Davis, 1984), which we exploit by using TÆMS to describe the physical and functional organization, and the behavior of the agents during the problem-solving process.

Sugawara and Lesser (1993, 1998) show how to introduce a domain-dependent detection, diagnosis and organizational redesign component into a multi-agent system that generates new coordination rules to correct detected performance issues. This work is based on significant amount of domain-specific knowledge and the use of comparative analysis developed by Hudlicka and Lesser (1987), i.e. comparing two similar cases, when there is not enough explanatory knowledge to understand why certain activities are important or redundant. In our framework we use a similar kind of reasoning, however based on a richer and domain-independent model that explicitly models the effects of resource usage; this eliminates the need for much of the specialized domain-specific knowledge that was used by them.

Our work uses the TÆMS and GPGP frameworks to tackle detection and diagnosis of problem-solving systems in a domain-independent fashion. Our approach focuses on the discovery of those relationships among agent activities which are important for explaining why an inappropriate behavior has occurred. We are particularly interested in detecting inappropriate assumptions about agent activities and resource usage which are responsible for the current coordination strategy implemented in GPGP being inappropriate. These assumptions are the basis of organization knowledge which conditions the information used by the GPGP coordination when making coordination decisions.

We are also interested in recognizing situations where coordination costs are very high will respect to gains achieved. This is another form of inappropriate behavior that needs to be diagnosed.

## 3 TÆMS and GPGP

We use the TÆMS framework (Decker and Lesser, 1993; Decker, 1995) to represent formally the coordination aspect of our problem. TÆMS allows the construction of a task model in which agent's activities are represented in terms of a set of task groups. A task group (as the one depicted in Figure 1) is an acyclical graph representing tasks and their subtasks. The leaves of the graph are called (executable) methods (like M1 to M5 in Figure 1), which have probability distribution on their characteristics like quality, cost, and duration. The quality of a task group depends on how and when its subtasks and their methods are executed. For example, quality can be accrued by a quality accumulation function (qaf) like *sum*, which indicates that not all subtasks of the parent task need to be accomplished. Other examples are *max* (or), *min* (and), etc. Besides, the local effects of the execution of methods on the quality and duration of their supertasks, there exist non-local effects (NLE) like for instance *enables*, *facilitates*, etc., that represent data flow relationships among tasks.

A task **T** may enable a method **M** in the sense that the quality of M cannot be accrued until T is completed, i.e. the earliest start time of M is the finish time of T. Therefore enables is a hard relationship, which means it has to be necessarily observed. When a task $\mathbf{T_1}$ facilitates other task $\mathbf{T_2}$, the duration and/or quality of $T_2$ is positively affected by a factor, but may not necessarily be observed since facilitates is a soft relationship. Another important set of NLE's applies to methods and their use of resources. *Uses*, *replenishes*, etc. are NLE's running from one method to one resource and altering its state. Conversely, *limited, exclusive access,* etc. are NLE's running from one resource to a method.

By using TÆMS it is possible to construct a task model of a problem-solving situation, including alternative ways that tasks can be carried out. The real structure is called an *objective* model of the environment. However, agents have each a *subjective* model or view of it, which they use to predict other agents' actions. Depending on how inaccurate the agents' subjective views are, unperceived characteristics may lead to either inappropriate or unexpected behavior.

The basic idea behind GPGP is that each agent constructs its own local view of the activities (that it intends to pursue in the short-to-medium-term time frame), and the relationships among these activities. This view can be augmented by information from other agents, thus becoming a view that is not entirely local. Individual *coordination mechanisms* that are part of GPGP help to construct these partial views, and to recognize and respond to particular task structure relationships by making commitments to other agents. These commitments result in a more coherent, coordinated behavior by affecting the tasks an agent will execute, when they will be executed, and whom their results will be transmitted to. The set of mechanisms are: (1) communicate non-local views; (2) communicate appropriate results; (3) avoid redundancy; and (4,5) handle hard and soft coordination relationships that extend between tasks at two agents. The gathered information about the task structure is used for agents to commit themselves to the execution of determined methods by certain times with certain qualities. These commitments are the basis of coordination of activities among agents.

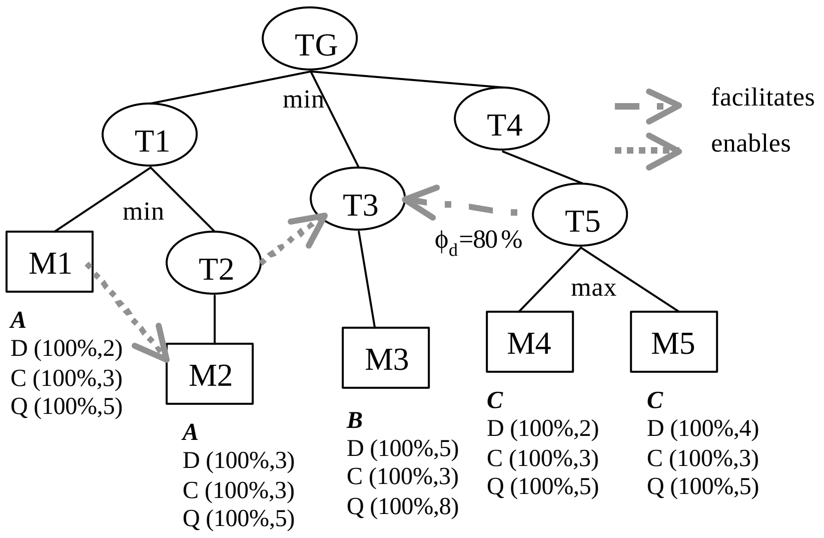


Figure 1: TÆMS Task Structure. TG is the task group. Ti's are the tasks, and $M_i$'s are the methods. Bold, italics letters (*A*,*B*,*C*) indicate the agent responsible for the task/method; D, C, and Q are the duration, cost, and quality distributions respectively. NLEs (enables and facilitates) are as indicated, and $\phi_d$ is the facilitation power.

## 4 Modifications on TÆMS and GPGP to Support a Diagnosis Module

### 4.1 Organizational Design

An organizational design specifies roles for agents, considering the characteristics of the tasks, and methods as well resources available, thus limiting agent's activities. This constitutes the long-term behavior, in opposition to the short-term behavior, in which agents decide which methods to execute in order to achieve the task, based on the dynamics of the current problem-solving situation.

If the nature of the task, or the availability of resources changes substantially, causing the system functioning to be ineffective, a method of detection and diagnosis is required, and thus several issues regarding representation need to be addressed:

- how to model the characteristics like frequency and which agent or classes of agents are likely to be responsible for certain tasks, and the relative importance of the task utility;
- how to model alternative methods for accomplishing a task so that the system can reason about the tradeoffs involved;
- how to model resources present in the environment.

An organization is (re)designed either when a problem of an unknown variety arrives or when the current organization is diagnosed as being inadequate. The inputs to organizational design, which can be described either in the form of grammar that specifies a whole class of potential problems (Nagendra Prasad *et al.* 1996), or as a set of representative TÆMS task structure, are: i) the current performance goals of the system; ii) long-term agent knowledge about typical tasks together with their frequencies of occurrence, possible relationships among other tasks in terms of co-occurrence, and specific performance goals and resource requirements; and iii) the available resources and their performance characteristics. The output of this process is an organizational design - a description for each agent of: tasks it is responsible for handling and their importance; coordination relationships among tasks handled by other agents to consider when scheduling this task, and the level of effort to be spent in establishing them; and long-term commitments to be honored, their priority, and which agents should be notified when these commitments cannot be honored.

An organizational design is used when an agent receives a request to execute a new goal (with its associated task structure). This request can come from either its local problem-solving component or other agents. Based on the organizational design, an agent's coordination component decides how to constrain the characteristics of the task so the agent can meet coordination requirements with other agents. For example, the coordination component might introduce a deadline in order for the task's results to be useful to another agent. Such decisions are made in collaboration with the coordination components of relevant agents.

## 4.2 Resource Model

In the original TÆMS model, resources were not explicitly represented. However, when specifying a task structure, it is still possible to implicitly represent resource constraints by using complex task interdependencies such as enables, disables, facilitates, and hinders, etc. When the tasks become more complex and more resource constraints are involved, such implicit representation becomes very inefficient in specifying the task structures. Therefore, an explicit resource representation for TÆMS is needed. With an explicit resource model, we can specify complex resource states and complex relationships between tasks and resources. Specifically, we identify two classes of non-local relationships, namely the *uses* relationship, which specifies the resource need of the tasks, and the *limits* relationship, which is used to represent the effect a resource has on a task's attributes such as duration, quality and cost (Decker, 1995).

However, the ease of task specification is just one reason for adding the explicit resource representation. Another reason is that coordination over resource constraints is a very important aspect of distributed problem solving, and thus we need to have a resource model common to all agents so that they can coordinate based on descriptions of resource state as well as their own belief of the resource states (Decker and Li, 1998). Another important reason is that when resource becomes more complex, e.g. having complex state information and state transition dynamics, an abstraction of resource is needed. Such abstraction can be provided through an agent that wraps around the resource. It is often the case that the access to a resource is controlled by an agent, rather than in the simple case that the resource is passive and the management of the resource is through the coordination of the agents using the resource.

Resouce modeling is tackled by Decker and Li (1998) and by Lesser et al. (1998a). The former discuss the coordinated hospital patient scheduling scenario, using TÆMS and GPGP, augmented by a sixth coordination mechanism, which handles the use of mutually exclusive resources by the tasks. When several agents try to use one of these resources at overlapping times, only one can actually get it and execute the task. Therefore, agents must send a bid for the resource in a specific time interval. Agents also have to compute the local priority of the task, which is a function of the utility. The agent which wins the bid, gets the resource and can keep its schedule; the others have to execute another task if possible and try to get that resource later.

In Lesser et al. (1998a), resource coordination are handled in a broader way. The Intelligent Home (IHome) scenario deals with task events and their use of resources, represented by means of TÆMS. When there are insufficient resources to meet all demands, there is a negotiation from a temporal perspective, also using priority allocation. There is no specific coordination over the interaction of agents (like in GPGP). These interactions are detected when agents announce their need of a given resource.

We base our model in both approaches. in order to support detection and diagnosis for resource related problems, as detailed in Section 5. Besides, we do resource monitoring and tracing. As for the explicit resource representation in TÆMS, we follow the implementation of Decker and Li (1998).

## 5 Conceptual Examples of Inappropriate Behavior

The situation specific nature of coordination activities means that coordination decisions based on local and possibly imprecise knowledge of the situations are error prone. Next, we show through several examples situations where this happens.

## 5.1 Overcoordination: Turning On GPGP Coordination Mechanisms

### 5.1.1 Mechanism1 (Update Non-Local Views)

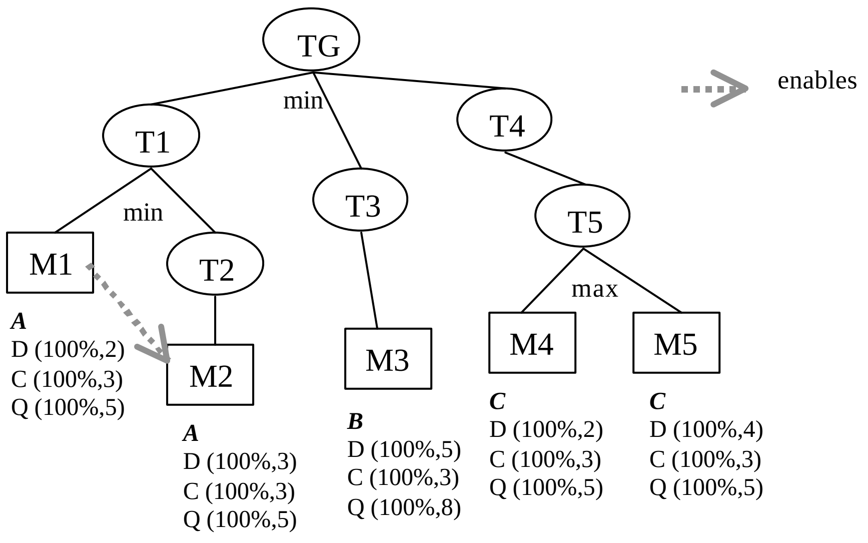


Figure 2:

Consider the objective TÆMS task structure depicted in Figure 2 and corresponding subjective task structures, shown in Figure 3, hold by agents A, B, and C (left, center, and right) respectively. We can see that the subjective task structures agent have are correct related to the objective one. This is not always the case. Thus the *local* beliefs ($\mathbf{B_k^L}$, denoting the beliefs set of the k-th agent) hold by agents can be expressed the following way:

$\mathbf{B_A^L}$ = {subtask (M1,T1); subtask (M2,T2); subtask((M1,T2), T1); subtask (T1, TG), enables (M1, M2); duration (M1,(100%,2)), duration (M2,(100%,3)); qaf (T1,min), qaf(TG,min), cost(M1,(100%,3)), cost(M2,(100%,3))}

$\mathbf{B_B^L}$ = {subtask (M3,T3); subtask (T3,TG), duration (M3,(100%,5)), qaf (T1,min), qaf(TG,min), cost(M3,(100%,3))}

$\mathbf{B_C^L}$ = {subtask (M4,T5); subtask (M5,T5); subtask((T5,T4); subtask (T4, TG), duration(M4,(100%,2)), duration (M5,(100%,4)); qaf (T5,max), qaf(TG,min), cost(M4,(100%,3)), cost(M5,(100%,3))}

The case we want to examine is one where the designer of the system, possibly unaware of the consequences, opts for turning on all GPGP coordination mechanisms, in particular the mechanism number 1, updating of non-local views. For details on how this and other coordination mechanisms discussed below work, please refer to Decker and Lesser (1995). The result of such updating produces the following beliefs $\mathbf{B_k^j}$ where the index j denotes the beliefs set resulting after the use of the coordination mechanism j):

$\mathbf{B_A^1} = \mathbf{B_A^L}$ +{nil}
$\mathbf{B_B^1} = \mathbf{B_A^L}$ +{nil}
$\mathbf{B_C^1} = \mathbf{B_A^L}$ +{nil}

This means that there was no addition of beliefs resulting from the use of the coordination mechanism 1, and hence this is a clear situation where there was overcoordination. If we assume that the use of this mechanism costs 1 unit for each agent, the total cost associated with the task group TG is the cost of coordination, namely 3, plus the cost of T1 (which is computed by adding costs of M1 and M2), the cost of M3, and the cost of M4 (this one is preferred over M5 due to lower duration and higher quality), which adds up 15 units. Without the unnecessary coordination, the cost would be only 12.

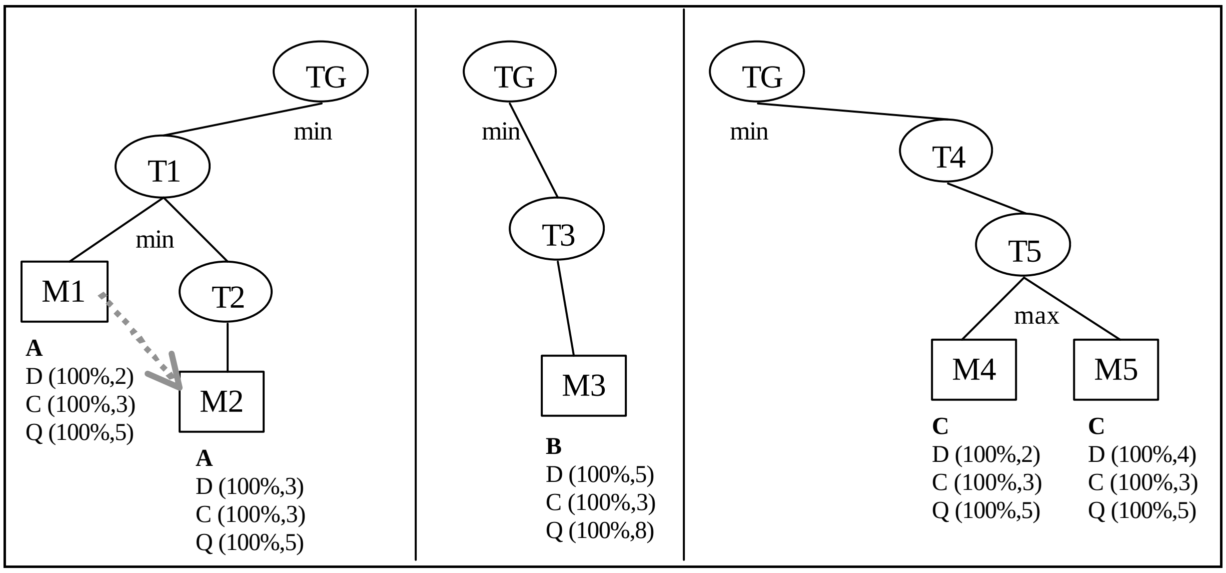


Figure 3:

### 5.1.2 Mechanism 3 (Recognizing Simple Redundancies)

Any task that uses a *max* quality accumulation function indicates that only one of its subtasks needs to be done. The mechanism 3 of GPGP addresses this kind of simple redundancy: when more than one agent wants to execute a redundant method, this mechanism randomly chooses one to execute. Let us assume that the designer turned on this mechanism too for the scenario discussed in the last section (Figures 2 and 3). The use of mechanism 3 produces:

$\mathbf{B_A}^{\mathbf{3}} = \mathbf{B_A}^{\mathbf{L}}$ + redundant{nil}
$\mathbf{B_B}^{\mathbf{3}} = \mathbf{B_A}^{\mathbf{L}}$ + redundant{nil}
$\mathbf{B_C}^{\mathbf{3}} = \mathbf{B_A}^{\mathbf{L}}$ + redundant{nil}

That means, again the use of a coordination mechanism did not add useful information to the agent's knowledge; agents only know a new piece of information, namely that their activities are not redundant. But this might have a high cost!

### 5.1.3 Mechanisms 4 and 5 (Recognize Hard and Soft Relationships)

Similarly to the cases above, assuming mechanisms 4 and 5 as turned on for that example, after triggering mechanism 4, the beliefs set looks like:

$\mathbf{B_A}^{\mathbf{4}} = \mathbf{B_A}^{\mathbf{L}}$ + enables{nil} [1]
$\mathbf{B_B}^{\mathbf{4}} = \mathbf{B_A}^{\mathbf{L}}$ + enables{nil}
$\mathbf{B_C}^{\mathbf{4}} = \mathbf{B_A}^{\mathbf{L}}$ + enables{nil}

[1] Actually this summation goes on with other defined hard NLEs (disables, etc.). The same applies to the soft NLEs as other may be defined (hinders, etc.)

That means that no previously unknown hard NLE was discovered (remember that the *enables* appearing in Figure 3 is local and is already in the set $\mathbf{B_A}^{\mathbf{L}}$).

Similarly, using mechanism 5:

$\mathbf{B_A}^{\mathbf{5}} = \mathbf{B_A}^{\mathbf{L}}$ + facilitates{nil}
$\mathbf{B_B}^{\mathbf{5}} = \mathbf{B_A}^{\mathbf{L}}$ + facilitates{nil}
$\mathbf{B_C}^{\mathbf{5}} = \mathbf{B_A}^{\mathbf{L}}$ + facilitates{nil}

Again, no useful information is added to the agents beliefs sets.

### 5.1.4 Coordination Mechanism 6 (Resource Constraint)

To discuss this example, let us use a scenario motivated by the IHome (Intelligent Home) project (see Lesser et al., 1998a) for a detailed description), which deals with using coordination of several home appliances and resources usage. Figure 4 shows the task structure for the scenario: when the user comes home after work, s/he expects to find hot water for a shower, plus the room at a preset temperature, and the following tasks done: dishes washed and the food ready to be served. These tasks involve several appliances agents like the dish-washer (DW), the micro-wave (MW), the air conditioner (A/C), the shower (S), as well as the agent controlling the boiler for hot water (B).

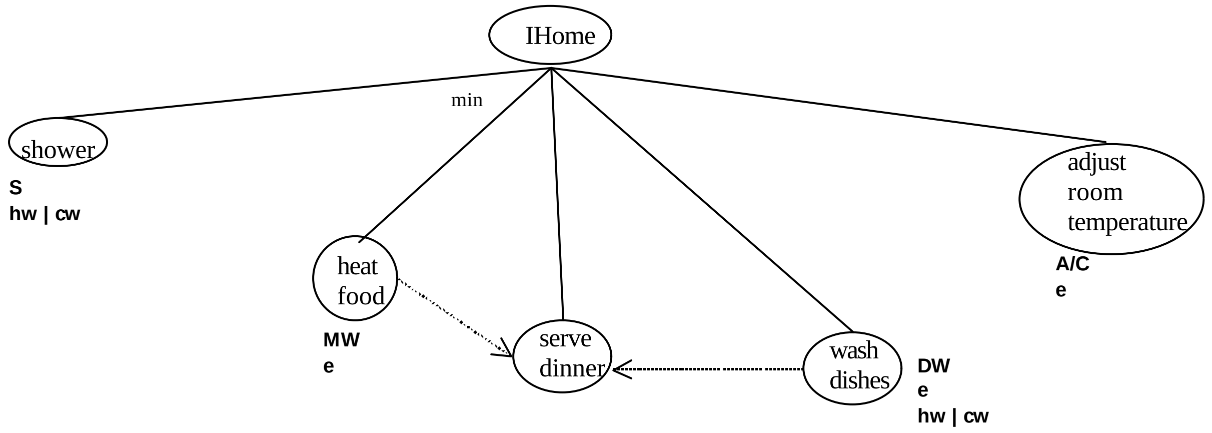


Figure 4: TAEMS Task Structure for the IHome scenario. Dot lines are *enables* relationships. The agents responsible (S, MW, DW, A/C) are shown below each task, and below these, the resources needs (e = electricity, hw = hot water, cw = cold water). Duration, cost and quality distributions are omitted for the sake of clarity.

The main difference between this scenario and the one discussed in the last sections, is that the case here involves resources on which usage agents may need to negotiate. In Decker and Li (1998) and in Lesser et al. (1998a), the authors discuss many issues related to usage of resources, like the use of a sixth GPGP coordination mechanism (resource constraint), and “agentification” of resources, respectively. Here we follow a mixture of both approaches, treating resources as non-exclusive commodities, but also using the resource constraint mechanism, plus the assignment of a priority to each resource request. The designer may opt for turning on or off the resource constraint coordination mechanism. In this section we discuss the former case. For the sake of clarity, resources are represented in Figure 4 not by the use of relationships like *uses* or *limits* but in a way similar to the icons depicted in the figures appearing in Lesser et al. (1998a). We use a notation for each task where below the task itself, we show the agent responsible for the task (bold, capitalized), and, in the next lines, the resources necessary (lower letter). The bars denote that a given resource may admit options like for instance, if there is no hot water, the user may opt for a short cold shower, or the dishes can be washed in cold water as well.

By turning on the resource constraint mechanism, potential overlap of resource usage are discovered, so agents are asked to send a directed bid of the time interval they need the resource, and the local priority they put on the execution of the task. In the implementation of Decker and Li (1998) concerning the sixth mechanism, as they deal with non-sharable resources only, after the bidding, the agent with higher priority gets the resource. We do not restrict the type of resource, thus in our model the agent's bid with higher priority gets as much of the resource it needs, and so on until the resource capacity is over.

To describe how the use of the mechanism works, we show the beliefs set updated by the use of the mechanism (the sets $\mathbf{B_k^L}$ are similar to those depicted in Section 5.1.1 and omitted here):

$\mathbf{B^6_S} = \mathbf{B^L_S}$ + gets{(hot water, 100%)}
$\mathbf{B^6_{MW}} = \mathbf{B^L_{MW}}$ **+** gets{(electricity, 100%)}
$\mathbf{B^6_{DW}} = \mathbf{B^L_{DW}}$ + gets{(electricity,100%),(hot water,100%)}
$\mathbf{B^6_{A/C}} = \mathbf{B^L_{A/C}}$ + gets{( electricity,100%)}

This means that, in this rather standard scenario, resources are enough, hence agents bid for them and get 100% of their needs in all cases.

The conclusion of all examples presented in Section 5.1 is that when the designer of the organization turns on one or more coordination mechanism, this may cause agents to overcoordinate and in some cases, this is associated with prohibitive costs. However, turning off those mechanisms is no solution either, as it is discussed in the next sections. Our work aims at providing agents with tools to recognize both situations via diagnosis.

## 5.2 Turning Off Coordination Mechanisms

### 5.2.1 Mechanism 1 and 4

Now consider the task structure depicted in Figure 1 (slightly modified in relation to the one of Figure 2). Assuming that the designer of the system does not want to pay the overhead of coordination (e.g. because in similar situations the costs did not pay off), then one or more GPGP mechanisms will be turned off. In this section we want to discuss the case where mechanisms 1 and 4 are turned off. The subjective task structures agents hold are still like those depicted in Figure 3. If they go on with the set of local beliefs, they will produce the following local schedules ($LS_k$) and commitments ($C_k$):

$LS_A$ = {M1,(1,2); M2,(3,5)}
$LS_B$ = {M3, (1,5)}
$LS_C$ = {M5, (1,2)}
$C_A$ = {deadline (T1,5)}
$C_B$ = {deadline (T3,5)}
$C_C$ = {deadline (T4,2)}

These schedules fail by not considering the NLEs, which agents are not aware. Method M3 for instance cannot start at time t=1 because it depends on results enabled only after task T2 finishes. Agent B expects the termination time for M3 to be t=5, and thus commits with this deadline for finishing M3. But by this time, M3 is in fact not yet started.

The detection mechanism which will be described later in Section 6, informs agent B that the start time is delayed, so that a diagnosis should be triggered to explain why this happened.

#### 5.2.2 Mechanism 5

To continue with the example above, let us discuss the case where mechanism 4 is turned on but mechanism 5 is off, and focus on the *facilitates* NLE which appears in Figure 5. In this case, A commits to finish T2 at t=5 and task M3 can start at time t=6 and finish by t=11. By the time M3 is starting, agent C has finished the execution of T5. Although agents B and C are unaware of the facilitation power of T5 over M3, it exists and causes a reduction of $\phi_d$ = 80% of M3's duration, that means from 5 units to 1 unit of time, thus M3 finishes by t=7. This unexpected sooner termination is an indication of undercoordination, and may start a diagnosis process as well.

#### 5.2.3 Mechanism 3 (Recognizing Redundancy and Overlapping)

We are particularly concerned with the issue of under or no coordination regarding task structures involving resources, and with of recognition of overlapping of activities. As the example discussed in Sugawara and Lesser (1993, 1998) proves, this is a serious issue, which may end up with the system showing a completely unexpected behavior. To address this issue, one of the extensions of GPGP proposed in Lesser et al. (1998b) is the recognition of overlapping activities by agents. We discuss here the implications of not using this mechanism. Two scenarios where these issues appear will be discussed in this and in the next section.

First, consider the following example pictured in Figure 5 which is abstracted from the WARREN system (Decker e*t al.*, 1997), a multi-agent and multi-user system for management of financial portfolios. WARREN consists of heterogeneous agents that work together in dynamically organized teams to retrieve financial information like stock prices, news, and fundamental data from various locations in the Internet. This scenario deals with agents ($Ag_1$ to $Ag_n$) performing problem-solving activities like information filtering, planning how to obtain a data, or reporting it to the user. In this scenario several agents are accessing one *information source* (shared resource) concurrently for different users. These agents do not coordinate over their resource usage or the relationships among the goals that they are trying to achieve, nor perform an information gathering action to check whether the agents are performing overlapping goals, i.e. that extended version of mechanism number 3 is off.

This is normally acceptable because it is rarely the case that information sources are overloaded, or agents are requesting similar information from them. In fact, the overhead to coordinate these agents would in normal situations decrease agent performance. However, suppose there is a major new story about a company that has just appeared. This may in turn cause a large number of users to request similar information about the company. In this case, specific information sources may get overloaded performing similar accesses of information about the company. For this new situation, introducing agent coordination so as not to overload a resource and to avoid agents doing redundant queries would pay off in improving system performance.

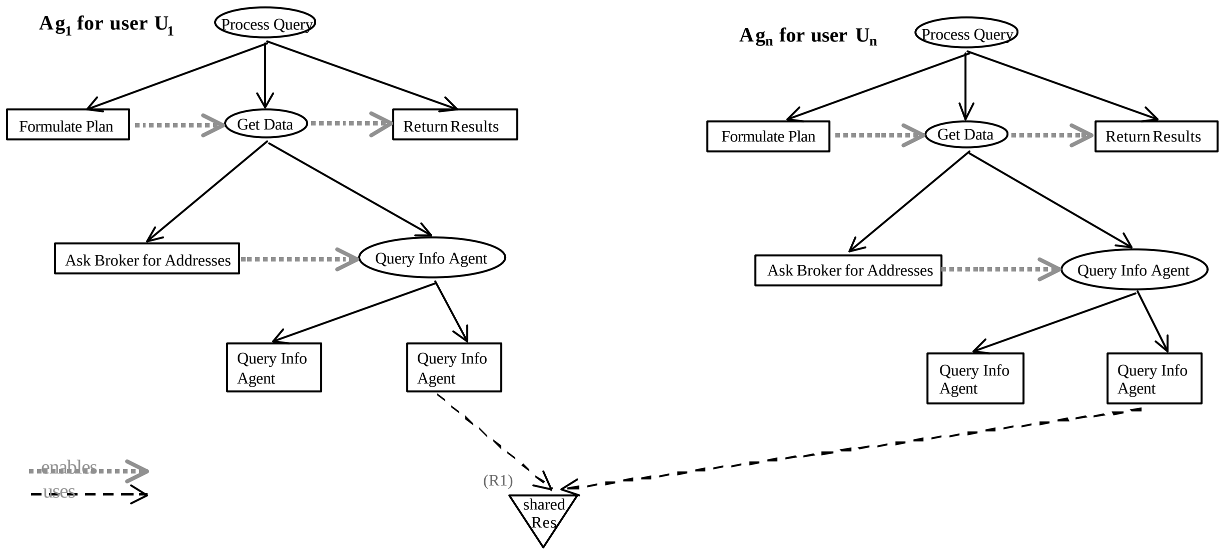


Figure 5: Task Structure for WARREN's Task Agents.

The question we are concerned with is how to detect that our default assumptions (information sources are not heavily loaded, agents are doing little redundant work, etc.) are no longer valid. Knowing that these assumptions are invalid would then allow us to change the coordination strategy of agents, so that they use an enhanced subjective view of the situation in making coordination decisions. This enhanced view will indicate whether there are other agents who are working on the same goal and whether the resources that are being accessed are already overloaded[2]. This ability to recognize redundancy would allow agents to share results of a query, so that the number of queries to the information source could be significantly reduced.

The first step to recognize redundancies is to analyze the type of the failure detected in a symptom-driven fashion. When the failure is related to a resource overload, this increases the likelihood that the agent is working with an incorrect view of the objective task structure. It might pay off to turn on one or more coordination mechanisms not previously used to construct the enhanced view we talk above. We return to this example later in Section 7.

### 5.2.4 Mechanism 6

The last example we want to discuss is a variation of the one presented in Section 5.1.4 above. Once the designer of that organization recognizes the high cost of the coordination versus the low or null benefits of it, s/he may opt for a no-coordination version. Assume that the organization depicted in Figure 4 appears (with a low frequency) in its slightly modified version shown in Figure 6. This is the party version, in which the user desires everything ready by 6 p.m. in order to serve the dinner for the guests.

[2] We also would recognize the reverse, namely that the assumption there was a lot of redundant agent activity is no longer valid.

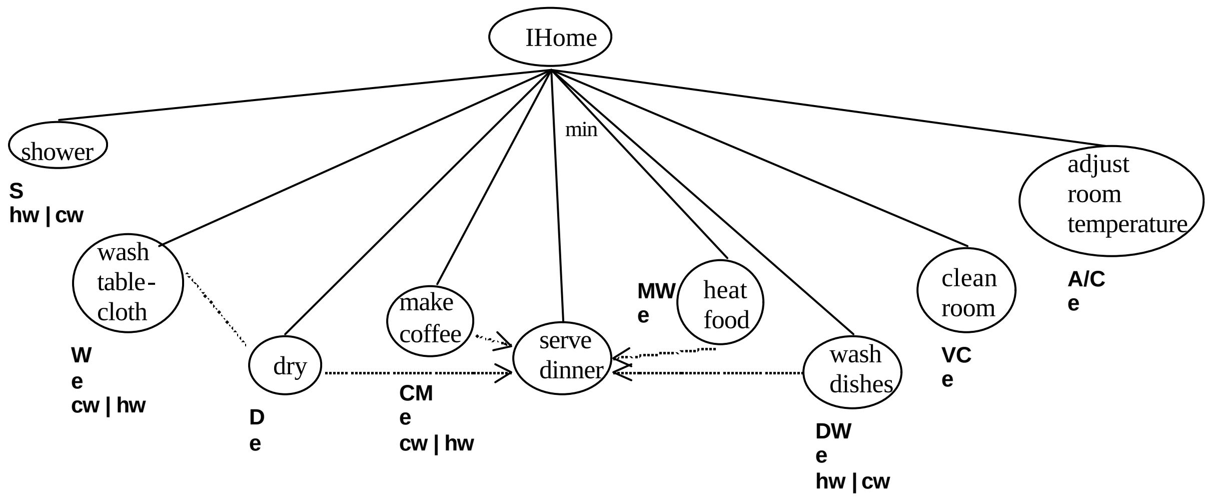


Figure 6: Party Version of the IHome Scenario.

New agents appear in this scenario: washer (W), dryer (D), coffee machine (CM), and the vacuum cleaner (VC). Resources are the same, namely electricity, cold and hot water. Serving the dinner involves activities that may need to be coordinated and that use more units of the existing resources. The fact that the coordination mechanism number 6 is off means that agents do not bid for resources; they just schedule activities as these resources were not constrained. What happens is that during the actual execution of methods and tasks[3], each agent requests the quantity necessary of resource, and either gets it or not. Hence, the presence of a clause like

$$\mathbf{B}^{6}_{\mathbf{MW}} = \mathbf{B}^{\mathbf{L}}_{\mathbf{MW}} + \text{gets}\{(\text{electricity}, 0\%)\}$$

indicates that the assumption of resource unconstraint is not valid anymore, which should trigger the diagnosis process.

This example shows that even if there is no redundancy, the recognition of a resource overload could cause an agent to search for other causes of a failure in the problem-solving process. This is a case where the agent could either trade-off spending more time and/or money to use the desired resource to perform the task, or perform it with less quality but quickly. Examples are to prepare coffee with instant powder instead of grinding beans, or to wash the table-cloth using cold cycles to wash and/or rinse.

Similar scenarios could as well occur because of hardware failures of information sources or the intrusion of a rogue agent (or a software bug) into the system that accesses information from the information sources at a very high rate.

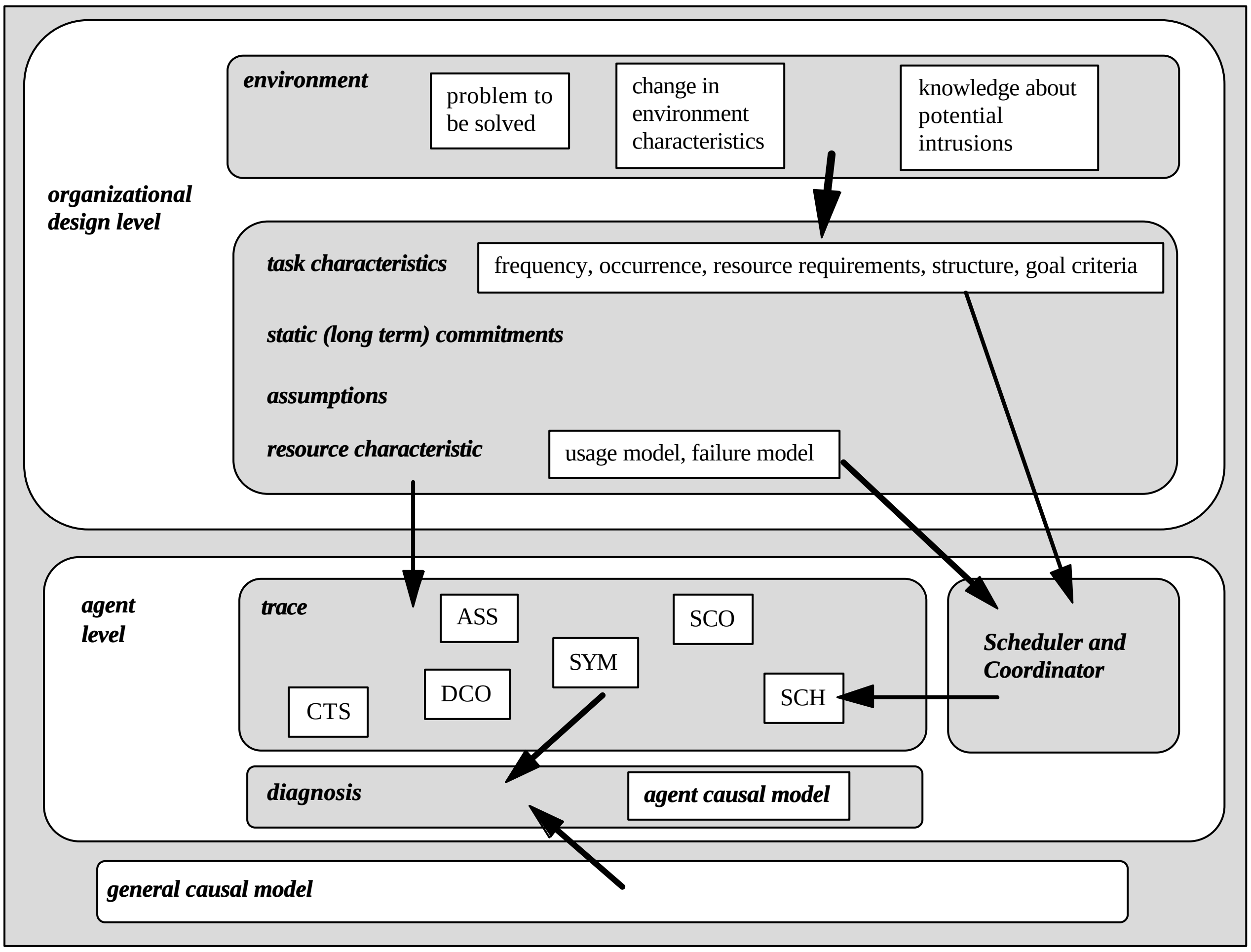
## 5.3 Summary

The sections above discussed many examples where the type of coordination chosen by setting GPGP coordination mechanism on or off in some possible ways, fails. Many other scenarios exist and our main goal is to develop a tool which tackles any of this scenarios via a diagnosis which is general enough since it works on the TAEMS and GPGP domain-independent frameworks. The next section presents our proposal for such tool.

[3] The tasks depicted in Figure 6 are all decomposable. See for instance Lesser et al. (1998a) for the task structure of the CoffeMachine agent.

# 6 Domain-Independent Diagnosis of Problem-Solving Failures

## 6.1 Proposed Architecture



environment
problem to be solved
change in environment characteristics
knowledge about potential intrusions
organizational design level
task characteristics
frequency, occurrence, resource requirements, structure, goal criteria
static (long term) commitments
assumptions
resource characteristic
usage model, failure model
agent level
trace
ASS
SCO
SYM
DCO
SCH
CTS
Scheduler and Coordinator
diagnosis
agent causal model
general causal model

Figure 7: Architecture of the Diagnosis Module

This section discusses our domain-independent framework to detect changes from an expected behavior in the problem-solving system (which may point to potential intrusions in the environment), diagnose the causes for this incorrect behavior, and, if it is the case, isolate faults caused by inappropriate coordination or changes in environment characteristics. We see this process having as input:

- a database containing the current organizational design including its assumptions concerning the environment;
- a trace of the problem-solving process representing the activities executed, their results in terms of performance characteristics (duration, quality, and cost), messages sent, commitments generated, symptoms detected, and resource usage[4].

Figure 7 shows the three components in the architecture. The **organizational design level** is composed of two parts: an acquisition of environmental, long-term knowledge as discussed in Section 4.1, and the output of the design process itself, namely, a description of the organization and the role of each agent in it (including

[4] We may also include some meta-level measures of the current state of agent activities such as those developed in Nagendra-Prasad and Lesser 1996.

description of alternative methods to accomplish tasks), as well as effort necessary, commitments to be honored, and assumptions made. We do not tackle here the process of design itself.

The **agent level** is composed of a coordination module, a scheduler, a module to keep a trace of the activities, and a diagnosis module. Details on the scheduler can be found in Decker (1995) and in Wagner et al. (1997). The trace and diagnosis are detailed below. The third level is the **causal model**, which shows the relation between symptoms and explanations for them. Notice that, at the agent level, there is also an instance of the causal model, which represents the agent's local perception of that more general model.

The **trace** (recording of agent behavior) is analyzed and symptom for an eventual abnormal behavior is detected. Agents then construct a local view of the actions leading to that situation. In case there is a need to exchange views with diagnosis component by other agents, a communication may take place. The process of **diagnosis** is triggered by generic symptoms, which reinforces the domain-independence of the framework. For instance, diagnosis is triggered when agents notice that either a resource used to accomplish the task, or a processor involved in the planning of the problem-solving is overloaded. The other main trigger conditions are commitments not being kept, and an unexpected result of task execution, such as its level of resource usage, quality of its results, and its duration or cost of its execution. In each of these cases, the system analyzes the symptom, finds an acceptable explanation, and modifies the parameters that control the agent behavior (part of the organizational design)[5].

We make a series of assumptions concerning the organizational design, the trace, and the diagnosis module. We assume that the activities of the agents in the environment are represented in a TÆMS task structure, that some components of the system are invulnerable to break-ins (e.g. the coordination and the diagnosis modules are protected by firewalls or run in different processors so that they are not corrupted), and that we have a means to keep a trace of these activities, as well as of the usage of resources available (through a resource monitor). Although we do not have to necessarily monitor all resources constantly, that we keep statistics of the high-level tasks arising in the environment. We treat both software bugs and lying agents in the same way. Also, we assume that the problem-solving component cannot decide if the description of the quality of results passed to it is accurate. On the other hand, in certain situations the diagnosis component may initiate a problem solving task (based on the known task structures) to verify the quality characteristics of an intermediate result.

Finally, we also make the assumptions that there is often multiple ways of deriving a result, alternative resources that can be used by a method in their execution, and that some methods have the capability, if so asked, to verify the quality of their input or the characteristics of an intermediate result. This aims at an auto-diagnosis process by the system itself through the use of its own problem solving capabilities.

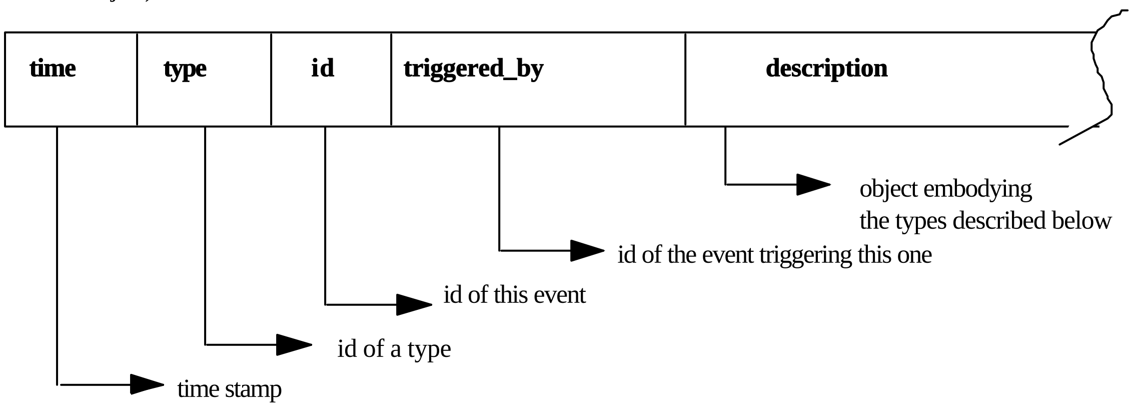
### 6.2 Trace

The monitoring of parameters like resource usage and task execution characteristics allows the determination of whether a change in behavior is statistically significant or not. If it is, a diagnosis process is triggered; otherwise

[5] Also symptoms such as resource underload, and wrong coordination leading to no commitments made.

the models (e.g. duration, quality, and cost distribution) have to be further monitored over time to check their validity.

• event (format of object):


time
type
id
triggered_by
description
object embodying the types described below
id of the event triggering this one
id of this event
id of a type
time stamp

• types of events:

- MSG (any kind of msg. exchanged between agents.)
- NTS (non-conditioned task structure)
- CTS (conditioned task structure for each agent)
- ASS (assumption(s) made)
- SCO (static commitments)
- DCO (dynamic commitments, both local and non-local)
- SCH (schedule)
- EXE (execution of a plan)
- SYM (symptom)

Figure 8: Structure of a Trace Event

The trace recording the events associated with the behavior of agents can contain objects of the following types:

- the task structure presented to the scheduler as a result of conditioning the task schedule generated by the problem solver with organization knowledge, and the task structure prior to this conditioning[6]
- the assumptions made (e.g. about resource availability, process load, and frequency of tasks arising in the environment[7])
- the current organizational structure, the coordination strategy (including status of GPGP coordination mechanisms), and reasonable ranges for alternative organizational models (in case one needs to test the validity of a revised model of the environment)
- messages exchanged during coordination or execution of tasks; this includes the updating of beliefs which occur for instance after a coordination mechanism detects non-local views
- commitments made by agents
- schedule for the execution of methods
- the actual characteristics of method execution
- symptoms
- messages

[6] This corresponds roughly to the subjective and objective task structures, respectively.
[7] Reasonable ranges for models allow us to recognize that an intrusion may have occurred.

Each trace event (shown schematically in Figure 8) involves the event time, type, ID, and the ID of the event that had triggered it. Using the trace it is possible to locate a symptom and those portions of the task structure which relate to the symptom, from the characteristics of method execution and resources usage, to the assumptions behind the construction of that particular task structure. For the sake of clarity, information such as ID and triggering object will be omitted of the descriptions of traces depicted in the next figures; only time, type of event, and description will be shown.

To understand the trace information, we will present an example of an agent's trace related to the WARREN scenario discussed in Section 5.2.3. The detailed non-conditioned (objective) and conditioned (subjective) views for the example, generated by the organizational designer, are shown in Figure 17, and in Figure 18, respectively. However, here we focus on the task structure decomposed among 3 agents (A, B, and C) for each user (previously we have considered $Ag_1$ to $Ag_n$ responsible for the whole task structure for each user).

```
time   type description
------------------------------------------------------------------
050    MSG  msg. to task assessor  to solve a problem
150    NTS  taems1a)
250    ASS  defaultAssumption
260    SCO  static non-local commitment
270    CTS  [AgentA,taems2]b)
285    DCO  [AgentA,M1,50]
290    SCH  [M1,1,50]
301    EXE  M1 Started Execution at AgentA
350    EXE  M1 Finished Execution at AgentA
```

Figure 9: Standard Trace for Agent A. a) Non-Conditioned Task Structure depicted in Figure 17. b) Conditioned Task Structure depicted in Figure 18.

When there is no abnormal situation, the traces for agent A, B, and C look like those depicted in Figure 9, Figure 10, and Figure 11 respectively. After the organizational designer receives a request to generate a design for a given problem, a non-conditioned task structure for it is created (object *taems1* in those cases), then assumptions are made (here a default assumption is used), long-term commitments are established, and a conditioned task structure is created for each agent. After, schedules and dynamic commitments, both local and non-local, are established at agent level. The rest of the trace concerns the execution of the methods and tasks, and will be discussed in more detail in Section 7, when traces of abnormal behavior are depicted.

```
time   type description
----------------------------------------------------------------
050    MSG  msg. to task assessor  to solve a problem
150    NTS  taems1
250    ASS  defaultAssumption
260    SCO  static non-local commitment
270    CTS  [AgentB,taems3]
280    DCO[a] [AgentB,$M_2$,100]
285    DCO[b] [AgentB,$T_1$,200]
290    SCH  [[$M_2$,51,100],[$M_3$,101,150],[$M_4$,151,200]]
351    EXE    $M_2$ Started Execution at AgentB
400    EXE    $M_2$ Finished Execution at AgentB
401    EXE    $M_3$ Started Execution at AgentB
450    EXE    $M_3$ Finished Execution at AgentB
451    EXE    $M_4$ Started Execution at AgentB
500    EXE    $M_4$ Finished Execution at AgentB
```

Figure 10: Standard Trace for Agent B. a) Local Commitment. b) Non-Local Commitment.

```
time   type description
-----------------------------------------------------------------
050    MSG  msg. to task assessor  to solve a problem
150    NTS  taems1
250    ASS  defaultAssumption
260    SCO  static non-local commitment
270    CTS  [AgentC,taems4]
290    SCH  [$M_5$,201,250]
501    EXE  $M_5$ Started Execution at AgentC
550    EXE  $M_5$ Finished Execution at AgentC
```

Figure 11: Standard Trace for Agent C.

## 6.3 Causal Model

In order to understand the trace of agents problem-solving, we introduce a causal model (CM). The CM is a graph object that maps symptoms to explanations, constructed from specifying the trace. It is intended to be as general as possible, in order to analyze various organizations described using TÆMS. Each agent constructs a particular instance of the general CM, in which the nodes refer to particular objects of the agent's trace kept during the process of problem-solving, like the task structure (TS) such as methods, tasks, resources, or NLE's. Also, some nodes and edges are trimmed off because they do not apply to the particular problem the agent is solving.

The algorithm I for creating the agent's instance of the general CM is shown in Figure 12.

```
for each agent A_i:
   create the agent CM by trimming off those portions which do not apply[8]
   while there is a trace event for A_i
      read the trace event
      if event is a symptom
         find symptom in the agent CM
         generate a DiagWeb containing the symptom and its ancestors[9]
         while there is a node in the DiagWeb
            if node is an explanation for  the symptom
                add its ancestors to the DiagWeb
         end
         if no explanation is found, the CM is incorrect[10]
   end
```

Figure 12: Algorithm I (searching the trace for symptoms)

Once a symptom is detected in the trace, there is a set of possible explanations for it. These sets are domain-independent and will work on any problem-solving system which meets the assumptions discussed before. The main classes of symptoms identified are unexpected *duration*, *quality*, and *cost* of a method execution, resource overload or underload, and commitments not being met. The latter class of explanations allows us to represent symptoms involving unexpected performance characteristics of deeper level tasks.

Some explanations discussed below are close related to intrusion. If an attack happens, delays in method execution, lowering of quality achieved, usage of resources and ultimately cost of execution of methods are very likely to occur. Within the framework, it is possible to trigger a learning process to try to understand the pattern of the changes, to verify why the change happens, or whether an agent is somehow cheating.

### 6.3.1 Explanations for Symptoms Related to Completion of a Method

When the symptom is that the *completion of a task/method is delayed*, say T, the possible explanations, as they appear in the causal model (Figure 13), are:

- Task-related (explanations which are related to the characteristics of the task):
  - completion time is delayed: the scheduled time for the task T to finish is delayed, which means that either T started later than expected, or its duration is longer than scheduled
  - start time delayed: either T is preempted by another task, or the completion of a previous task, say T', is delayed, or results from an enabling task (not necessarily T') are delayed
  - previous (scheduled) task completion is delayed: a task scheduled to be finished before (like T') did not finish on time, in which case another diagnosis reasoning for T' is initiated (see the explanation for the first node of this list)
  - preemption of task by scheduler: T is delayed because a higher-priority task was put ahead of that one; the designer of the organization must analyze the task structure to find out why this happened
  - duration is longer than expected: if the duration of T takes longer than scheduled, the possible explanations are as below

[8] Such as NLE's or resources not appearing in the task structure of the particular problem being solved
[9] See algorithm II
[10] Or a combination of symptoms must be checked

    - the statistical variance of the distribution of the duration falls in an unacceptable range: all tasks admit a variation in the duration, which is given by a statistical distribution
    - the model for duration distribution is inappropriate: it is necessary to collect more information through further observations, in order to learn the correct distribution
    - insufficient local processing capacity: if T takes too long because the capacity of the agent to perform it is not as expected, the assumptions associated with processor usage must be revised by the designer of the organization
    - incorrect model of resource usage by specific tasks: the model of usage of a given resource by T is not as expected
    - incorrect resource model: the designer has an incorrect model of the resource characteristics when the organization is planned
    - unacceptable statistical variance of resource usage: like the duration of a task, resource usage is also based on a statistical distribution; variations on it might be unacceptable
    - resource computation does not consider resource usage, either because the access to resource was not considered in the duration of the method, or because the resource was not monitored
    - no coordination because of incorrect conditioned view (e.g. resource usage was not considered)
- Resource-related (explanations which relate to usage of resource):
    - incorrect model of resource usage: the model of *overall* usage of T (not necessarily and not only by T) is not as expected (e.g. other agents lied or were unaware of their actual usage)
    - unacceptable statistical model of resource load: similar to other statistical models
- Non-Local Relationship-related (explanations which are related to the existence of NLE relationships between tasks):
    - enabling task delayed: it is necessary to trace back in the task structure, find out which is the agent responsible for delivering these results, and explain why these are delayed
    - incorrect model of facilitation power: this is also a case in which the agent responsible has to be contacted; it might be that it was lying about this power
    - incorrect model of facilitation quality: similar to facilitation power
- Coordination-related (explanations which are tied to the coordination type):
    - no coordination: the designer opted to ignore coordinating with other agents, i.e. this decision appears as an assumption (see assumption-related below); agents need to understand which assumptions were the basis for the organizational designer deciding that it was not necessary to coordinate over resources in this specific situation; after agents models have to be modified to include the coordination in that particular situation causing the symptom to arise
    - incorrect coordination: the coordination with other agents is incorrect; for instance, incomplete coordination does not include and/or explore information about correlated usage like redundancy, overlapping, or possible exchange of favors between agents (von Martial, 92); the model needs to be enhanced to include these relationships
    - coordination with incorrect information: the designer possesses incorrect information when deciding the type of coordination

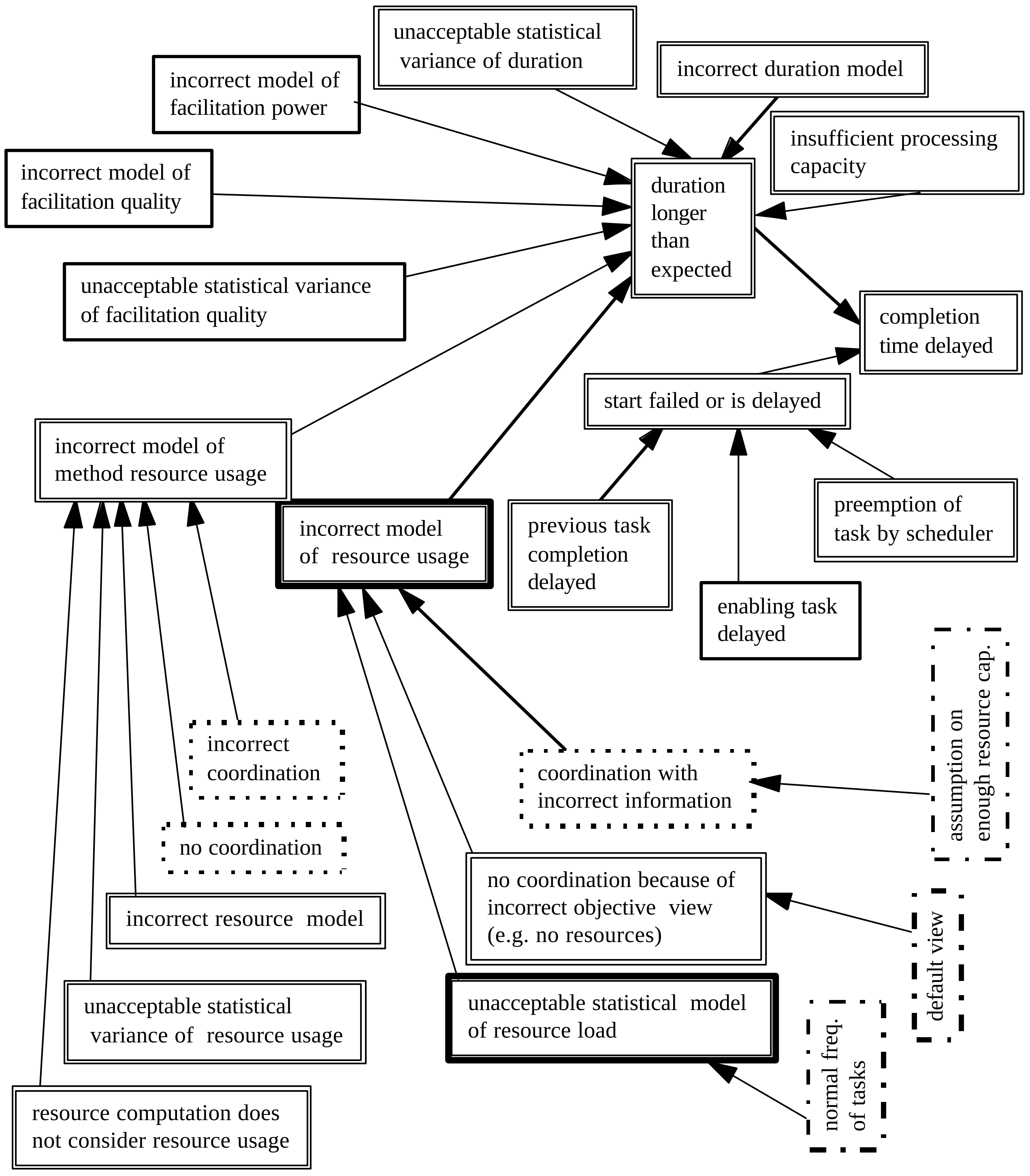

unacceptable statistical variance of duration
incorrect duration model
incorrect model of facilitation power
incorrect model of facilitation quality
insufficient processing capacity
duration longer than expected
unacceptable statistical variance of facilitation quality
completion time delayed
start failed or is delayed
incorrect model of method resource usage
incorrect model of resource usage
previous task completion delayed
preemption of task by scheduler
enabling task delayed
assumption on enough resource cap.
incorrect coordination
coordination with incorrect information
no coordination
incorrect resource model
no coordination because of incorrect objective view (e.g. no resources)
default view
unacceptable statistical variance of resource usage
unacceptable statistical model of resource load
normal freq. of tasks
resource computation does not consider resource usage

Figure 13: Causal Model for Explanations Related to Duration of Method. Explanations (boxes) are related to: resource (bold line), task (double line), coordination (dot line), NLE (single line), assumption (dash and dot)

- Assumption-related (explanations which relate to assumptions made):
  - enough resource capacity: it is assumed that resources are unconstrained
  - default view: the organization assumed a default view for the use of resource, which might not be valid anymore; normally, it is the case that agents do not want to pay the overhead of constantly finding out the update states of a resource and then computing a new distribution for expected duration
  - normal frequency of tasks: it is assumed that the frequency of tasks arising in the environment adheres to some known distribution.

### 6.3.2 Explanations for Symptoms Related to Resource Overload

A *resource overload*, which may occur due to either unpredictable activities (using that resource), lack of resource model, an incorrect or incomplete model of resources (i.e. one which does not call for coordination), or a failure to account for the use of that resource. This kind of symptom is likely to occur together with others since lack of or inappropriate coordination over resource usage or delay in information exchanging are associated with methods taking too long or incurring more cost than expected.

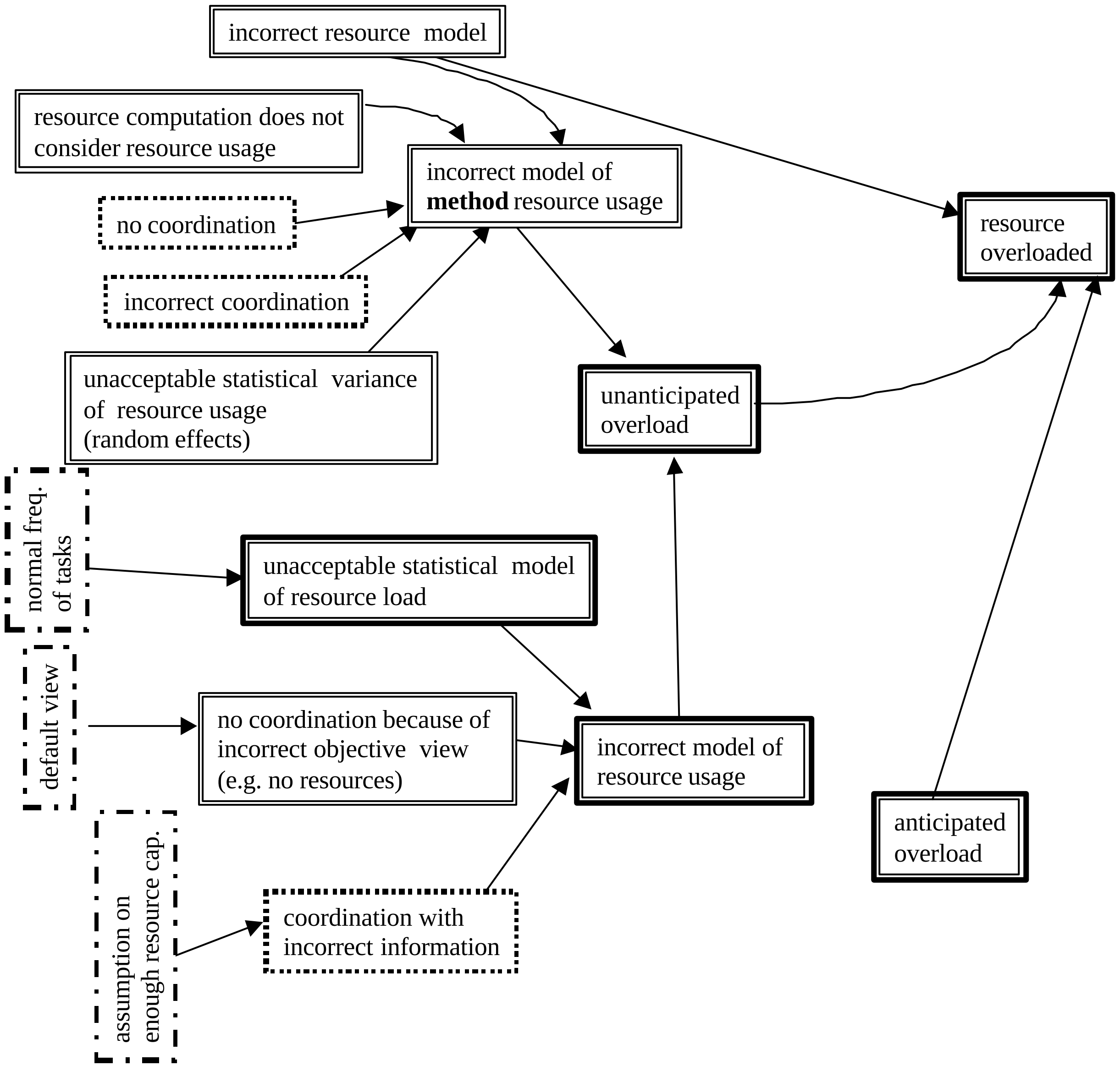

incorrect resource model
resource computation does not consider resource usage
incorrect model of **method** resource usage
no coordination
incorrect coordination
unacceptable statistical variance of resource usage (random effects)
resource overloaded
unanticipated overload
normal freq. of tasks
unacceptable statistical model of resource load
default view
no coordination because of incorrect objective view (e.g. no resources)
incorrect model of resource usage
anticipated overload
assumption on enough resource cap.
coordination with incorrect information

Figure 14: Causal Model for Explanations Related to Resource Overload

The possible explanations (most already discussed in the last section, thus only cited here), as depicted in Figure 14 are:

- Resource-related:
  - anticipated overload: the designer predicted a potential overload but opted not to consider it
  - unanticipated overload: caused by either an incorrect model of method resource usage, or by an incorrect general model of resource usage

    - the model of resource usage is incorrect
    - unacceptable statistical model of resource load
- Task-related:
    - incorrect model of method resource usage (the *uses* relationship)
    - incorrect resource model (the *limits* relationship)
    - unacceptable statistical variance of resource usage
    - resource computation does not consider resource usage
    - no coordination because of incorrect non-conditioned view (e.g. no resources in the model)
- Assumption-related:
    - the assumption of frequency of method execution is incorrect, and this has to be reported to the organization designer
    - availability of resource is different than assumed
    - default view is used
    - frequency of tasks is normal
- Coordination-related
    - incorrect coordination
    - no coordination over resource usage
    - incorrect coordination over resource
    - coordination with incorrect information

### 6.3.3 Explanations for Symptoms Related to Quality of a Task

The explanations for quality related symptoms in the causal model (Figure 15) are:

- Non Local Relationship-related:
    - incorrect model of facilitation power
    - incorrect model of facilitation quality
    - unacceptable statistical variance of facilitation quality: the quality accrued is not as predicted by the statistical distribution
    - no enabling task: the organizational design is incorrect so that a task keeps waiting for enabling results which do not arrive, possibly because the agent supposed to deliver them ignored an enabling relationship (commitment that would cause an agent to send the results) and thus fails to report the results associated with it
- Task-related:
    - preemption by scheduler
    - insufficient local processing capacity
    - quality lower than expected in a previous task: this finishes with quality lower than expected, and cannot keep the commitment on the quality to be delivered
    - low expected quality: caused by an agent lying about its capabilities
    - incorrect model for quality distribution: the statistical distribution the designer is working with might be incorrect; this has to be verified via learning

- unacceptable statistical variance of quality

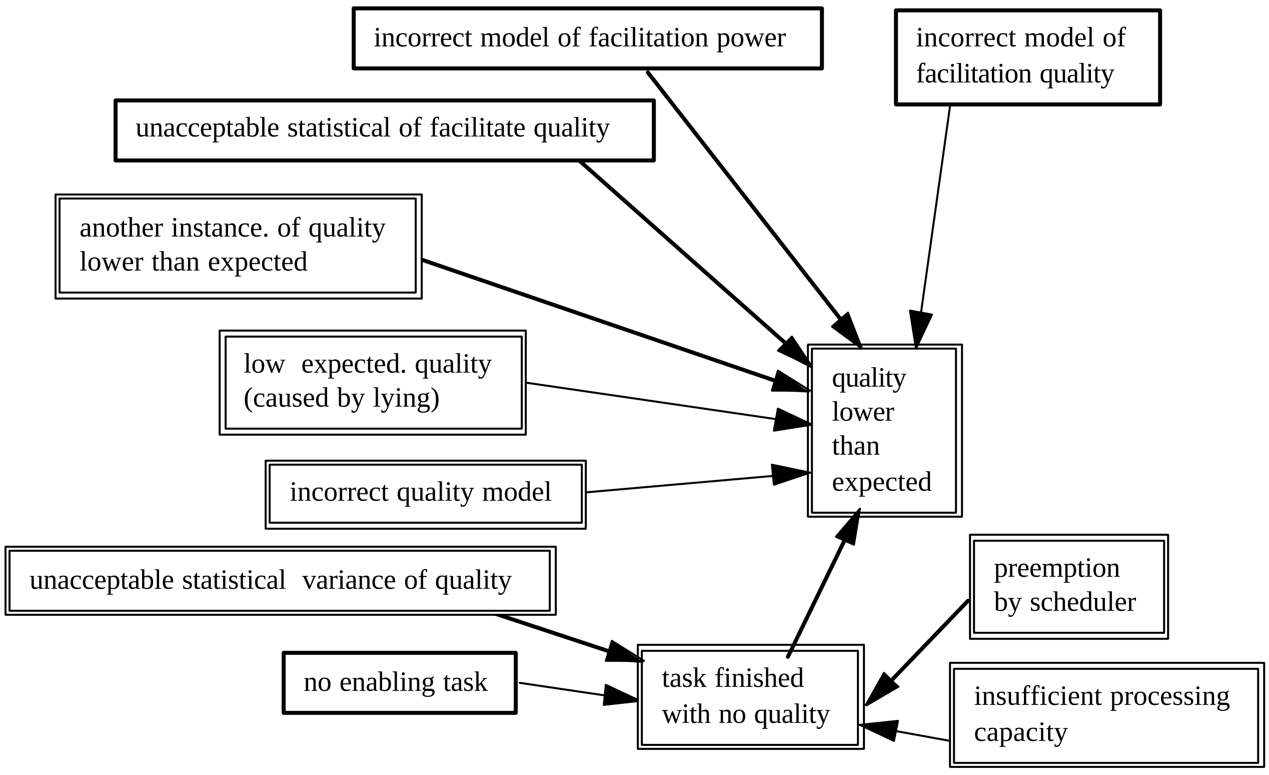

incorrect model of facilitation power
incorrect model of facilitation quality
unacceptable statistical of facilitate quality
another instance. of quality lower than expected
low expected. quality (caused by lying)
incorrect quality model
quality lower than expected
unacceptable statistical variance of quality
preemption by scheduler
no enabling task
task finished with no quality
insufficient processing capacity

Figure 15: Causal Model for Explanations Related to Quality Lower Than Expected

## 6.4 Diagnosis

The process of diagnosing starts at agent level when a symptom is detected. First, a local view of the problem is constructed, using the events retrieved from the trace: the symptom, the portion of the task structure tied to the method or task which is related to the symptom (e.g. a delay in the execution of a method), the coordination strategy used, the assumptions behind it, messages exchanged, etc.

Once the diagnosis component of the agent constructs its local causal view, it has to analyze the need for exchanging of views with diagnosis components in other agents. This is the case if, for instance, a subtask that is related to the symptom is part of a large task that is distributed among multiple agents, or the subtask requires information for its execution from a task being executed by another agent, or if some non-local resource is accessed. In this case the agent will send a request for other agents possibly involved, to start a diagnosis themselves (if this is not already the case), and share their local views. In either case, the local or extended view starts the diagnosis reasoning. This is based on the causal model held by the agent.

```
(using the agent CM and the symptom found[11])
while there is a node in the agent CM
   if node relates to the object described in symptom
      instantiate the corresponding nodes with object[12]
end
```

Figure 16: Algorithm II (generation of the agent Causal Model upon finding a symptom)

Besides the sets of symptoms, explanations, and actions, there are sets of analysis associated with each explanation, which aim at deciding whether it is a valid explanation for the given symptom or set of symptoms, in a particular situation. For instance to verify the explanation “the expected results from a facilitate NLE are delayed”, there must be determined whether i) the delayed method is linked to another agent through a facilitates NLE, and in case the link exists, whether ii) the results expected are delayed. Basically each explanation is associated with a set of at least one type of prescribed verification procedure.

# 7 Diagnosis of Incorrect Behavior in the WARREN

## 7.1 Scenario

Our first attempt towards testing the domain-independent framework was to model those diagnosis problems posed by Sugawara and Lesser (1993, 1998) by representing the network scenario (LODES agents) discussed in terms of TÆMS task structures. Since they explicitly represent the interdependence between subtasks, it provides a way to decompose the structure in constituent components, which are then easier analyzed as for possible inappropriate behavior. By analyzing the components it is possible, for instance, to determine which methods or resources are responsible for the changes. This attempt was successfully concluded with Sugawara’s first and second examples being represented as TÆMS task structures, in which we add a model of resources usage by the methods and assumptions on resource availability, and acceptable range for the model’s parameters.

We are currently modeling similar problems from the WARREN scenario as the one already discussed in Section 5.2.3. There are many similarities between LODES and WARREN scenarios in what concerns diagnosis of problem-solving failures. One typical example is the assumption that *normally* there is enough communication and computational resources available on the environment to sustain a certain level of non-coherent behavior, so it is not necessary to pay the overhead of discovering and implementing each time the appropriate coordination strategy. Those situations in which this is unacceptable need to be identified and the appropriate coordination strategy for them discovered. If, for some reasons, the volume of accesses to the shared resource increases beyond a threshold, requests to it will be queued and this might cause the duration of methods accessing them to be delayed: when this happens, agents start a diagnosis process to discover why a symptom was triggered that is related to the execution of a method taking too long. The relevant part of the trace is then recovered for further analysis. Since this particular symptom is not directly related to a need of exchanging local views with other agents, the diagnostic process starts in that agent according to our diagnosis algorithm.

[11] The symptom event retrieved from the trace has information relating it to some TS objects like for example: SYMPTOM “duration of method <method_object> longer than expected“

[12] For example the node “start time delayed” of the general CM, turns “start time delayed at <method_object>“

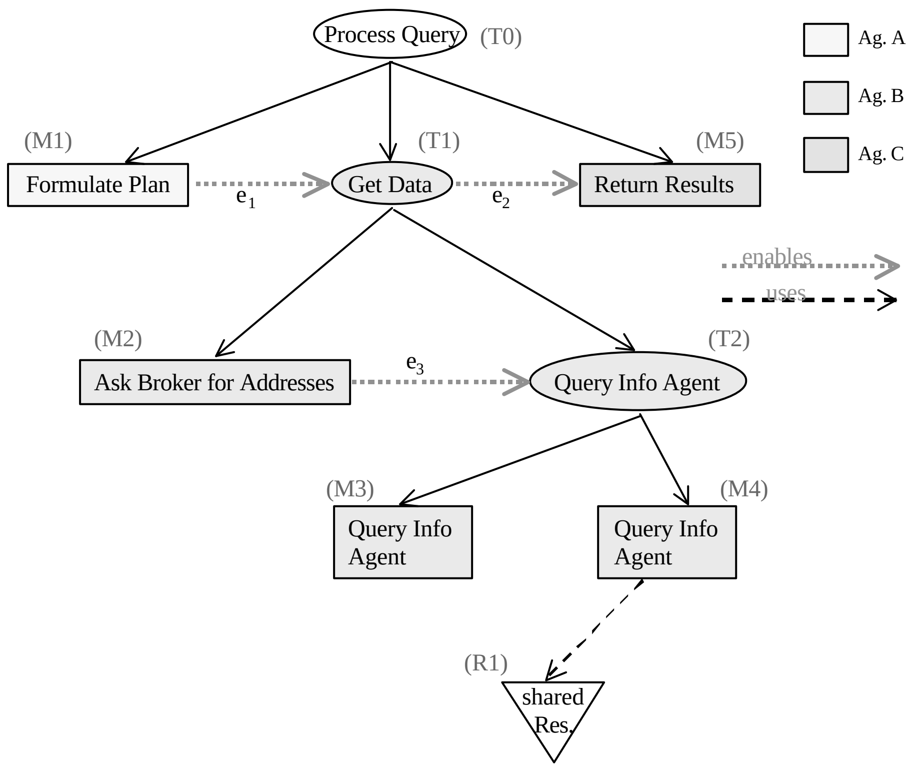


Figure 17: Task Structure of the WARREN scenario (Query of One User)

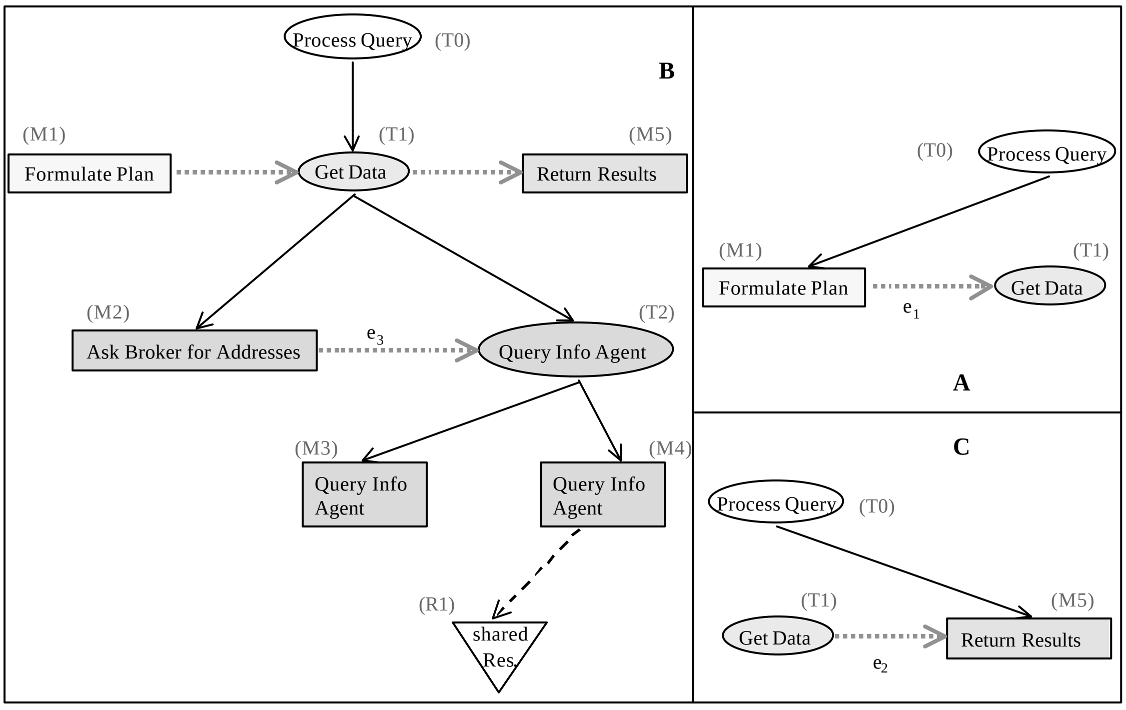


Figure 18: Subjective Task Structures for Agents A, B, and C.

## 7.2 The Situation in Which the Assumptions Hold

Consider the example depicted in Figures 17 and 18. If the frequency of access to resource $R_1$ is normal, then only a few agents are likely to be concurrently accessing it, thus $R_1$ is not likely to become overloaded, and the method $M_4$ completion time will be as expected. If no other abnormality occurs, the trace of the problem-solving looks like the one depicted in Figure 9. All methods start and finish as scheduled, and no symptom appears.

## 7.3 Changes in the Environment

When there are changes in the environment and/or the nature of the task changes, it is likely that the problem-solving process needs to adapt to these changes. This section describes how the scenario may change, why, and how a diagnosis may help.

The following list contains the mostly likely changes in the environment affecting the problem-solving process built in the agents:

1. with too many agents accessing a given resource, it gets overload
2. a processor in one of the agents fails, causing communication problems with the other agents, or problems in processing the task itself
3. a task with its subtasks/methods fails to begin executing, as for example $T_1$ or $T_2$
4. as above, but here an alternative task to the failed one is then selected (alternate tasks are not shown in Figure 17, but we assume that they exist)
5. a task with its subtasks/methods, which should co-occur with another one, fails to appear in the environment
6. a task gets preempted by another
7. a method fails, as a consequence of a software bug or an intrusion
8. a method uses a resource above the level forecasted due to a software bug or an intrusion
9. two methods take too long, both within the statistical variance, but their combined effect causes further delays
10. the transmission of results from an enable relationship (e.g. $e_1$) is slightly delayed, and $M_3$ or $M_4$ takes longer, so that the overall deadline of $T_2$ is not met
11. a task previously assumed as not important is delayed enough to be worth coordinating over

The next subsections show examples of some of the situations described above. Although the symptom detected in the trace is the same in almost every case, namely “duration of method $M_4$ longer than expected”, the causes behind it can be quite different.

### 7.3.1 Diagnosing Resource Overload

Figure 19 shows a trace of agent B activities. At time t=520, a symptom related to the execution of method $M_4$ appears. This triggers the use of the Algorithm I, which constructs a causal model based on the agent view of the problem. The node in the causal model which matches the symptom is found, and the portion containing this node and its ancestors is further revised so to trim off nodes which do not apply. For instance, looking at Figure 13, the node “duration longer than expected” is further instantiated with the method object reported in the symptom message of the trace (`_m`$_4$). Table 1 shows how the ancestors of this node are instantiated.

| Node | Object Instantiated |
|---|---|
| duration longer than expected (object) | _method-$m_4$ |
| insufficient processing capacity (object) | _agentB |
| incorrect duration model (object) | _duration-model-for-method-m4 |
| unacceptable statistical variance of duration (object) | _statistical-model-for-method-m4 |
| incorrect model of facilitation power (object) | not instantiated: no facilitation to method |
| incorrect model of facilitation quality (object) | not instantiated: no facilitation to method |
| unacceptable statistical variance of facilitation quality (object) | not instantiated: no facilitation to method |
| incorrect model of method resource usage (object) [a)] | _model-usage-of-r1-by-m4[b)] |
| incorrect model of resource usage (object) | _model-usage-of-r1 |
| unacceptable statistical variance of resource load (object) | _statistical-model-for-load-of-r1 |
| no coordination/incorrect objective view (object) | _coordination-assigned-for-agentB[c)] |
| coordination with incorrect information (object) | _coordination-assigned-for-agentB |
| assuming normal frequency of tasks (object) | _assumption |
| assuming default view (object) | _assumption |
| assuming enough resource capacity (object) | _assumption |

Table 1: Instantiation of Agent B Causal Model. a) Ancestors of this node not further detailed. b) Objects $r_1$ and $m_4$ come from the symptom message. c) Object embodied in the assumption object.

```
time   type description
--------------------------------------------------------------
050    MSG  msg. to task assessor  to solve a problem
150    NTS  taems1
250    ASS  defaultAssumption
260    SCO  static non-local commitment
270    CTS  [AgentB,taems3]
280    DCO [AgentB,M2,100]
285    DCO [AgentB,T1,200]
290    SCH  [[M2,51,100],[M3,101,150],[M4,151,200]]
351    EXE    M2 Started Execution at AgentB
400    EXE    M2 Finished Execution at AgentB
401    EXE    M3 Started Execution at AgentB
450    EXE    M3 Finished Execution at AgentB
451    EXE    M4 Started Execution at AgentB
520    SYM    DurationLongerThanExpected (M4)
580    SYM    ResourceOverloaded (R1)
600    EXE    M4 Finished Execution at AgentB
```

Figure 19: Trace Containing Symptoms Duration Longer Than Expected and Resource Overloaded (from Agent B execution)

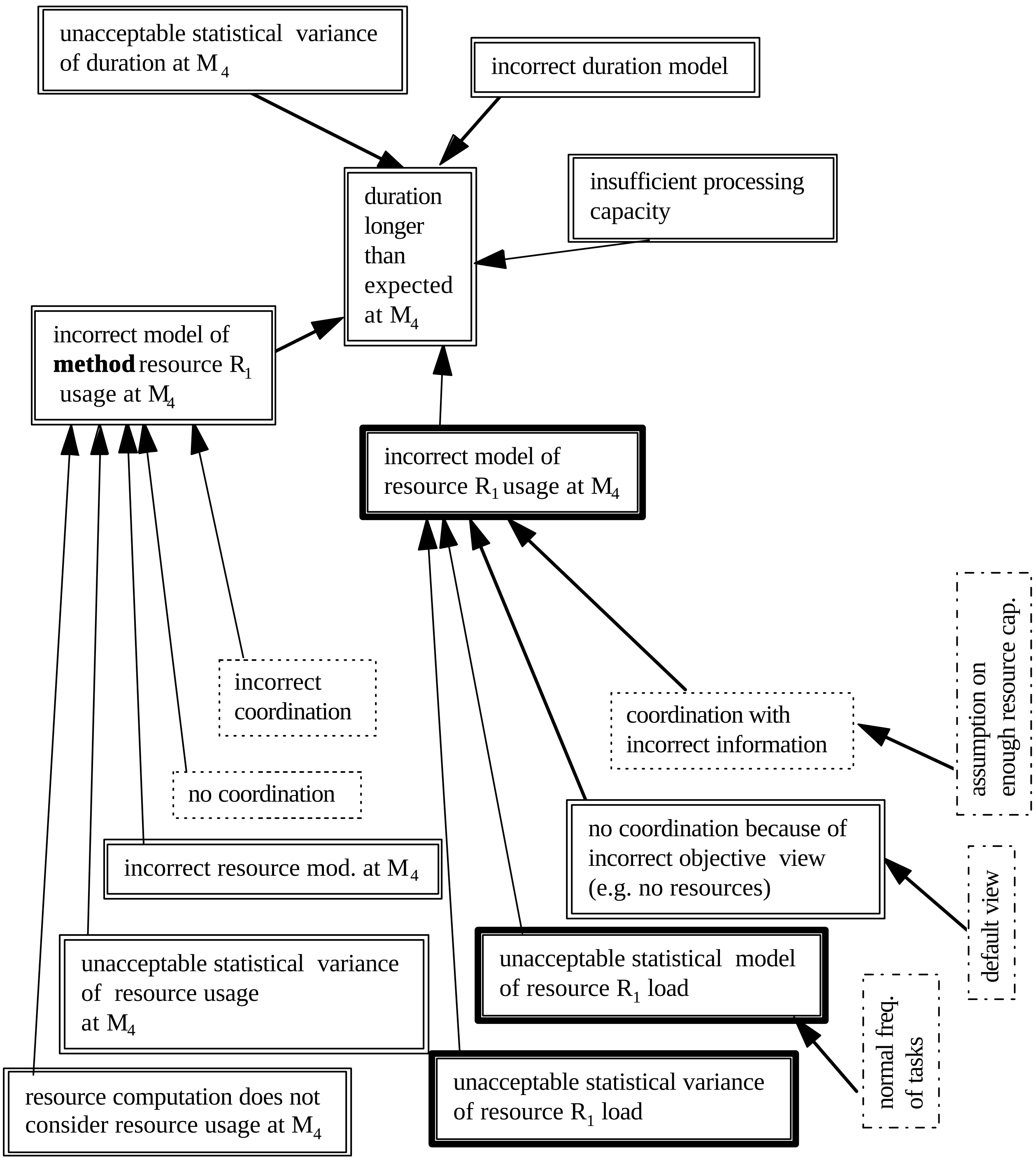

unacceptable statistical variance of duration at $M_4$
incorrect duration model
insufficient processing capacity
duration longer than expected at $M_4$
incorrect model of **method** resource $R_1$ usage at $M_4$
incorrect model of resource $R_1$ usage at $M_4$
incorrect coordination
assumption on enough resource cap.
coordination with incorrect information
no coordination
incorrect resource mod. at $M_4$
no coordination because of incorrect objective view (e.g. no resources)
default view
unacceptable statistical variance of resource usage at $M_4$
unacceptable statistical model of resource $R_1$ load
normal freq. of tasks
resource computation does not consider resource usage at $M_4$
unacceptable statistical variance of resource $R_1$ load

Figure 20: Agent Causal Model

Notice that some objects are not instantiated because they cannot be found in the trace. This is the case of facilitation to method $M_4$. By looking at the object _taems3 (the conditioned task structure assigned to agent B) appearing in the trace, no facilitate NLE is found (the task structure is depicted in Figure 17). Notice also that some objects are not explicitly shown in the trace because objects in them embody others, like for instance the assumption object, which contains information about coordination, views, model of resource usage, and so on. The agent causal model for that specific symptom, containing only ancestors of symptom, is shown in Figure 20. Having this model, the next step is to perform the analysis prescribed for each node, in order to test whether or not they can explain the symptom.

In the example we are following, let us assume that all statistical models, as well as the duration model are correct, so that there remain three nodes to test: “insufficient processing capacity (_agentB)”, “incorrect model of resource _$r_1$ usage by _$m_4$”, and “incorrect model of resource _$r_1$ usage at _$m_4$”). The fact that some tasks (e.g. $T_1$ and $M_3$) started and finished on time in agent B reduces the likelihood that there was insufficient processing capacity for agent B to perform its tasks.

However, the latter two explanations need further analysis. Again, let us assume that the statistical models (here concerning resources) are correct. This leaves two possible explanations for “incorrect model of resource _$r_1$ usage by _$m_4$”, namely “coordination with incorrect information at _agentB”, and “no coordination/incorrect objective view at _agentB”.

The same diagnosis would be reached by following the other symptom that also appears in the trace, namely “resource overloaded _$r_1$” (see Figure 14), since the overloaded is an unanticipated one.

### 7.3.2 Diagnosing Processor Failure

The trace depicted in Figure 21 also shows a symptom related to the duration of a method, namely “duration longer than expected at method _$m_1$”. The first step is to construct an agent causal model for this situation, using the general one and that symptom. This is similar to the one from the section above, except that, since method $M_1$ does not access any resources, those nodes related to resources are not instantiated. Other objects are instantiated in a way similar to that depicted in Table 1.

Once the agent causal model is constructed, the nodes in it must be analyzed to find out which explanations can account for the symptom. In this case the only possible explanation is “insufficient processing capacity”, since resource and facilitation nodes are not instantiated and, as assumed, all models (statistical, duration) are correct.

The diagnosis process for problems number three and seven in the list given in Section 7.3 are very similar to the one discussed here.

```
time   type description
-----------------------------------------------------------------
050    MSG  msg. to task assessor  to solve a problem
150    NTS  taems1
250    ASS  defaultAssumption
260    SCO  static non-local commitment
270    CTS  [AgentA,taems2]
285    DCOd) [AgentA,M1,50]
290    SCH  [M1,1,50]
301    EXE  M1 Started Execution at AgentA
360    SYM  DurationLongerThanExpected (M1)
365    MSG  decommitment msg. from AgentA to AgentB
...
```

Figure 21: Trace Containing the Symptom Duration Longer Than Expected due to Processor Failure (from Agent A execution)

### 7.3.3 Diagnosing Task Preemption

The symptom appearing in the trace shown in Figure 22 also relates to the execution time of tasks and methods. However, the symptom this time are "start of method _t$_1$ failed or is delayed" and "start of method _m$_2$ failed or is delayed". For the first symptom, in the general causal model, the three ancestors of this symptom node are:

- "previous task completion delayed", instantiated to object _m$_1$
- " enabling task delayed", not instantiated since M$_2$ has no direct enabling
- "preemption of task by scheduler", instantiated to the object _t$_1$

Hence the agent causal model has only two nodes, and their analysis is as follows. Agent B cannot know if M$_1$ is delayed, so it tells agent A to check this. The fact that task T$_1$ does not even starts can be explained by the fact that the results from the enabling e$_1$ had not arrived. Thus, agent B does not analyses further. Agent A, in its turn, constructs a causal model very similar to that of agent B, where the symptom appearing is "start of method _m$_1$ failed or is delayed". Agent A then analyses its trace and finds out that method M$_9$ (part of a task not shown in the figure) started in place of M$_1$, leading to the conclusion that T$_1$ got preempted.

```
time   type description
-----------------------------------------------------------------
050    MSG  msg. to task assessor  to solve a problem
150    NTS  taems1,B
250    ASS  defaultAssumption
260    SCO  static non-local commitment
270    CTS  [AgentB,taems3]
280    DCO [AgentB,M2,100]
285    DCO [AgentB,T1,200]
290    SCH  [[M2,51,100],[M3,101,150],[M4,151,200]]
360    SYM  StartFailedOrDelayed (T1)
360    SYM  StartFailedOrDelayed (M2)
```

Figure 22: Trace Containing Symptom Start Failed or Delayed because Method M$_1$ is preempted. a) M$_9$ starts in the place of M$_1$.

### 7.3.4 Diagnosing Multiple Faults

In some cases, a symptom is found, which cannot be explained by a single node of the causal model. Performing the analysis related to all nodes in the model is not enough to determine the explanation for the symptom. As an example, consider the case in which the methods M$_2$ *and* M$_3$ (Figure 17) are each delayed, but within a certain threshold, thus within the statistical variance. The analysis associated with the node "unacceptable statistical variance of duration" in the causal model (Figure 13) finds nothing wrong since the variance is acceptable. However, the combined effect of those delays explains the observed symptom of method M$_4$ being delayed beyond an acceptable threshold.

In order to deal with such combination of events causing a phenomenon, we perform a quantitative analysis and keep a log of those explanations which could be combined with others (i.e. those which are close to the threshold).

Other examples of multiple fault scenarios are:

- More than one symptom appears, each explained by a single candidate, say $E_1$ explains $S_1$, and $E_2$ explains $S_2$. However, $E_1$ and $E_2$ may not be verified individually (e.g. because certain thresholds were not reached), but together they may explain a third symptom ($E_1 \wedge E_2 \rightarrow S_3$).
- Candidate (to explanations) $E_1$ and $E_2$ explain symptoms $S_1$ and $S_2$ ($E_1$ , $E_2 \rightarrow S_1$ , $S_2$). However, it is not known whether $S_1 \rightarrow E_1$ or not, and whether $S_2 \rightarrow E_2$ or not.
- One candidate $E_1$ may explain two or more symptoms.
- More than one candidate of explanation exist for a given symptom.

## 8 Conclusion

Our main concern has been the tradeoff between the effort to select an effective coordination strategy and its results. This reinforces the need of situation-specific control rules, in which the benefits and the costs of coordination for the current situation are taken into account. Past researches (e.g. Sugawara and Lesser 1993, 1998) addressed a particular coordination scenario (diagnosis of local area network) by a learning method which identify which information improved coordination in specific problem-solving situations, not exploring further range of problems though. Motivated by this limitation, we have been focusing on questions like "how much domain dependent knowledge is needed?", "which level of detail this knowledge must be reflected in the traces of the agents' activities?", "how can the reasoning and the results of the diagnosis be expressed in a domain-independent way?". Thus, we decided to focus on a situation-specific control, which is based more on a long-term understanding of the frequency of tasks occurring, the character of these tasks in terms of their resource and coordination needs and the available resources.

Our first result is the conclusion that, by using domain-independent frameworks for representing agent's goals and methods (TÆMS), and coordination mechanisms (GPGP), we are able to abstract the aspects of the domain that affect coordination (e.g. goals, criteria for successful performance, performance characteristics and resource requirements associated with the alternative methods for accomplishing of the goals, and both the qualitative and quantitative interdependencies among these methods and those of other agents). Within an extended, situation-specific GPGP (Lesser et al., 1998b), an organizational designer can focus coordination activities to reason only about a subset of possible coordination relationships (those specified) existing among agents.

Second, we are tackling here detection and diagnosis so to use the same model employed to represent the organizational knowledge and the coordination. This knowledge plus the traces of the agents' activities are used to detect changes from an expected behavior of the problem-solving system, locate a symptom, and find an explanation for it, given a mapping of symptom-explanations and ways to verify them, without the need of domain specific knowledge.

Finally, we successfully modeled the diagnosis problems posed by Sugawara and Lesser (1993) in a domain-independent way, by representing them as TÆMS task structures, adding both a model of resources usage by the methods, and assumptions on resource availability, and by applying our diagnosis reasoning.

Ultimately, we want to focus on systems which can survive in changing environments. Thus, diagnosing inappropriate behavior of the problem-solving system is a crucial step to correct the system's domain knowledge, the model of the tasks arising in the environment, and consequently to allow good coordination which, in its turn, is the key to survivable systems.

In short, the most important aspects of our research are the exploitation of *model-based, domain-independent* distributed diagnosis of inappropriate behavior, the further adaptation of agent coordination based on that diagnostic, and the use of a single model for this whole process.

As for future developments, we see the need to extend GPGP both to increase the number of coordination mechanisms, and to allow more organizational control. We are also concerned with practical issues like "how much situation-specificity can be achieved when the framework aims to be very broad in problem representation?", "are there potential issues of scale?", "for which situations is it worthwhile to make an off-line design based on a description of the environment (likely tasks), versus an on-line learning of the organization as a result of a series of adaptations based on experience?".

## References

K. S. Decker and V. R. Lesser 1993. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington. AAAI Press.

K. S. Decker 1995. Environmental Centered Analysis and Design of Coordination Mechanisms. *Ph.D. Thesis*, University of Massachusetts.

K. S. Decker and V. R. Lesser 1995. Designing a Family of Coordination Algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, San Francisco, CA. AAAI Press.

K. S. Decker 1996. TÆMS: A framework for analysis and design of coordination mechanisms. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 16. Wiley Inter-Science.

K. S. Decker, K. Sycara, and D. Zeng 1997. Designing a multi-agent portfolio management system. In *Proceedings of the AAAI Workshop on Internet Information Systems.*

K. S. Decker and J. Li, 1998. Coordinated Hospital Patient Scheduling. In: *Proc. of the Third International Conference on Multi-Agent Systems (ICMAS 98*), pages 104-111, IEEE Press.

E. Hudlicka and V.R. Lesser 1987. Modeling and Diagnosing Problem-Solving System Behavior. *IEEE Trans. on System, Man, and Cybernetics*, **17**(3), pp. 407-419.

V. R. Lesser, M. Atighetchi, B. Benyo, B. Horling, A. Raja, R. Vincent, and T. Wagner, P. Xuan, and S. XQ Zhang 1998a. A Multi-Agent System for Intelligent Environment Control. *Tech. Rep. TR-98-40,* Univ. of Massachusetts. Also submitted to *Agents'99*.

V. R. Lesser, K. Decker, N. Carver, A. Garvey, D. Neiman, M. Nagendra-Prasad, and T. Wagner 1998b. Evolution of the GPGP Domain-Independent Coordination Framework. *Tech. Rep. TR-98-05,* Univ. of Massachusetts.

F. von Martial 1992. *Coordinating Plans of Autonomous Agents.* LNAI 610, Berlin, Springer Verlag.

M. V. Nagendra Prasad, V. R. Lesser, and S. E. Lander 1996. Learning organizational roles in a heterogeneous multi-agent system. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*.

T. Sugawara and V.R. Lesser 1993. On-Line Learning of Coordination Plans. *COINS Tech. Rep. 93-27*, Computer Science Dep., University of Massachusetts.

T. Sugawara and V.R. Lesser 1998. Learning to Improve Coordinated Actions in Cooperative Distributed Problem-Solving Environments. *Machine Learning,* Kluwer Academic Publishers (to appear, 1998).

T. Wagner, A. Garvey, and V. R. Lesser 1997. Complex Goal Criteria and Its Application in Design-to-Criteria Scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*.

## Apendix I: The diagnostic actions associated with the causal model

The causal model described in section 6.3 (Figures 13, 14 and 15) gives the conceptual graph representation for identifying the possible causes. Each node in the causal model corresponds to either a symptom/fact to be verified or a hypothesis for the cause of the symptom/fact.

The agent performs diagnostic activities by constructing and expanding a graph that represents the causal relations of the activities/events happened during the course of the problem solving activities. When constructing and expanding the graph, the agent will need to utilize the causal mode by verifying the symptoms and checking the hypotheses associated with the actual problem solving. These actions are domain independent diagnostic actions implemented by the diagnostic module. The outcomes of the actions decide how the graph should be expanded; in other words, these actions specify the rules to be taken by the diagnostic module. In the following text we specify the actions associated with each node in the causal model.

1 completion-time-delayed(task) --- This is a simple check by comparing the expected completion time of the task as indicated in the schedule and the actual completion time as seen by the execution module.
2 start-delayed(task) --- This is also a simple check by comparing the expected start time (indicated in the schedule) and the actual start time as seen by the execution module.
3 previous-task-completion-delayed(task) --- This is to check if the task preceding this one (task) in the schedule has a delayed completion. This leads to completion-time-delayed(preceding task).
4 enabling-task-delayed(task) --- Also a simple check to verift: (1) if task is enabled by some other task (there is an enables relations imposed on task, and if so, (2) does that enabling task has completion-time-delayed(enabling task)?
5 preemption-of-task (task) --- This is to see if the task has been interrupted (suspended) during its execution. A simple check of the execution module's log will reveal if this is true.
6 duration-longer-than-expected(task) --- This is to compare the actual task(method) duration (the difference between completion time and start time) and the duration specified in the TÆMS task structure. Normally, TÆMS specifies a duration distribution, therefore the comparison is based on probabilistic measures.
7 insufficient-processing-capacity (agent) --- This relates to the organizational role of the agent. If tasks often take longer time to complete because the agent has more tasks (load) than the amount the designer has expected. To detect this the agent needs to be monitored for a long time so that the average load and the frequency of deadline misses can be calculated.
8 unacceptable-statistical-variance-of-duration (task) --- The duration in TÆMS allow some variances, thus a single case of duration-longer-than-expected may simply be because of an unexpected variance. In this case nothing is wrong, just that the statistics is against the expectation. However, high frequency of such variance may indicate an incorrect duration distribution. Therefore, if necessary, the agent should start monitoring the task duration distribution.
9 incorrect-duration-model (task) --- This is directly related to the previous hypothesis. Here as a result of the monitoring process (note the monitoring process is also the learning process), the agent now has a more accurate organizational view of the task duration distribution.
10 incorrect-model-of-method-resource-usage (uses) --- A task may take longer duration because the task needs more resource than previously thought. Note the resource usage is typically specified as a *uses* NLE for the method requesting that resource, and usually has a distribution associated with it. This model of distributiom is specified by the organizational designer, but may be incorrect. To verify this hypothesis

we need to turn on the resource monitor (at the resource side) and start learning the actual resource usage distribution of that method.

11 incorrect-resource-model (agent) --- This refer to the case that the resource characteristics is different than the agent previoulsy thought, for example, the network is of lower bandwidth or more error-prone, etc. To detect this we need to compare the agent's view (assumption) of the resource against the objective resource model. However, in some cases an objective model cannot be obtained. In this case, statistical monitoring and reasoning are needed in order to deduct the objective model.

12 unacceptable-statistical-variance-of-task-resource-usage (uses) --- The amount of resource used by the task is still within the variance of the *uses* relation, but of lower confidence level. Again this could be just a case of statistical variance. If necessary, turn the resource monitor on.

13 resource-computation-does-not-consider-resource-usage (task) --- the duration distribution of a task does not take into account the resource usage (this depends on the resource characterisitics), i.e., initial duration computation assumes normal network traffic, thus does no include network delay. There is no way to check this directly, but the idea is to look at the Limits NLE and to see if the Limits NLE takes effect in an unexpected way, e.g., the Limits is ignored (assumed to be not effective).

14 no-coordination (agent) --- This refers to the case that the organizational design believes (explicitly) that the agent needs not coordinate with other agents over this resource. The designer made this decision based on some assumption of the resource, but that assumption is not valid and some coordination is needed.

15 incorrect-coordination (agent) --- In this case, the diagnosis module finds out that the agent did attempted to coordinate over this resource, but the coordiation is incomplete because of the the agent does not have or explore important information about correlated usage, such as overlapping, redundancy. Some relationship is missing in the agent's view on which the coordination activities are based.

16 incorrect-model-of-resource-usage (agent) --- The model of overall resource usage of a resource is not correct, because of agent lied or unawareness of their actual usage (e.g., a software bug). The agent maintains a local model of the resource, and has some assumption about the resource usage patterns, but the model is wrong because of wrong information.

17 unacceptable-resource-load-model (resource) --- To verify this, resource monitoring is needed in order to obtain the actual resource load statistics, and then compare it with the assumed load model.

18 no-coordination-because-of-incorrect-objective-view(agent) --- The agent does not have the correct *uses* relationship(s) when coordinating with other agents. This can be detected by checking if the agent's objective view contains the *uses* NLE(s) or not.

19 coordination-with-incorrect-information (agent) --- The designer possesses incorrect information when deciding the type of coordination. To check this, the diagnosis module needs to verify the designer's assumptions about the objective view.

20 enough-resource-capacity (resource) --- The assumption is that resources are enough, but there is an overload of resource.

21 default-view (agent) --- the organization assumes a default view for the use of the resource, because the overhead of finding out the actual view. In order to be able to verify this, it is assumed that the agent explicitly specifies that a default objective view is used.

22 normal-task-frequency (agent) --- the agent assumes that the tasks arrive at their normal frequency (as specified by the designer) in order to decide the resource load. This information is assumed to be explicitly represented in the agent reasoning process. The verification process would then become a simple comparison between the actual frequency and the normal frequency.

23 incorrect-model-of-facilitation-power (facilitates) --- This is to verify that the facilitation power is statistically lower than what is specified in the TÆMS task strucuture by the designer. The action would be to start monitoring the facilitates relation and to compare the actual facilitation power with the specification.

24 incorrect-model-of-facilitation-quality (facilitates) --- This refers to the situation that the quality threshold associated with the facilitates NLE is wrong, therefore results in the failure of the expected facilitation.

The action would be to compare the quality outcome of the facilitating task and the quality threshold of the facilitates NLE.

25 unacceptable-statistical-variance-of-facilitation-quality (facilitates) --- This refers to the case that the facilitation power allow some variances based on the quality of the facilitating task. However, the actual facilitation power is of lower confidence level. Note in this case it may be the case that no fault is present, but the task is unlucky.

26 resource-overloaded (resource) --- A resource overload can be reported either by the resource manager (or the agent managing this resource), or by the agent that receives a Limits NLE that indicating a resource overload situation.

27 anticipated-overload (agent) --- Here the organizational designer has anticipated the resource overload situation. For example, the designer estimates that the network overload happens about once every two hours, or something like 1% chance of overload at any given time. There can also be a detailed model about the degree of the overload, or even based on the task invovled.  These overload are treated as normal. Only when the overload is happening more frequently and/or more intensely than specified (by way of resource monitoring), this estimation is challenged.

28 unanticipated-overload (agent) --- This is the opposite of the  previous hypothesis. Note that more detailed resource model the agent has, the easier to identify whether an overload is unanticipated.

29 quality-lower-than-expected (task) --- This is to check whether the quality of the task is lower than expected in the commitment.  Note that a commitment specifies an expected quality (somewhat similar to the expected stated in the schedule about the expected duration of a task.)

30 facilitating-task-quality-too-low (task) --- This is to check that  if the facilitating task's quality is lower than the threshold value of the facilitates NLE. If so, the diagnosis module would start diagnosis the symptom quality-lower-than-expected(facilitating task).

31 incorrect-quality-model (task) --- similar to incorrect-duation-model(task), but looking at quality instead of duration.

32 low-expected-quality (task) --- This refers to the case that the actual quality is lower than the agent announced (i.e.. agent lying about the quality of task). It is unclear what information is needed to verify this hypothesis right now.

33 task-finished-with-no-quality (task) --- This is easy to check. No quality means quality is zero (or negative). Or, since quality is also time-dependent, missing the deadline can also mean the task failed.

34 unacceptable-statistical-variance-of-quality (task) --- This is another case of having sheer bad luck. In TÆMS quality specification  can allow a variance thus we can decide the confidence level of the quality outcome.

35 no-enabling-task (enables) --- This refers to the case that an expected enables NLE does not arrive in time. To verify this, first check if there are such NLE(s) expected, and whether and when they arrived. Based on the diagnosis result, it could be that the Enables has a wrong model, or the enabling task has lower quality than expected, or completion of the enabling task is delayed.

## 5 Apendix II: Classes Defined in the Implementation

It follows a definition of the classes implemented in Java, organized in ? main groups , namely TÆMS objects, agent related definitions, the problem (example) being solved, the trace, the causal model, and the diagnosis.

```
Agent
class Agent {
  public static int count = 0;
  public static Agent list[] = new Agent[32];
  public int id;
public Trace trace;
  public TaskStructure ts;
public java.util.Vector resources;
```

Trace

```
class Trace extends java.util.Vector {
  private int agentId;
class TraceEvent {
  public static int count = 1000;
 public int evtime;            // event time
  public int eventType;        // event type
  private int id;               // id, different from index
  public int trig;             // what triggered this event
  public Object content;       // content ...
class Assumption implements TraceTypes {
  public int type;  }
class Commitment  {
  public int id;
  public Object label;
  public int type; //0=do  1=deadline  2=earliest_start_time
  public Object agent;
  public int task;
  public double negotiability;
  public double utility;
  public double importance;
  public double minimumQuality;
  public int earliestStartTime;
  public int deadline;
  public int satisfied;  //0=t  1=f  2=unknown
class CoordMode implements TraceTypes {
  public int type;
class Symptom implements TraceTypes {
  int time;
  public int type;
  public java.util.Vector verifiedExpAct;
class ExecTimeSymptom extends Symptom {
  public static java.util.Vector listExpl = new java.util.Vector();
  public Method method;
class ResourceSymptom extends Symptom {
  public static java.util.Vector listExpl = new java.util.Vector();
  public Resource resource;
  public int resourceId;
class Execution  implements TraceTypes {
  public    int type;
  public   int methodId;
  public   int agentId;
  public   int startTime;
  public   int quality;
  public   int status;
class MSG  implements TraceTypes {
  public static MSG list[] = new MSG [32];
  public static int count = 0;
  private int id;
  public int _from;
  public int _to;
  public int _type; // the content of the msg
class NonLocalCommitment  {
  public int id;
  public Object label;
  public int task;
  public int fromAgent;
  public int toAgent;
  public double qualityDistr[];
  public double timeDistr[];
class Plan implements TraceTypes {
class Schedule extends java.util.Vector implements TraceTypes {
class ResLog extends java.util.Vector {
  int resId;
class ResourceCheckable extends Checkable {
  public Resource rsc;
class ResourceEvent {
  int evtime;  // the time stamp
  int task;  // calling task id (node id)
  int act;

  TAEMS
class Node {
  protected int nodeType;
class Task extends Node implements Defines {
  public static int count = 0;
  public static Task list[] = new Task [64];
class TaskGroup extends Node implements Defines {
  public static int count = 0;
  public static TaskGroup list[] = new TaskGroup [32];
class TaskStructure extends Node implements Defines {
  public static int count = 0;
  public static TaskStructure list[] = new TaskStructure [10];
  public int deadline;
  public java.util.Vector relations;
class Method extends Node implements Defines {
  public static int count = 0;
  public static Method list[] = new Method [64];
  public int duration;
  public int startTime;
  public int finishTime;
  public int accruedTime;
  public int status;
  public java.util.Vector resource;
   interface Relationship {
class Enables implements Relationship  {
  public int id;
  public String label;
  public int agentId;
  public int from;  // from-task
  public int to;    // to-task
  public int delay;
class Facilitates implements Relationship {
  public int id;
  public String label;
  public int agentId;
  public int from;
  public int to;
  public double qualityPowerDistr[];
  public double durationPowerDistr[];
  public double costPowerDistr[];
  public int delay;
class Resource extends Node implements Defines {
  public static int count = 0;
  public static Resource list[] = new Resource [64];
  public int type;
  public double capacity;
  public ResLog log;
```

```
class ResourceLimits implements Relationship {
  public int methodId;
  public int resourceId;
  public double saturation;
  public double percent;
  public int est;       // earliest-start-time
  public int duration;
  public double quality;
class ResourceUses implements Relationship {
  public int methodId;
  public int resourceId;
  public double amount;
  public int est;       // earliest-start-time
  public int duration;
  public double quality;
```

Causal Model

```
class CausalModel {
private Vertex[]  v;
  private Edge[]  e;
class Edge {
  public Vertex from;
  public Vertex to;
  public String name;
class NLEVertex extends Vertex
class ResourceVertex extends Vertex {
class TaskVertex extends Vertex {
class Vertex {
  public String name;
  public String comment;
  public int nAncestor;
  public Vertex[] ancestor;
```

Diagnosis

```
class DiagModule implements TraceTypes {
  public Agent agent;
  public CausalModel model;  // -- the model to be used
  public Problem problem;
  public Trace trace;
  public int time;         // -- current time;
  public DiagWeb result;   //  diag results
class DiagWeb {
  private java.util.Vector checkables;
  private java.util.Vector edges;
  class CheckEdge {
   public Checkable from;
   public Checkable to;
  class Checkable implements TraceTypes {
   // internal data strucuture
   public int time;             // the time (range) involved
   public Vertex v;             // what symptom to check
    public int    value;          // the value of it
  class NLECheckable extends Checkable {
   public Relationship nle;
  class TaskCheckable extends Checkable {
    public Node task; // Task, Method, TG, TS
```

Problem being Solved

```
public  abstract class  Problem  implements TraceTypes, Defines
  public int  numAgent;
  public int  numRes;
  public Trace traces[]; // for use in new TraceAnalyzer
  public ResLog reslog[];
public class Warren1 implements TraceTypes  {
class WarrenProblem extends Problem {
  // extra things from Warren1Trace
  public int  linedelay[]; // line cost
  public TaskStructure ts[];
  public java.util.Random rand ;
```