On-Line Learning of Coordination Plans

Toshiharu Sugawara and **Victor Lesser**

COINS Technical Report 93−27

June 1993

On-Line Learning of Coordination Plans*

Toshiharu Sugawara
NTT Basic Research Laboratories
3 – 9 – 11 Midori-cho, Musashino
Tokyo 180, Japan
sugawara@ntt-20.ntt.jp

Victor Lesser
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
lesser@cs.umass.edu

Abstract

Coordination is an essential technique in cooperative distributed problem solving (CDPS). However, sophisticated coordination strategies are not always cost-effective in all problem-solving situations. This paper presents a learning method to acquire coordination plans for specific problem-solving situations so that the appropriate type of coordination strategy is used. This learning is accomplished by recording and analyzing traces of inferences after problem solving. The analysis results in identification of situations where inappropriate coordination strategies have caused redundant activities or the lack of timely execution of important activities, thus degrading system performance. The analysis is also used to create situation-specific coordination plans, which use additional non-local information about activities in the networks, that are added to the system to remedy the problem. These situation-specific plans have the effect of introducing different levels of coordination into this system. We present this approach for coordination using two examples from a real distributed problem-solving application involving diagnosis of a local area network.

*This research was conducted at the University of Massachusetts at Amherst. Toshiharu Sugawara's stay at the University of Massachusetts was supported by Nippon Telegraph and Telephone Corporation (NTT). This research was also supported in part by a grant from Network General Corporation, ARPA contract N00014-92-J-1698 and ONR contract N00014-92-J-1450. The content of the information does not necessarily reflect the position or the policy of the organizations supporting this research, and no official endorsement should be inferred. This is an extended version of a paper with the same title that will appear in the proceedings of 12th workshop on Distributed Artificial Intelligence.

Contents

1 Introduction

Many real world application domains such as network control and diagnosis, cooperating robots, and office automation require for their solution cooperative distributed problem solving (CDPS) [15]. In these domains, agents in the network cannot solve all their own local problems without cooperating with other agents. Additionally, we assume that agents may be solving a number of different problems concurrently. Some of these problems are directly interrelated because they are subproblems of a larger problem; some of these are indirectly interrelated through the resources required in achieving their solution, and others are independent. The interrelationships among different agent activities thus may be dynamic based on specific problems currently being solved in the network and the specifics of the environment. For some interrelated problems, coordination of activities among agents is key to the timely and efficient generation of their solutions. For example, lack of effective coordination can result in:

(a) the inability to solve a problem because one agent prematurely committed a resource or executed an inappropriate action making a solution impossible.

(b) the unnecessary expenditure of potentially overloaded or costly processing and communication resources to derive results that are either redundant, no longer needed, or of marginal utility.

(c) the lack of timely execution of important activities. Other agents who need the results of these activities for their problem solving are sufficiently delayed, so that the utility of their problem solving is significantly degraded.

However, sophisticated coordination regimes that require extensive use of processing and communication resources may not always be worthwhile. In general, coordination requires the acquisition of non-local information about the state of activities of other agents and requires computations to reason about this information in order to make more informed local control decisions [16]. For example, coordination to avoid redundant activities may be unnecessary if processing resources are not overloaded and if communication is neither expensive nor overloaded. In this case, local problem solving is more efficient since there is no additional overhead for coordination, and redundant computation and communication in the network does not impact the cost/speed of local problem solving. This situation can occur in, for example, internetwork [2] diagnosis. Multiple agents, as a part of their diagnostic process, may send a number of test packets to get the same information. This does not cause any serious side-effects if the agents are connected via high-speed lines. However, if there are slow lines or if a tariff is associated with each message, these test packets may cause a problem. In this case, the use of processing and communication resources to coordinate agents is worthwhile if they can significantly reduce the number of test packets.

We feel that in complicated and dynamic distributed applications, it is extremely difficult to anticipate all the problem-solving situations and associated environments in which coordination

is worthwhile. Instead, we propose that the CDPS system learns on-line from its experiences the appropriate coordination strategy for the specific problem-solving situation. Our intuitions are that both explanation-based learning (EBL) [5, 20] techniques and statistical learning techniques [19, 21] can be useful. For this specific work described here, EBL techniques will be used. We feel that extensive information about the effectiveness of specific coordination rules can be derived by analyzing a trace of problem solving; through this analysis, the system can tailor its coordination strategy to eliminate redundant operations, communications, and job requests, and appropriately prioritize specific processing and communication actions. We also feel that agents can often predict the appropriate compromise results of negotiations based on past negotiations in the same, or in a similar situation; thus, they can bypass the need for costly negotiations. Various kinds of situation-specific control rules for CDPS can be derived by the proposed learning method. For example, coordination rules about when, to whom, and which information should be sent (communication policy and organizational knowledge), decision rules on the priorities of messages and actions, and explanations to quickly and accurately understand the meanings of non-local information can be derived.

The important questions that need to be answered in order to implement learning of coordination rules are:

(1) how to identify situations in which inappropriate coordination has occurred,

(2) how to diagnose the reasons for inappropriate coordination, and

(3) how to the modify control strategy so that appropriate coordination will occur the next time this specific situation arises.

Our approach uses a cooperative causal analysis based on local and non-local domain models and traces of distributed problem-solving executions. By domain models, we mean knowledge about how problems observed at one agent will be manifested at other agents, about network topologies, about the detailed working of an agent's local problem solving, etc. This causal analysis is triggered by a number of conditions such as problem execution failing outright, taking too long, or using too many resources. These trigger conditions are, in our current implementation, defined by the system designers. However, we hope that in the future these conditions (especially the exact parameter values) can also be acquired through experience. This analysis is also invoked randomly in order to look for hidden problems that are not observed under current conditions. Comparative analysis [12] which compares the current trace with the past traces of similar situations is also used when the system does not have a sufficiently rich domain model to fully explain the failure.

Our approach to learning coordination rules can be couched in the following more general perspective that comes out of work by Decker and Lesser on generic coordination strategies [3]. Each agent makes scheduling decisions based on a subjective view of its own and other agents' activities. In certain situations this subjective view will lead to ineffective/inappropriate problem solving because certain local and non-local task interrelationships have not been appropriately

taken into account. We assume that there is some component that is monitoring the problem solving in the system that recognizes this undesirable situation. We then try to recognize what aspects of the actual objective task structure were not taken into account in the scheduling of local activities that are responsible for the perceived problem. We then modify the local control to add additional local and non-local information gathering and analysis actions to recognize when the specific objective structure that caused the problem is occurring. In the case where this objective structure is recognized, a new scheduling strategy is used to avoid the problem by taking into account a more accurate view of the actual coordination relationships.

In the following section we briefly describe the continuous, CDPS application system, LODES [25, 26], which performs internetwork diagnosis. All examples and research issues in this paper are obtained from problems observed during the operations of this system. Section 3 describes an example problem which is caused by the uncoordinated actions of agents. In Section 4, based on the example developed in the previous section, our CDPS learning architecture will be elaborated. As a part of this discussion, we will also detail how situation-specific coordination rules can be added to a suitably modified version of the LODES architecture. Section 5 and Section 6 present, in detail, how this learning framework is used on two example problems. Section 7 describes the related research and the future directions of our research.

2 LODES System

2.1 LODES Overview

The LODES is a continuous, cooperative distributed knowledge-based system capable of monitoring TCP/IP-based internetwork traffic over Ethernet™and diagnosing problems that occur during transmission of information (mainly layer-2-to-layer-4-related problems). Each network segment has a LODES agent; these agents diagnose problems either cooperatively or autonomously depending on the problem type. A LODES agent consists of four components (see Fig. 1): the NOBS (Network Observer) component analyzes all packets flowing through its network segment and calculates statistical data. Based on this analysis and knowledge about symptoms of internetwork problems, the NOBS component can recognize the potential existence of a network problem. Additionally, NOBS can save packet data into a file in a timely manner to be used for later analysis. The NePS (Network Packet Sender) component can build and send test packets to any host so that the LODES agent can understand the features and the current state of this host. The Chales (Communication Handler) component is responsible for communicating with other agents. The LAND (Local Network Diagnostic) component is the main part of this system that controls other components during diagnosis. During periods when the network is functioning correctly, the NOBS and LAND components build models of the evolving features of local network segments such as how many host nodes are attached, which nodes are routers, what kind of services are supplied by each node, when a node is attached or replaced, what kind of services are typically used, how each node reacts to test packets and some

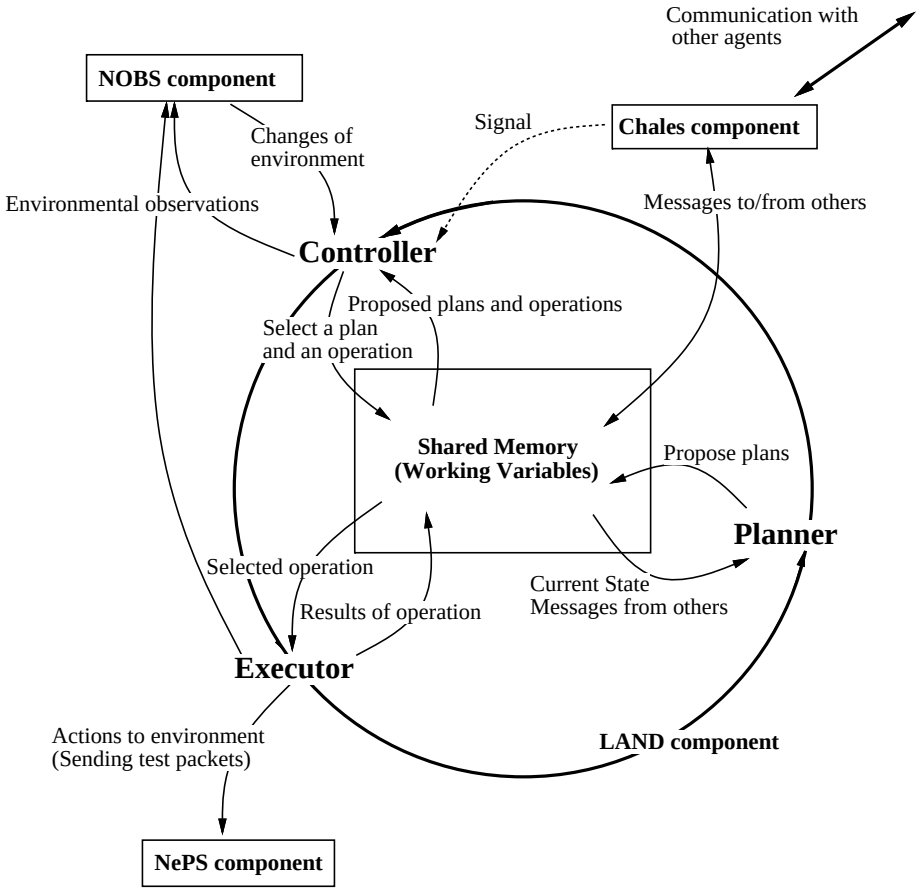


Figure 1: Conceptual Structure of LODES

undesired or unexpected packets (such as unknown protocol packets, a packet whose destination is another node), routing definitions in adjacent routers, and throughput of lines between local and adjacent network segments. Note that each agent has the same long-term diagnostic knowledge. However, features of network segments can be quite different, so that the diagnostic strategies and plans used by an agent for the same problem may be different.

The LAND has three modules: Planner, Controller, and Executor. The Planner, based on local and acquired data, proposes the possible causes of the current problems (such a cause is called a *hypothesis* (HP)), together with a likelihood rating. Based on these ratings, the Controller selects the most plausible HP and places this HP in a shared memory area (SMA). The Planner builds a high-level plan to verify the correctness of this hypothesis as a valid explanation for the problem and translates each of the plan steps into an appropriate sequence of mid-level plans. These plans are further expanded into a sequence of operations to achieve the plan's objectives. These sequences of plans and operations are stored in the SMA and are retrieved by the Controller. The Controller also looks at received data and requested jobs (which in turn are translated into operations) from other agents in selecting an appropriate operation which,

from its local viewpoint, contributes most to the overall network diagnosis. Then the Executor executes this selected operation and its results are stored in the SMA. After the initiation of the operation, the Planner reanalyzes what the most appropriate HP and plan is to work on, based on the newly received data and the result of the operation.

2.2 Coordination Level

Coordination of LODES diagnosis agents is classified into three levels: *nearly autonomous coordination* (A-coordination), *shallow coordination* (S-coordination), and *deep coordination* (D-coordination, see Fig. 2). The nearly autonomous coordination level implies that each agent's local control regime is almost exclusively computed based on data available locally. In this mode, it is assumed that knowledge about the specifics of operations and their results at other agents will minimally affect the decision about what operations to choose next. Furthermore, analyzing data received from other agents will unnecessarily slow down local problem-solving activities. In contrast, the S-coordination level involves analyzing data received from other agents and requesting data and jobs from other agents. However, the non-local knowledge used in making control decisions is still severely limited since information about the range of activities of other agents and detailed features of the environments of other agents is not used. Control of local problem solving is based on only a small number of parameters or ratings attached to job requests from other agents [27]. Both of these coordination levels can result in some important jobs not being executed in a timely manner, some messages not being sent at the right time, and redundant job execution. However, it is assumed that this lack of coordination does not result in a serious side-effect. In situations requiring S-coordination level, payoff-matrix-based or utility-based coordination is sufficient; tighter coordination in these situations leads to unnecessary communications and control inferences reducing local problem-solving efficiency. When a job is requested — for example, the receiving agent uses without modification, as though derived by itself, an attached rating number decided by the requesting agent — the receiving agent does not analyze why it is important. The D-coordination level implies that local control decisions involve the use of a partial global view of the goals and activities of other agents [7]. This view is achieved by agents exchanging non-local data such as their current HP, scheduled plans, resource usages, intermediary results, and observed data of other agents. A partial global view enables each agent to understand the problem-solving state of other agents, so that control decisions about which local tasks to execute next are made to optimize network problem-solving performance rather than local problem solving. D-coordination is also necessary to avoid serious side-effect interactions among agents and environments.

The development of situation-specific control rules can be seen as an attempt to learn which coordination level is appropriate for a specific problem-solving situation (see Fig. 3). We assume that normally an agent reasons under S-coordination, that is, it decides the ratings of plans and operations based on local and acquired non-local pay-off matrices or utility numbers [27]. When the D-coordination level is selected according to predefined knowledge or learned coordination rules, an agent identifies the required non-local information, coordination actions, and

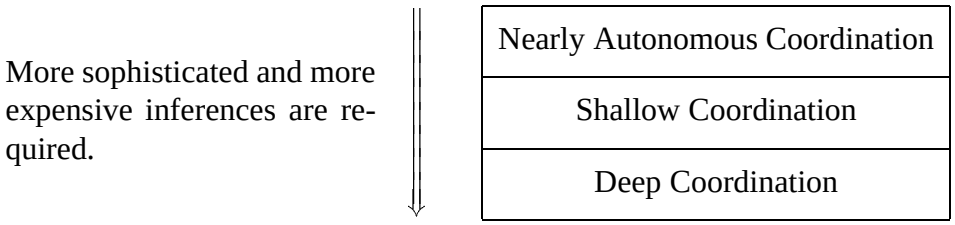


Figure 2: Coordination Levels

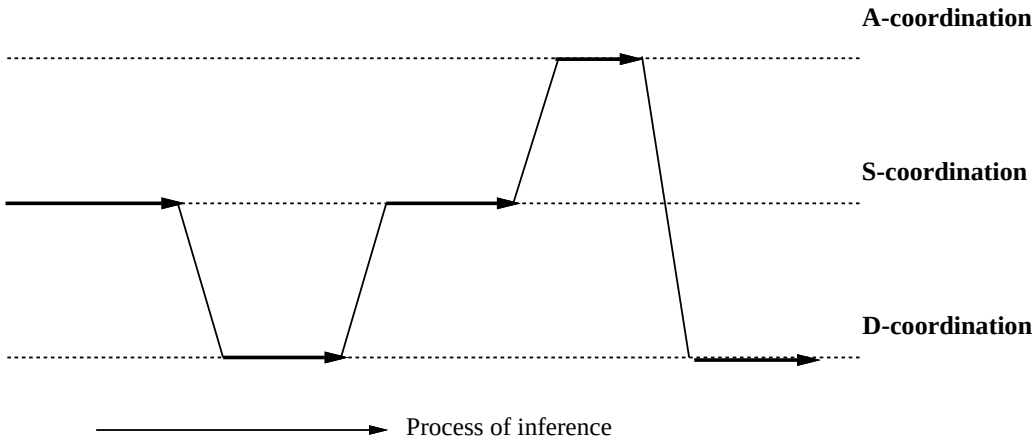


Figure 3: Changes of Coordination Levels

negotiations. Then it decides on the ratings of plans or schedules for the required actions for coordination. Under A-coordination, references to messages from other agents are postponed that are associated with plans operating under A-coordination, until messages associated with plans operating under S- or D-coordination are received, or until the agent finds that no further local problem solving can be performed without access to non-local data. When multiple problems that require different coordination levels occur simultaneously, agents can handle messages from/to other agents on the problem-specific and situation-specific basis. That is, depending on which problem the message is involved in, the message is analyzed under the appropriate coordination level.

2.3 Models of Network and Other Agents

Each LODES agent initially has only a local view of the computer network including local routing definitions, local topological structures, the number of host nodes attached, and maximal throughput to adjacent network segments. This model is derived as a result of analysis of packet

flow through the agent-specific network segment. To always maintain a complete and accurate model of the entire network can be quite expensive and time-consuming if the size of the network and the number of hosts are large, due to changing and evolving network features over time. Instead, the LODES agents exchange local information and use the domain theory during problem solving based on the needs of the specific diagnostic situation. This enables the agents to incur the cost of obtaining a more encompassing, accurate model of the network only when necessary. The universal and local network domain theory model used by a LODES agent is described in Appendix 2.

3 An Example Problem

Fig. 4 shows the network environment for the example problem where there are network diagnosis (LODES) agents L1, L2, ..., L7 monitoring traffic on different segments of the network. The example problem is as follows: HostA on Net1 sends a broadcast to Net7, but most of the hosts in Net7 cannot understand that protocol. Upon receiving the broadcast, they simultaneously send back packets of ICMP (Internet Control Message Protocol [2]) 'port unreachable' to HostA[1]. This symptom is perceived by the NOBS component in every LODES agent along the return path to HostA. In this case, L1, ..., L7 react to this flow of ICMP packets because these packets traverse each segment of the network. The packet data containing this symptom previously saved by the NOBS is passed to the main LAND component before the diagnostic process is started. Note that LODES agents must react to decide whether or not the problem is tolerable when this symptom is perceived, because it is possible that so many hosts send back the ICMP packets; the gravity of this problem is decided based on the number of the ICMP packets and the line throughput (and the frequency of this problem).

The high-level plan that is created in each LODES agent for this problem is as follows: The goal of the first step is to get basic information such as who sent the original packets that were not understood and who did not understand it, the number of the original packets, and the number of ICMP 'port unreachable' packets (**Get-Basic-Info**). The goal of the second step is to identify both the local route that packets passed through (**Find-the-Route**), and the quality or other limitation of the adjacent lines (**Get-Line-Cost**). The goal of the final step is to decide the gravity of the problem based on the collected data and, if appropriate, to warn the network managers (**Decide-Gravity**) (See Fig. 5). To make this high-level plan concrete, agents map these goals into specific plans (mid-level plans) that can be directly translated to a sequence of operations. For example, the goal **Get-Line-Cost** can be achieved by one of the following mid-level plans: (1) **Get-Max-Throughput**, **Get-Current-Traffic**, and **Estimate-Current-Cost-from-Throughput** sequence, (2) **Kinds-of-Line**, **Get-Current-Traffic**, and **Estimate-Current-Cost-from-Linetype** sequence, or (3) **Get-RTT-between-Both-Ends** and **Estimate-Cost-from-RTT** sequence[2]. In addition, the monetary cost of communication is also calcu-

[1] In order to report that this packet is discarded because the protocol is unknown.

[2] RTT stands for "Round Trip Time."

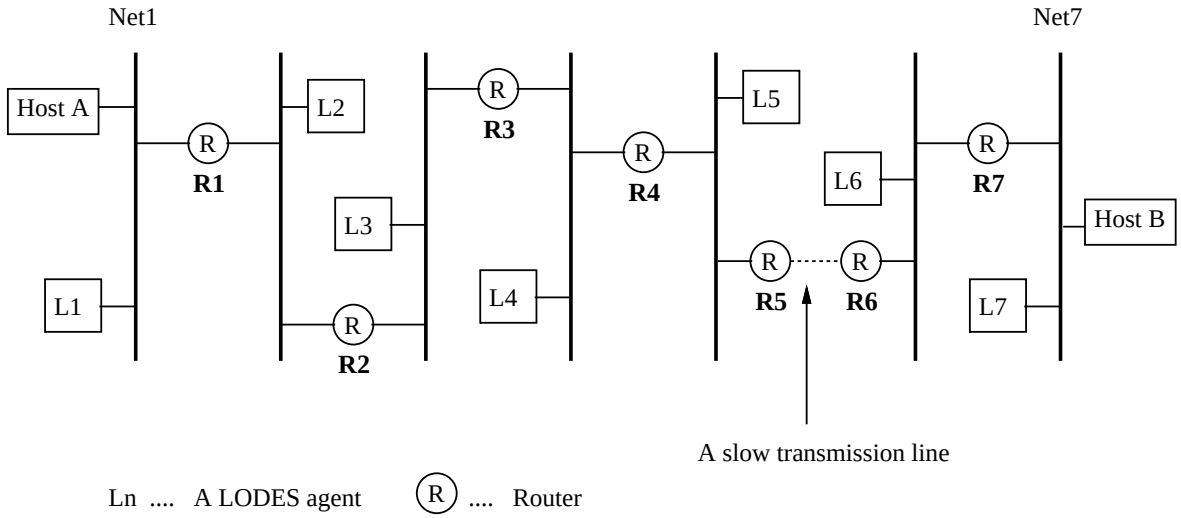


Figure 4: Network Environment for the Example problem

lated if communication occurs over a line that incurs an explicit charge for each communication transaction. If we assume that the adjacent routers do not look at Simple Network Management Protocol (SNMP [2]) packets, then the mid-level plan (3) is the most appropriate one for the current network environment. In the initial diagnosis of this problem, all agents perform only S-cooperation involving the exchange of domain data. As a part of the diagnosis, agents initially exchange basic information about the network segments of both ends. At the end of their initial diagnosis, they all exchange their decisions about the cause of the problem and its severity. This last exchange of information permits agents to recognize when they have come to different conclusions.

An unexpected problem occurs during this diagnostic problem solving when most of the routers in Fig. 4 do not look at SNMP packets and thus plan (3) is performed. The execution of this plan by each LODES agent causes the simultaneous flow of many ICMP echo request and reply packets. This, in turns, triggers the NOBS in each LODES to initiate a diagnostic process to understand this flow. This side-effect problem caused by the diagnostic process is called the *secondary problem* in contrast to the original problem which we call the *primary problem*[3]. Even if the primary problem is tolerated (because the number of ICMP unreachable packets flowing in the primary problem is low), the secondary problem is not, because an ICMP echo request packet is sent intentionally by the LODES whereas an ICMP port unreachable packet is a domain level error message. Thus the final decision of the secondary diagnosis is different. Especially L5 and L6 cannot tolerate the secondary problem because they know their common adjacent line is a slow transmission one. The cause of this secondary problem is that each LODES agent sends

[3]The secondary problem leads to quite similar problem-solving activities as in the primary problem except that the agents reuse the results of the **Find-the-Route** and **Get-Line-Cost**. These results can be reused because they were computed very recently.

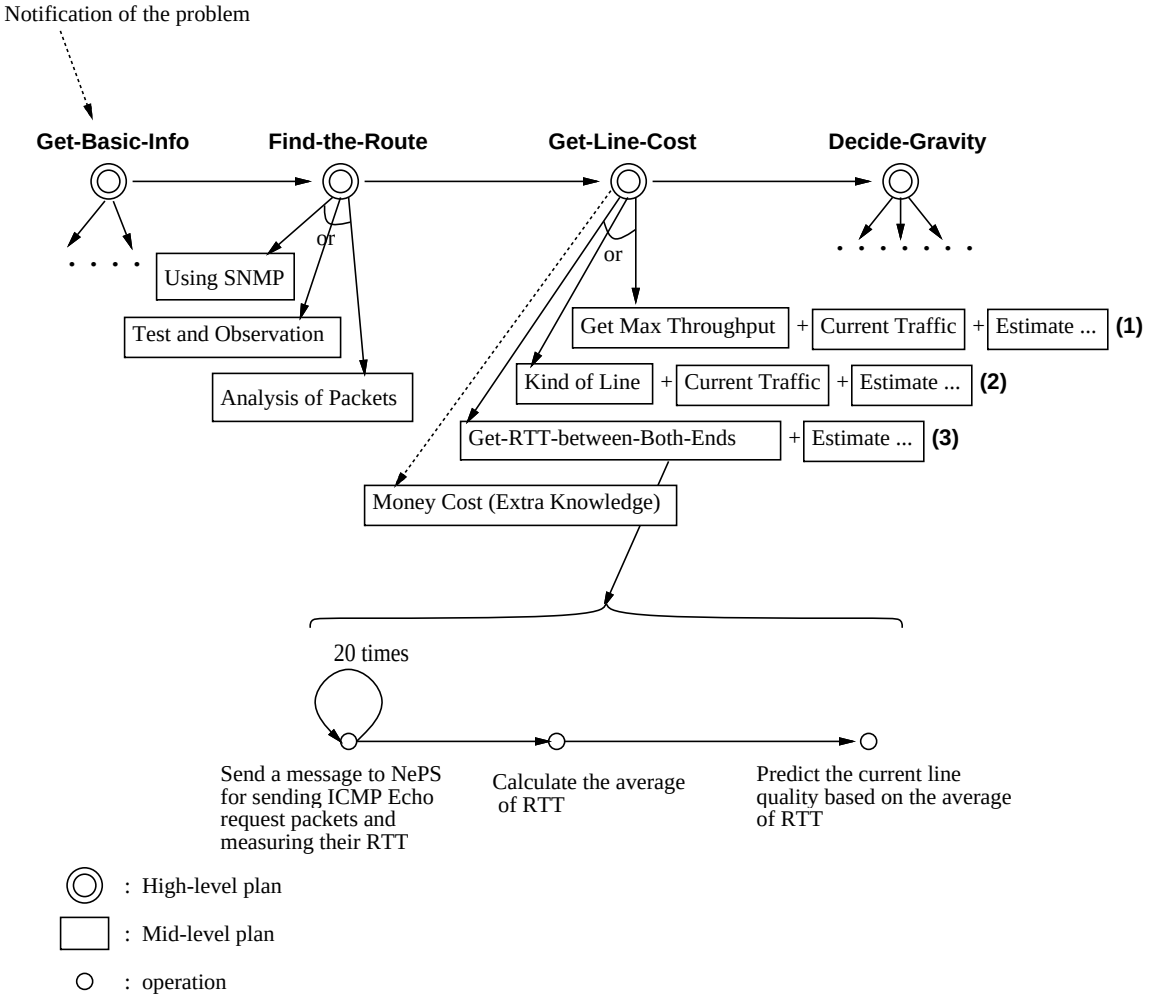


Figure 5: Plans and Operations for the Example problem

many (twenty) ICMP echo request packets and receives these replies (ICMP echo reply packets) for measuring the round-trip-time of messages in the plan **Get-RTT-between-Both-Ends**. This is redundant because the results of each LODES agent executing this plan are expected to be identical.

The secondary problem occurred because the following inferences were not included in local control decision-making:

- That there exists a slow line in the communication path being analyzed. This information is expressed by the variable **MaxThrput** (model of non-local network features)
- That the diagnosis will generate 40 packets (model of agent's actions)
- That other agents redundantly perform the same diagnosis (model of events in other agents)

Thus, to understand this situation and, in this case, decide that only one agent should be executing this plan, a situation-specific D-coordination control plan based on knowledge of the domain model and analysis of their own and other agents' inferences is required.

Our objective in this paper is to describe how agents can analyze these primary and secondary problems and learn coordination rules so that they can avoid recurrence of the secondary problem in the future. Using this example, the framework of our learning method is developed in the following section. A more detailed explanation of how the coordination rules for this situation are derived is described in Section 5.

4 Learning Coordination Plans

4.1 Roles of Learning Coordination Plans in Distributed Problem Solving

Since, in general, CDPS agents are semi-autonomous systems and operate in different local environments, each agent may have different perceptions of what goals or activities are most important to work on and how to best achieve these goals. This means that quite different decision-making may occur at different agents. The lack of information about the problem-solving states of other agents leads to uncertainty in local decision-making which can, in some cases, cause serious problems [16]. Since the costs for all agents exchanging all information seem unrealistically high, it is important to understand what information in (specific) problem-solving situations should be exchanged. Furthermore, even if an agent has the appropriate non-local information, computation costs to explore to the fullest the implications of this information may be impractical. Thus, what needs to be understood is not only what information should be transmitted among agents but also what type of inferencing should be done on this information.

Learning in CDPS can improve the coordination activities by attacking these issues. The initial problem-solving process may be inefficient or fail, but by learning what sort of information

and reasoning is required in a specific situation, more efficient inferencing and more effective coordination can be achieved. In addition, selecting the appropriate coordination level is also possible by understanding what kind of information was exchanged and whether or not agents required non-local models in a similar past situation. This learning can lead to the system doing only the necessary control reasoning for a specific situation, thus avoiding inefficiencies due to both over- and under-coordination.

In summary, learning coordination control in CDPS should result in an understanding from a situation-specific perspective:

- What coordination level is appropriate;
- What situation-specific coordination is appropriate;
- What non-local information is important or redundant in a specific situation;
- When and to whom important information should be sent;
- How to decide the priority of each message and action;
- How to quickly understand the meaning of non-local information.

Note that the first four items suggest that learning in CDPS involves acquiring organizational design-level knowledge, because they are partial solutions to the basic question *which agent does what, when* [9].

4.2 Conceptual Cooperative Learning Model

In this section, a model of network coordination is described in which learning is incorporated. There are three major issues involved in implementing a learning mechanism: when to invoke learning, how to coordinate learning among agents, and when and how to incorporate learned results in other agents.

In general, the learning process should be invoked when the following four cases are recognized:

(a) When a problem-solving process failed; an agent must identify the reasons such as why an important operation was not chosen, and why a negotiation during problem solving did not reach consensus. For example, when a number of agents have scheduled actions which have a conflict, they must learn which agent should first give up its action and select the second best action.

(b) When a problem-solving process took much longer time than the expected time or could not finish within a requested time. This occurs, for example, because agents that played

important roles in a diagnosis were idle for a long time waiting for answers from other agents and an important operation was delayed, or because agents performed redundant local activities or unnecessary coordination activities.

(c) When an unacceptable situation occurred in the environment during problem solving such as an unacceptable amount of network traffic that is generated as a result of the diagnostic process. For example, a LODES agent has an environmental observer which always monitors its local network environment (see also the first example). In general, to detect this type of situation, systems must have the environmental monitoring function and the description of unacceptable situations.

(d) When the system diagnosed a problem not previously encountered or executed a plan not previously chosen. In these cases, the system might efficiently diagnosed it and no learning results may be derived. It is, however, important that the system not only learn new coordination rules but also evaluate its own inference process for a new situation. For example, after a system designer added a new rule, the system can understand that this rule is really selected in a timely manner and works effectively for the diagnosis.

Additionally, the learning process should also be invoked on a periodic basis. There may be latent problems that have not yet manifested themselves. For example, redundant messages and activities in a problem-solving trace could alert the learning system to the need for coordination if at some time later, the network became overloaded or more segments were added. In any case, when the learning process is invoked we call the problem it is working on as a *learning analysis problem* (LAP). The LAP is called *non-local* if it is caused by the lack of coordination. If the LAP is caused by only the lack of local analysis[4], the LAP is called *local*.

As with network diagnosis, communication and coordination among agents involved in learning is also required when there is insufficient local information for analysis to determine the exact cause of the problem. Agents can exchange the following information when working on a LAP:

(1) Full or partial traces of the problem-solving process and the objectives of selected plans.

(2) Requests for analysis of traces.

(3) New coordination rules that remedy the problem.

(4) Required *conditions* for preventing or avoiding recurrence of the same LAP.

For example, (1) and (2) are used for identifying why (non-local) plans or operations were executed redundantly or not executed at the right time. When an agent has more accurate view of the current LAP and it can also create a new coordination rule to avoid the same LAP, the

[4]The example problems are that (1) an important message already arrived but the agent did not read it soon and (2) it takes a long time to understand the acquired non-local information.

rule should be sent to other agents so that they can decide whether or not the proposed rule is acceptable (see (3)). When the proposed rule is rejected or a new coordination rule cannot be generated in the agent, the conditions for avoiding or preventing the LAP are sent to the concerned agents. In this case, each agent finds its own coordination rule which does not violate the desired conditions.

Negotiation may be required to achieve consensus on new coordination rules. For example, when an agent cannot find an alternative plan satisfying the required conditions, it may request permission to use the old plan to avoid unsuccessful or extremely inefficient problem solving. Other agents, especially the agent affected by the LAP, should decide whether using the old plan is acceptable or an alternative plan should be proposed.

We also sometimes identify 'similar situations' for comparative analysis and for applying identified coordination rules. Since most LAPs are concerned with the efficiency of inference, two similar situations of problem-solving processes in this paper are defined as situations where (i) two problems have the same initial local data in the application domain (in internetworking diagnosis, they have the same symptom), (ii) the same diagnostic hypothesis is selected, and (iii) the same sequence of plans is created to confirm the selected hypothesis. We must also consider values of (local and non-local) variables that express the intermediary results of the inference in each situation[5]. However specifying all values may lead to over-specification because some of them may be useless and redundant. To understand which variables are important for specifying the situation, comparative analysis is an important technique if the system has no strong domain theory. Moreover, for distinguishing a problem from others including multiple-cause cases as well as single-cause cases, we introduce the concept of *distinguishability*. Two problems are distinguishable if and only if there is at least one observable result which has a different value in each problem. We assume that when two problems are not distinguishable they are identical.

To extend the definition of 'similar situations,' the concept of *commutative plans* is also introduced since the execution order of plans is uncertain in a distributed environment. Let a sequence of plans P_1, ..., P_n represent the executed order in the mainstream. Plans P_{i-1} and P_i are commutative if and only if P_i refers to the variables defined in or before P_{i-2}, and the execution of P_i prior to P_{i-1} does not affect the execution of P_{i-1}. Using this concept of commutative plans, the equivalence relation among sequences of plans can be defined. The recognition of equivalent plans which implicitly represent similar problem-solving situations. Using this concept, (iii) in the previous paragraph can be substituted for "the equivalent sequence of plans is created to confirm the selected hypothesis."[6]

The control incorporating learned coordination rules is implemented as follows: appropriate HPs, high-level plans, mid-level plans, and operations are usually selected based on the ratings which are locally or non-locally calculated from a payoff matrix or utility numbers, that is, the inference is under the S-coordination. After learning coordination rules, HPs, plans, and op-

[5] In our example, agents must consider the values of MaxThrput in the agents on the route of packets.

[6] To be precise, an agent must distinguish between plans proposed locally and plans induced by messages from other agents. The identification of similar situations in distributed problem solving should be discussed elsewhere in detail.

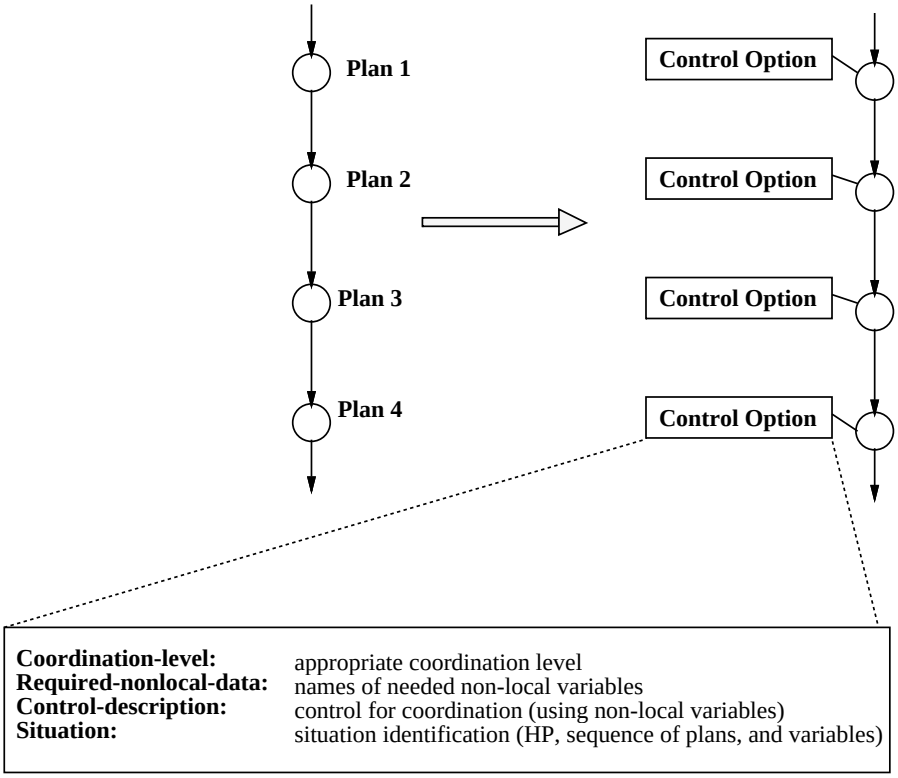


Figure 6: Model of Coordination Control

erations have special coordination annotations, which are called *control options* in this paper (Fig 6). A control option describes the coordination level, needed non-local information, additional non-local control and communications based on this non-local information, and when this coordination is required, that is, deciding whether or not the current situation is similar to one of the past examples. This control option is referred by the Controller before executing the corresponding plans and operations. In our example, a control option corresponding to the plan **Get-RTT-between-Both-Ends** is identified by learning (see Fig. 14). This control option describes how, before performing this plan, agents should identify the existence of a slow line, and if it exists, additional coordination described in the "control-description" is executed.

If an agent does not have sufficient non-local information, it is possible that the agent decides that the current problem is identical to a past problem and other agents decide that these are not identical (and also vice versa). In such a case, this problem-solving process based on the learned coordination rule may fail or execute inappropriate coordination. However, again invoking our learning method, agents can understand what non-local information was important for avoiding the recognized inappropriate behaviors, by analyzing the trace. They then can exchange this information for identifying the problem. We will discuss this topic, using an example, in Section 6.3.

The next issue is whether or not the learned coordination rules should be sent to other agents. If the LAP is non-local, the concerning agents can identify their own new plans or coordination rules. However, should these plans be sent to all other agents? Our current answer is no because the learning results depend highly on their local knowledge, thus, such a learning result is often not useful. Furthermore, the same LAP may not occur in other environments. For example, the secondary problem in our example never occurs in the network where all routers look at SNMP packets or all segments are connected via high-speed lines. However, if only universal domain theory[7] is used to identify a learned coordination rule, this rule may be useful to all agents.

4.3 Traces of Inference

This section addresses what kinds of data should be stored as a trace of problem solving. We assume that each agent is a sequential machine, so that its activities can be described by a sequence of operations. Since messages from other agents can arrive at anytime, a local operation may be suspended or canceled according to the importance of arriving messages.

First we try to express these events and actions sequentially. For example, suppose that a message, M1, arrives during the execution of the operation A1. According to the rating of this message, the agent takes one of the following actions: (1) suspends the current operation, reads M1, and then either cancels or resumes the current operation, or (2) reads M1 at a later time after the completion of some operations. In both cases, activities of this agent are expressed as in Fig. 7. In the case (2), we assume that the message M1 arrives right after the operation A1.

Our trace of problem-solving activities is a sequence of nodes with annotations. This trace is classified into two levels: the plan level and the operation level (see Fig. 8). The plan-level trace is an alternate sequence of plan nodes and decision-state nodes. A decision-state node corresponds to a decision-making process which determines coordination level, a plan to execute, and plans to be discarded. A plan node corresponds to an executed mid-level plan. This plan node is expanded to the corresponding operation-level trace, which is a sequence of operation nodes, where all operations are invoked for achieving the objective of the mid-level plan. All nodes express the following inference activities:

Decision-State Node: Why the current coordination level, the current HP, and the current plan were chosen. Why the current plan was suspended. The (organizational and coordination) knowledge, local data and non-local data used for these decisions.

Plan Node: The expanded sequence of operations for the corresponding plan. Data and long-term knowledge used in constructing the sequence of operations. Used (local and non-local) data, requesting and requested jobs during the execution of the plan. Newly proposed plans and HPs generated as the result of the execution of the plan.

[7] see Appendix 2.

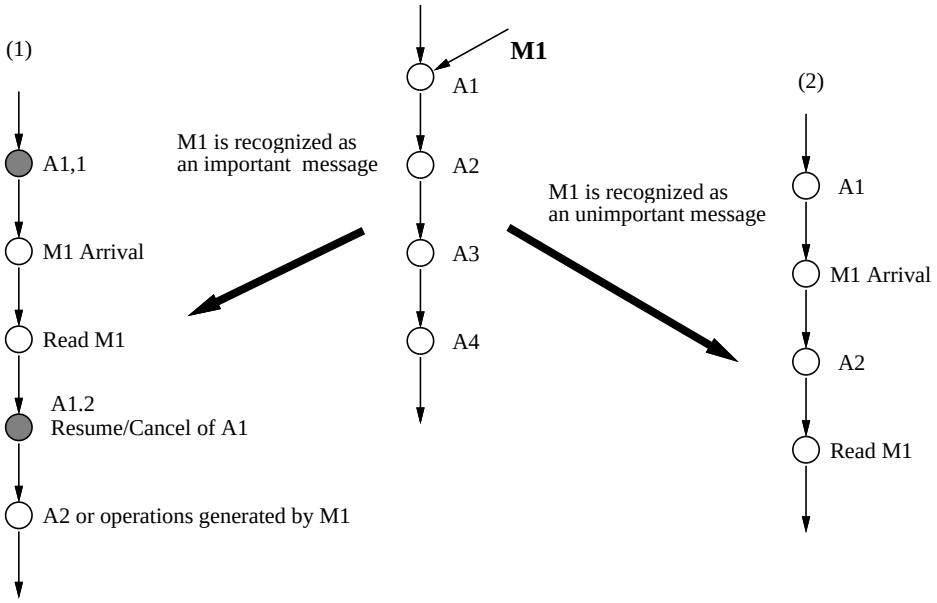


Figure 7: Examples of Sequences of Events and Actions

Operation Node: The variables referred to, the variables that were set, knowledge used for the execution of the corresponding operation. If this node corresponds to the analysis of received data, knowledge used for this analysis.

An annotation describes the important pieces of information for the inference activities of the corresponding node so that the same inference can be re-played. Annotations are described in Appendix 1 in detail. Fig. 8 illustrates a number of example traces. (a) shows the sequence of activities in which Plan B is selected at State 1 and the operations B1 to B4 are executed for this plan. In (b) and (c), the message M1 with a high rating arrives during the operation A1, which is suspended while M1 is analyzed. Then, in the new decision state, the current plan is canceled and the new plan, Plan B, is invoked in (b), whereas the current plan is resumed in (c). A number of actual examples are illustrated in Fig. 12, Fig. 15, and Fig. 16.

4.4 Learning Methods

Before explaining our learning method, an important concept needs to be defined, which we call the *mainstream* of problem solving. Intuitively, the mainstream is a set of activities involving agents that eventually contribute to the completion of the problem-solving process. More formally, the mainstream of problem solving is defined as follows: The final plan that directly led to the final result is in the mainstream. Plans that led to the results referred to in any plan in the mainstream are in the mainstream. Plans that proposed or strongly supported any plan in

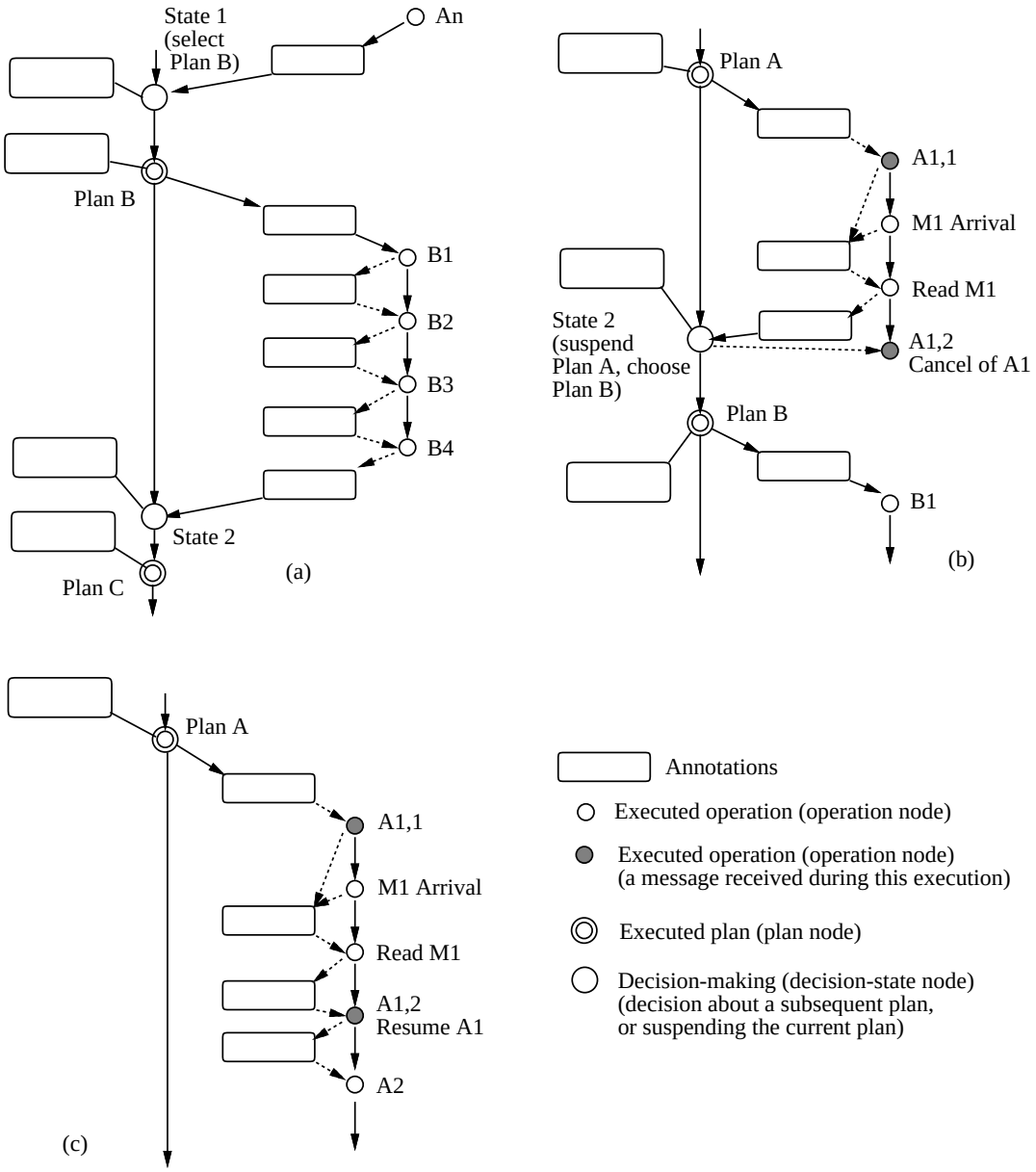


Figure 8: Examples of Traces

the mainstream are in the mainstream. Plans that sent messages that induced any plan in the mainstream are in the mainstream. An operation is in the mainstream if the plan invoking it is in the mainstream. The process of constructing the mainstream may involve activities of other agents when operations requested by these agents or variable values sent by these agents were respectively executed or referred to as parts of the mainstream problem solving. Sometimes a problem-solving process may be prematurely aborted due to a time-out. In this case, the construction of the mainstream requires the system to restart the problem-solving process at the place it was aborted so that a final result can be generated[8]. If agents cannot find any result by applying all possible plans, it implies that some plans are lacking so system managers have to add new plans for this problem and there is no mainstream in this case. Plans or operations which do not belong to the mainstream can be seen as eventually redundant activities in this specific problem-solving situation.

The learning of situation-specific coordination rules follow the steps described below:

Tracing back the mainstream: The first step is to identify the mainstream of problem solving. Agents that produced the final results construct the mainstream from the traces of their problem solving through a backward chaining process. If they used non-local information as part of its problem solving, this backward chaining process spreads to other agents in order to understand their contribution to producing the final results.

Detection of LAPs: The second step involves the identification of LAPs. These LAPs are found by analyzing the mainstream based on the following events:

(1) Variables that were never referred or used;

(2) A mainstream activity whose long delay can be tied directly to the delay of computation of other mainstream activities;

(3) Redundant activities such as operation executions, communications, and negotiations that adversely affect the progress of problem solving associated with the mainstream;

(4) Redundant activities not part of the mainstream that are extremely costly or have interactions with the system's users;

(5) Redundant activities in the mainstream; for example, a variable was defined twice and known values are sent from other agents; and

(6) Understanding of non-local information using non-local domain theory if this process exceeds a certain number of steps or a certain amount of time.

There is no mainstream when the problem-solving process reaches a dead end and no result is produced. In this case, the LAP problem becomes a question of why some plans were not selected, or why a negotiation process failed. If an unacceptable situation is reported

[8]This is necessary to understand what final result should be produced before the time-out. Then our learning method can identify which parts of the problem solving were redundant and eliminate these parts.

by a component external to problem solving (like the NOBS component in LODES), each agent must find which plan causes this situation according to the history of actions and their effects on the environments. In our example, the NOBS component recognized the problem that there was an unusually large flow of ICMP echo packets. The agents then concluded, based on the history of actions, that the plan **Get-RTT-between-Both-Ends** caused this LAP.

Agents cannot easily determine whether a LAP is local or non-local. Though (6) is always local because consuming too much time for reasoning about non-local information is a local agent problem, redundant reasonings that are induced by messages from other agents and that are caused by the lack of appropriate non-local information cannot be directly recognized by a single agent without reference to the activities of other agents.

LAP analysis: The third step involves identifying the reasons for a LAP occurrence, that is, why redundant or inappropriate activities occur. As part of this step, an agent also identifies conditions for preventing or avoiding the recurrence of the LAP. A LAP is usually caused by the lack of appropriate non-local data and/or appropriate rules for the analysis of this non-local data. Another type of problem is that the inferencing required to understand the meaning of non-local data is time-consuming. For example, suppose that, to recognize the meaning of a delivered value, it is necessary to refer to another non-local value. In this case, the LAP may occur because the agent requests the value and may be idle for a long time waiting for the answer. Furthermore, if the agent has to apply the domain theory in this recognition, a new explanation rule to quickly understand the meaning of this non-local data can be derived by applying EBL. Also, comparative analysis is useful in cases where there is not sufficient explanatory knowledge to understand why certain activities are important or redundant. That is, by comparing two similar cases, agents can establish when a certain activity is redundant and when it is not. Furthermore, comparing agent traces is often useful in explaining why they reached different conclusions even though their input data appeared identical. Note that the comparative analysis assumes that the two traces must be similar; thus, if they are completely different or agents are heterogeneous (or have different problem-solving architecture), this technique is not applicable. In our example, the plan **Get-RTT-between-Both-Ends** was executed in many agents simultaneously because their adjacent router did not look at SNMP packets. This is redundant, however, because the result of this plan is usually identical in all agents. Derived conditions for preventing this LAP serve (1) to reduce this redundancy of the plan or (2) to reduce the number of ICMP echo packets.

Coordination Plan Modification: The next step involves following the conditions in the previous step and modifying the agent control structures to incorporate these conditions: (M1) substituting another mid-level plan that is expected to produce the same result, (M2) postponing an action until relevant values are obtained or the appropriate state has been achieved, (M3) changing the order of messages, or (M4) using results obtained from another agent instead of generating them locally. The modifications of local control struc-

tures of an agent may have to be done in conjunction with changes to other agents' control structures in order to have the original modifications achieve the desired effect. In this case, the affected agents need to agree to make the associated changes to their local control structures.

In the example developed previously, the agent requests of its partner agent that both values are sent simultaneously to remedy the LAP since they are only meaningful as a pair. If the partner agent agrees with this proposal, it modifies its local control so that both values are transmitted together. Another example of cooperative modification occurs when an agent can establish that a specific variable value is key to distinguishing whether an activity is important or not. The agent can modify its local control or request the modification of the non-local control from another agent so that this value would be computed or obtained prior to this action, and this action would be done only if this job is really important.

An alternative approach to avoiding the recurrence of a LAP when it involves multiple agents is to have one agent act in a centralized manner rather than instituting a distributed control solution. In this case, one agent, as a representative of this agent network constructs an appropriate non-local coordination plan (and this coordination activity may be performed in a centralized manner controlled by this representative agent). This representative agent is chosen so that it can understand the reason for the LAP and can identify conditions for avoiding the LAP that should be followed by other agents. Proposing the non-local coordination plan by this agent as well as identifying the conditions is appropriate, if it is possible. This centralized learning of coordination plans can also introduce new synchronized activities if the LAP is caused by simultaneous actions of different agents. If it is impossible or if the proposal is rejected by other agents, the identified conditions are distributed into the agent network and each agent builds and modifies its own coordination plan. The situation where the centralized approach is required occurs in the previously described example. In this case, there are only two agents that are seriously affected by other agents and they can understand why they are affected. Thus, one of these two agents proposes a coordination rule in which one of the agents performs the plan **Get-RTT-between-Both-Ends** and others wait for the result by applying the modification (M4). The derived coordination rule is also performed in a centralized manner.

Negotiation: When either proposed coordination plans or proposed conditions for avoiding the LAP cannot be accepted, negotiation among agents is necessary to reach a compromise which is not best but better than ending in a rupture. By exchanging the parts of the conditions that cannot be satisfied, they can decide whether or not it is acceptable. Depending on the compromise results, some agents may select the second best plan. If no agreement is reached, this is an unsolvable problem and reported to system designers. In our example, the derived new coordination plan may be refused by the end nodes L1 and L7 if the primary problem is serious; their activities are much busier than others since they are in the networks where hosts causing and affected by the primary problem are attached. They may understand that the proposed coordination rule will cause a long delay. The alternative proposal is for L1 and L7 to perform the original plan by ignoring coordination

or choosing another plan by which the same result can be expected. Since we assume that the primary problem is not so serious, L1 and L7 can accept the proposed coordination plan.

Situation Identification: An agent also identifies when the learned coordination rules should be applied. In our learning framework, the learned rules are usually applied to similar situations defined in Section 4.2. There are, however, some cases where it is not appropriate to rely on the defined similarity because it is based on local information and non-local information acquired so far; the LAP may occur because of lack of non-local information. In such a case, this required non-local information should be taken into account to accurately identity when new coordination rules should be applied. In our example, not only the proposed HP and plan sequence but also the values of the variable **MaxThrput** in other agents are required to identify whether or not there is a slow line. The result of 'LAP analysis' is useful to understand this missing information. Analysis based on explanatory knowledge and comparative analysis is also performed if further analysis is required. In our example, by comparing the traces of L5 and L6 which made the same decision on the secondary problem and also by comparing the trace of L5 and L4 which made different decisions, the required variable to identify the situation where the proposed coordination is necessary is **MaxThrput**. If agents have strong domain theory and they can explain that only L5 and L6 complained of the secondary problem because of the value of **MaxThrput**, comparative analysis may not be necessary. The detailed explanation of our example is shown in the following section.

The derived coordination control rule is described as a control option which corresponds to an HP, a plan, or an operation. The coordination rule enables the system to decide whether the new plan or the original plan should be selected according to the non-local information that was missing in the previous problem solving.

During the learning process, an appropriate coordination level is also determined. For example, if an HP and the plan sequence to verify this HP is selected, and if no non-local data is referred during this plan sequence, then agents can choose A-coordination when this HP is rated higher than a certain number. Conversely, when agents require additional non-local information and its analysis before a plan is executed, D-coordination should be selected while this plan is executed.

4.5 Proposed Architecture

This subsection describes a new LODES architecture which can incorporate learned, situation-specific coordination rules.

The major changes to this architecture are in the Controller; this module is divided into three submodules: Scheduler, Coordinator, and Organizer. The Organizer determines relatively long-term decisions such as the current coordination level (A-, S-, or D-coordination) and with

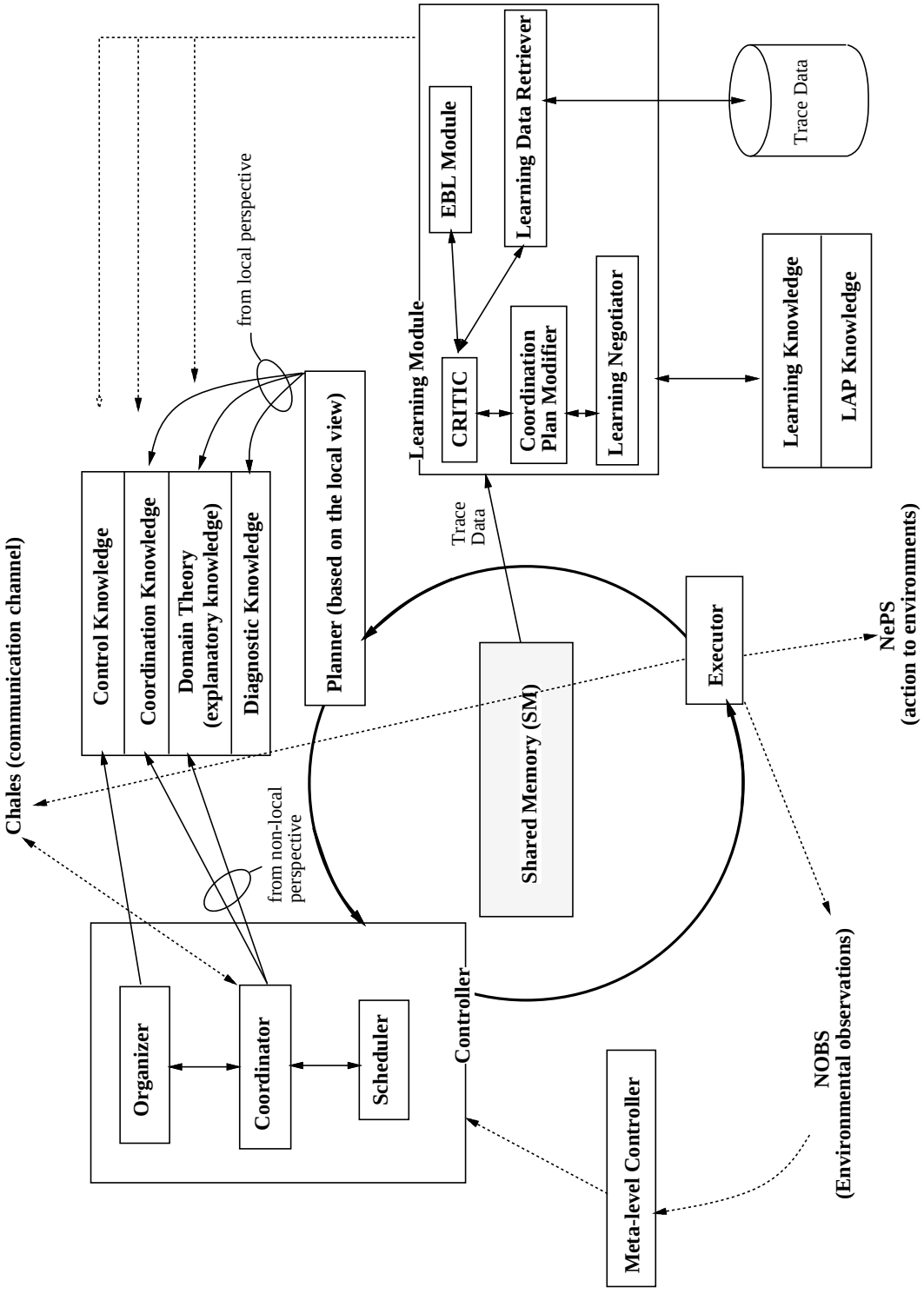


Figure 9: New Architecture of LODES

whom this agent should coordinate. The Coordinator selects appropriate coordinating actions. The Scheduler chooses an appropriate operation out of those proposed by the Coordinator and the Planner. When a plan or an operation is selected by the Scheduler, the Organizer and the Coordinator look at the corresponding control option (as well as the predefined coordination knowledge). According to the control option, the Organizer selects an appropriate coordination level. When there is not a control option specified, the system is under the default coordination level that is S-coordination. The Coordinator performs appropriate actions according to the selected coordination level. For example, when D-coordination is selected, the Coordinator identifies which non-local information is required and how to coordinate with other agents based on the control option. It requests this information or appropriate jobs of other agents, analyzes the acquired non-local results, and proposes appropriate activities to the Scheduler (before the corresponding plan or operation is executed). Other activities of the Coordinator are, for example, sending local information (D/S), understanding non-local data using domain theory (D/S), rating and re-rating requested jobs (D/S), acquiring from other agents their candidate control plans (D), getting current resource usages (D), getting coordination level in other agents (D), informing other agents of the coordination activities in the current specific situation(D), and rating data and jobs being sent (D/S). If A-coordination is selected, the Coordinator postpones processing of received messages from other agents.

For this architectural modification, knowledge is also re-organized; in the previous architecture, knowledge for control and coordination was intertwined with diagnostic knowledge. Diagnostic knowledge includes the set of rules for troubleshooting and a domain theory that describes network behaviors and network features so as to predict local and non-local internetwork activities and states. In the modified architecture, diagnostic knowledge is clearly separable from control knowledge and coordination knowledge that is referred to by the Organizer and the Coordinator, respectively.

Furthermore a Meta-level Controller component is added to LODES so that an agent can not only automatically start a diagnostic process according to a signal from the environmental observer (NOBS) and end the current diagnosis, but can also understand whether or not a learning process should be invoked. For example, a LAP caused by the occurrence of a serious side-effect as a result of LODES agents' activities can trigger learning. According to the Meta-level Controller's decision, traces of problem solving are kept and are analyzed for the subsequent use by the learning process.

The learning module, which achieves the functions described in the previous subsection, consists of five submodules as shown in Fig. 9. The Critic submodule is in charge of 'tracing back the mainstream,' 'detection of LAP,' 'LAP analysis' and 'situation identification' in the learning steps. The Critic invokes the EBL-module to derive important pieces of explanations for quickly understanding non-local information. The Learning Data Retriever stores and retrieves past trace data by indexing based on the definition of 'similar situations.' The Coordination Plan Modifier modifies the coordination rules according to the conditions identified by the Critics in the LAP analysis step. The Learning Negotiator negotiates about new coordination rules. Knowledge for learning is also added into LODES. LAP knowledge is used to

categorize which of the six cases the LAP falls under as described in Section 4.4. Learning knowledge specifies how to modify or replace current plans according to the type of LAP and given conditions for avoiding the LAP.

A learned coordination rule is stored in the following knowledge bases:

- if it is an explanation for quickly understanding non-local information then it is stored in Domain Theory
- if it is a rule to obtain and analyze non-local information then it is stored in Coordination Knowledge (as a control option).
- if it is a rule to decide a coordination level then it is stored in Control Knowledge (as a control option).

These derived pieces of knowledge override previously defined knowledge.

5 Analysis of the Example Problem

In this section, we discuss in detail how coordination rules can be learned from our sample example.

5.1 Analysis of the Unacceptable Situation

5.1.1 Looking at the History of Operations

Because of the secondary problem involving excessive communication over a slow-speed line caused by the operations of LODES agents, the Meta-level Controller decides that learning is necessary after the current problem solving is completed. This decision is broadcast to all concerned agents, L1, ..., L7 to keep the trace of the diagnosis, to start the learning process, and to stand by for the coordination activities for learning.

The first step of learning is for the Critic module of each agent to find the mainstream (Tracing back the mainstream) and then the reason why this unacceptable situation has occurred (Detection of LAPs).

In our example, the NOBS component reports the problem called “ICMP Echo Storm.” By analyzing the collected packet data, the Critic in L5 and L6 realizes that some LODES agents have sent many ICMP echo request packets, and then received the replies to their packets. Furthermore, these Critics understand that only LODES agents L5 and L6 complained of the secondary problem (we choose L5’s Critic as a representative, which we call the R-Critic) though

. . . . .
(21) Measure-RTT called in Get-RTT-between-Both-Ends
(22) Measure-RTT called in Get-RTT-between-Both-Ends
. . . . .
(40) Measure-RTT called in Get-RTT-between-Both-Ends
(41) <Calculate-average> called in Get-RTT-between-Both-Ends
(42) Measure-RTT called in Get-RTT-between-Both-Ends
. . . . .

Figure 10: History of executed operations

other agents were also involved. The R-Critic sends this data to the Critics in other agents and asks them to analyze why they sent these packets in the previous diagnostic attempt. Each Critic finds the operation **Measure-RTT** in which NePS sent twenty ICMP echo request packets and the replies to them (Fig. 10). This operation was invoked in the plan **Get-RTT-between-Both-Ends**.

5.1.2 Using a Network Model

In the second learning step, the causal relation between actions invoked by diagnosis of the primary problem solving and the observed symptom of the secondary problem must be analyzed. For this purpose, agents use the domain model of the internetwork. From collected actions external to the environment and from the domain model, the R-Critic understands that 40 of ICMP echo request packets and their replies flowed in Net5 during the previous problem solving from/to each agent.

5.2 Identifying the Causal Plan to Be Eliminated or Replaced

The Critic must identify why this plan is selected by all agents and find the conditions for avoiding recurrence of the secondary problem (LAP analysis). The Coordination Plan Modifier (CPM) investigates whether it is possible for the plans or operations to be eliminated due to their redundancy, or to be substituted for other plans or operations which are expected to have the same result without causing the secondary problem (Coordination Plan Modification). For the sake of this investigation, the Critic analyzes the objectives of the operations, mid-level plans, and high-level plans causing the secondary problem according to the results of the previous step.

Looking at the objective of the mid-level plan **Get-RTT-between-Both-Ends** in Fig. 11, this plan was executed in order to compute the RTT between both ends. Basically, this result does not depend on the observational point, but it depends on the locations of the end nodes[9].

[9]This knowledge is expressed in the definition of **Get-RTT-between-Both-Ends** since its objective "average-

```
(DEFOPERATION MEASURE-RTT
   (TYPE side-effect time-consuming)
   (PROCEDURE (Measure-RTT-with #loc:Node))
   (Communicate-with
    (NePS
     (EXPECTED-EVENT  ( Send(Self #loc:Node "ICMP echo request")
                        Recv(Self #loc:Node "ICMP echo reply") ))))
   )

(DEFPROC MEASURE-RTT-WITH (x)
  (setq rtt (NePS (format nil "ping to ~A" x)))
  ;; rtt will be 0 if x is the local LODES.
  (if (integer rtt)
      (push rtt #V:RTT-LIST)
    NIL))

(defPlan Get-RTT-between-Both-Ends
   (Objective (average-RTT #V:End-Node-A #V:End-Node-B))
   (Objective-Variables #V:RTT-between-ends)
   (scenario1
     (setq #V:RTT-LIST nil)
     (iterate 20
              (#op:Measure-RTT #V:End-Node-A))
     (when (= (length #V:RTT-LIST) 0)
           (propose-high-level-view #HV:Line-Failure 10))
              ;; 10 ... strongly proposed
     (when (>= (length #V:RTT-LIST) 18)
           (setq #loc:Ave1 (#op:Take-Average #V:RTT-LIST)))
     (when (>= 17 (length #V:RTT-LIST) 1)
           (propose-high-level-view #HV:Line-Bad-Condition 5))
              ;; 5 ... proposed (average)
     (setq #V:RTT-LIST nil)
     (iterate 20
              (#op:Measure-RTT #V:End-Node-B))
         . . . . . .
     ))
```

Figure 11: Example Expressions of Operations, Procedures, and Mid-Level Plan

Therefore, performing this plan in a number of agents is redundant. This result is sent to the CPM and the modification (M4) in Section 4.4 is first applied.

At this stage, the CPM in L5 can build a coordination plan in which one of the agents performs the plan and others will wait for the result. This coordination plan is proposed to other concerned agents because it will be performed in cooperation with other agents. This proposal is accepted in this example since it can generate an answer as accurate as the original one and there is no alternative plan.

5.3 When This New Control Should Be Chosen

The next step is to identify when the new control for coordination is applicable and appropriate. Only L5 and L6 concluded that the secondary problem was intolerable. The R-Critic must identify information that led to this result. Note that the learned coordination rule is not more appropriate than the original one in all situations. In a (normal) environment where all networks are connected via fast lines, excessive communication is tolerable, and the overhead introduced by agent's synchronizing activities for coordination is more costly than excessive communication that results from lack of coordination.

5.3.1 Creating a Partial Global View

To decide whether or not the new control is appropriate, agents must get some additional (non-local) information from other agents (Situation Identification). We describe how to identify this information by comparative analysis in this section. To understand what kind of information is incomplete, two analyses of data are essential: why L5 and L6 made the same decision, and why different decisions are made by others. In our example, L5 can understand that L6 and L4 also observed the same symptom using the network model. The R-Critic must identify which data lead L5 and L6 to the same decision and why L4 can ignore the secondary problem. This data is the non-local information that L5 and L6 have (or do not have) but others do not have (have). By comparing two traces of different but similar reasonings, the R-Critic finds why they reached the same result or the different results.

Before applying the comparative analysis to our example, some variables are eliminated using the network hierarchical model (see Appendix 2). The variable **Echo-Senders** (whose values are names or IP addresses of hosts and are used for identifying the logical locations of the hosts) has multiple values and they belong to different subnetworks; therefore, the result of this diagnosis did not depend on this variable (see Fig. 12).[10] The variable **Echo-Destinations** can also be eliminated in the same way.

RTT" is the function of both ends.

[10] Because, based on the model of "structure of network" described in the Appendix 2, the values of **Echo-Sender** belong to more than two subnetworks.

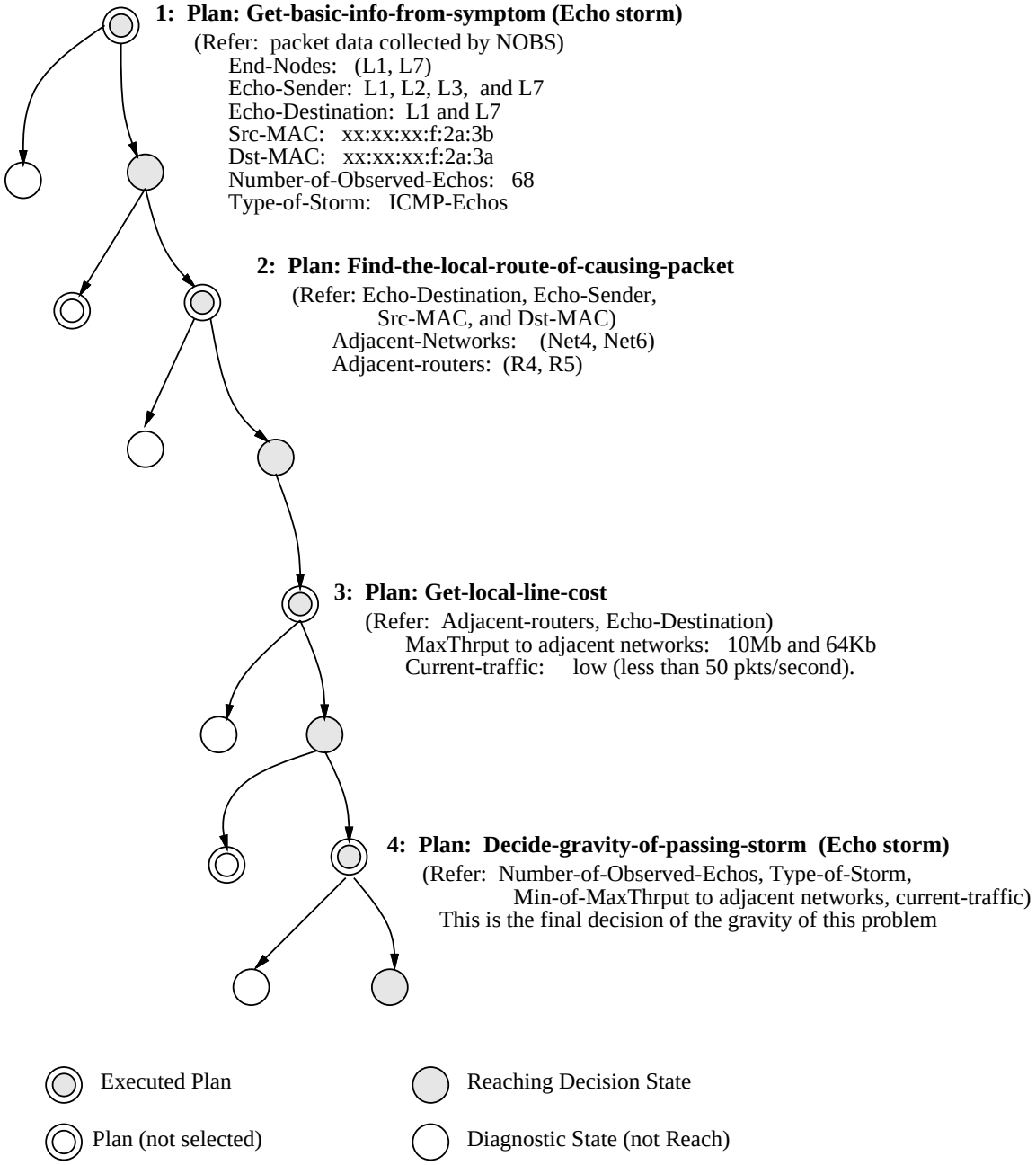


Figure 12: The Trace of the Secondary Problem in L5

The diagnostic trace is expressed by a tree of plan and decision states which alternately appear. After a plan is executed, the process reaches a decision state, and at this state, another plan is selected and executed. This cycle is iterated until the goal is reached.

Table 1: Variable Comparison (L5 and L6)

Variables	values in L5	values in L6	
Adjacent networks	Net4, Net6	Net5 and Net7	eliminated
Adjacent routers	R4, R5	R6, R7	eliminated
End-nodes	(L1, L7)	(L1, L7)	
Src-MAC	xx:xx:xx:f:2a:3b	xx:xx:xx:0:12:8c	eliminated
Dst-MAC	xx:xx:xx:f:2a:3a	xx:xx:xx:0:2f:7e	eliminated
Type-of-Storm	ICMP-Echoes	ICMP-Echoes	
MaxThrput1	10,000,000	64,000	eliminated
MaxThrput2	64,000	10,000,000	eliminated
Min-of-MaxThrput (Quantitative Measure)	64,000	64,000	64,000 (max of L5 and L6's values)
Observed-Number-of-Echoes (Quantitative Measure)	68	61	61 (min of L5 and L6's values)
Current-Traffic	low	low	

Let us compare the traces of the secondary problem in L5 and L6. We assume that they had the identical diagnostic traces as shown in Fig. 12. L5 can understand, using the domain models, that L6 also observed the same symptom. The R-Critic gets the diagnostic trace of the secondary problem from L6 and compares them to understand why they reached the same result even though some local variables have different values (see Table 1).

By this comparison, the R-Critic finds that the variables having different values (except variables of quantitative measures) do not cause the deviation of the two diagnostic processes so they can be eliminated. Quantitative measures are, in general, difficult to eliminate by comparative analysis. However, by taking the weaker condition, in this case, we can decide the new values of these measures.

Let us compare the traces which led to the different results. R-Critic in L5 can understand that the same symptom was observed in L4 but L4 decided that it was not a serious problem. In this diagnosis, L4 had almost the same diagnostic trace (as shown in Fig. 13). L4 also referred to almost the same variables and the differences of their values are very important for deciding the gravity of the secondary problem. As shown in Fig. 13, the final plan deduced different results. This plan referred to the variables, **Observed-Number-of-Echoes**, **Min-of-MaxThrput**, **Type-of-Storm**, and **Current-Traffic**. Thus we can predict that the specific values for some of these variables are important and this fact is also consistent with the results of the previous comparative analysis. By comparing the values of the variables (see Table 2), the **Min-of-MaxThrput**

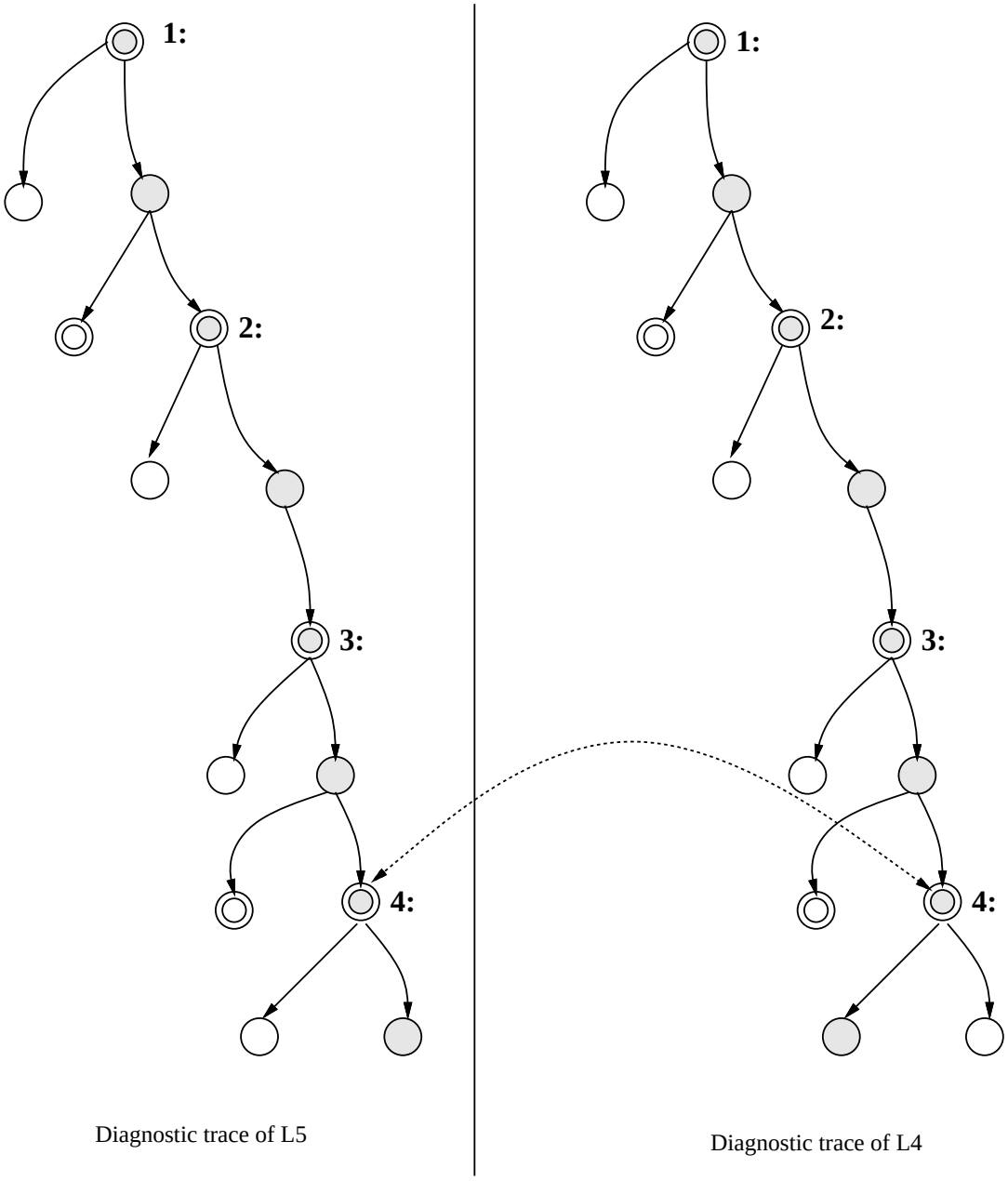


Figure 13: The Trace of the Secondary Problem in L4 and L5

The trace of L5 is identical to Fig. 12 although the plan names and calculated variable values are omitted. The numbers attached to the plan nodes correspond to the plans in Fig. 12. The trace of L6 is also identical to the L5's except the result of the 4th (final) plan.

Table 2: Variable Comparison (L4 and L5)

Variables	values in L4	values from the previous CA	
End-nodes	(L1, L7)	(L1, L7)	eliminated
Type-of-Storm	ICMP-Echoes	ICMP-Echoes	eliminated
Min-of-MaxThrput (Quantitative Measure)	10,000,000	64,000	
Observed-Number-of-Echoes (Quantitative Measure)	68	61	eliminated (almost identical)
Current-Traffic	low	low	eliminated

appears to be the most important variable in understanding the non-local situation for preventing the problematic situation derived by the primary diagnostic process. This variable describes the minimum value of the maximum throughput capacities of the links.

According to the analysis described above, an agent can now learn that when it chooses the plan **Get-RTT-between-Both-Ends**, the values of **Min-of-MaxThrput** in other agents, that is, the existence of the slow transmission line, is quite important non-local information. Then an agent will be able to acquire the values of this variable in other agents before it performs the learned coordination action. This coordination rule is stored with coordination knowledge as a control option described in Fig. 14. This means that before performing this plan, an agent collect the values of the variable **MaxThrput** from agents along the paths to L1 and L7 and, if there is a slow transmission line, one of the agents should perform this plan and others should wait for this answer. **Allocate** is a primitive coordination function which can allocate this plan task to an appropriate agent, and the function **Wait-for-the-result-of** waits for this answer if the local agent is not allocated this plan task.

It is significant that D-coordination was selected here. This is because, to decide who should do this plan, non-local meta-level information such as resource usage, scheduled plans, the network structure and features[11], and whether another agent has already done this plan task in a different context must be understood. This function is implemented in **Allocation** under D-coordination. When some agents do not have this coordination rule, the agent that knows this rule has to acquaint these agents with this rule so they will not perform the plan themselves. These coordination activities are implemented by the Coordinator under D-coordination[12]

[11]For example, an agent which is located in the left side of the slow line in Fig. 4 should perform this in order to reduce the number of packets through the slow line.

[12]Although this is an important issue, this paper does not cover it. We need further discussion on this situation. Note that Section 4.2 describes how no learned results are sent to other agents in general. However, in this case, these participating agents actually face the similar problem-solving situation where the learned rule may be useful.

Note that if L4 or L6 does not observe the secondary problem, this comparative analysis will be incomplete and the derived incomplete result may cause redundant communication and excessive coordination because of less generalization. For example, if L4 missed this problem[13], L5 could not eliminate the variables in Table 2 by comparative analysis, so that agents would need to exchange and analyze five variables described in Table 2. This is not only redundant and over-coordinating but also misses some situations where the learned coordination rule is required. Thus agents must wait for another occurrence of a similar problem to prevent this problem. Also note that if the system has a strong domain theory and can explain why L5 and L6 complained but others did not, this comparative analysis is not necessary; the EBL algorithm can be directly applied.

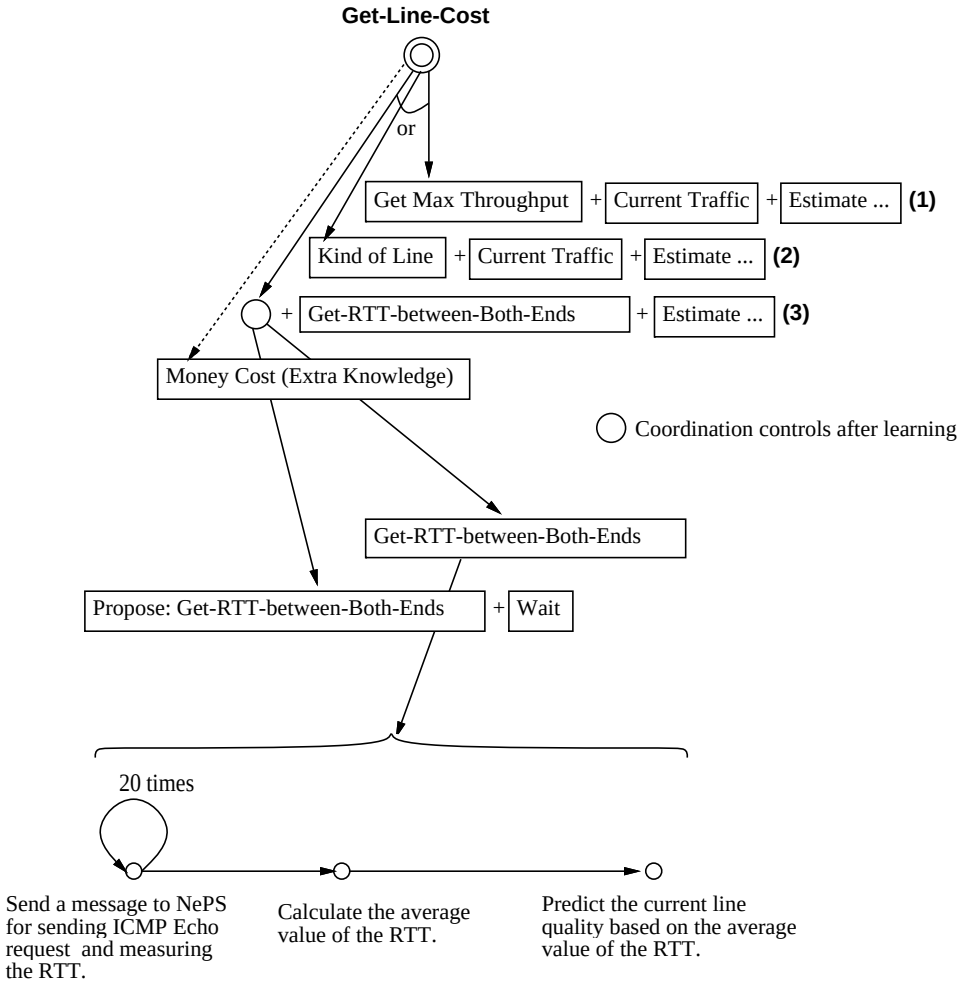
5.3.2 In Order To Choose Appropriate Control

In general, appropriate plans and operations should be decided by their communication cost[14], CPU cost, and accuracy (and the network manager's policy). In the preceding discussion, a coordination rule is derived. However, if the system can understand its own control information about plans and operations such as necessary time for execution, number of packets transmitted, accuracy of their results, and required CPU resources, the Coordination Plan Modifier can gracefully determine another appropriate plan. For example, when an agent node is requested to execute the plan **Get-RTT-between-Both-Ends** and it knows of the existence of a slow line, the number of packets required in the coordination activity is important. Thus, the number of the requesting agents should be taken into consideration in deciding the priority of the plan. According to the comparison of the two controls (see Table 3), the requests of more than three agents makes the priority of the derived coordination activity higher.

Even if an agent proposes the plan **Get-RTT-between-Both-Ends** to another node, this node may have a different viewpoint and find other plans and tasks that it decides are more important than the proposed plan in a similar problem. In such a situation, a network control regime similar to generic partial global planning (GPGP) [3] is required. Even if the local priority of a plan is low, if other agents request the plan and its result facilitates their reasoning, it should be performed. How to decide plan priorities, especially for plans requested by other agents, is one of the difficult issues for real-world systems. An agent must take into account quantitative evaluations of non-local requested actions as well as their logical causalities. These quantitative evaluations, for example, include how many agents request the action, importance of the expected result, and the action's resource usages (CPU and I/O). The learned result can be reflected back into the GPGP algorithm by understanding which non-local data are important for deciding priorities of plans and, thus, how to coordinate among agents.

[13] For example, NOBS in L4 might drop some packets because of the flow control or the limitation of CPU power.

[14] Communication cost is determined by max throughput, current traffic, and line tariff cost. Thus, communication cost is determined by non-local data.



Coordination-level: D [deep coordination]
Required-nonlocal-data: MaxThrput from agents along the path
Situation (pre-condition: (min-of MaxThrput < low)
HP: Decide-gravity-of-ICMP-unreachable-storm
high-level-seq: (Get-Basic-Info Find-the-route Get-line-cost Decide-gravity)
high-level-plan: Get-line-Cost
mid-level-seq: (Get-RTT-between-Both-Ends Estimate-Cost-from-RTT)
mid-level-plan: Get-RTT-between-Both-Ends)
Control-description: (Allocate Get-RTT-between-Both-Ends)
(Wait-for-the-result-of Get-RTT-between-Both-Ends)

Figure 14: The Learned Coordination Rule to Prevent Redundant Work

Table 3: Comparison of Two Control Methods

	Original Control	Learned Coordination Control
# of packets	20 packets per agent (simultaneous flow)	20 packets by an end agent + a number of packets for coordination
Communication between agents	none	request, wait, and answer
CPU	Low CPU resource	High CPU resource for coordination and scheduling
Idle time	relatively small (1 to 3 seconds expected)	Varies (100ms to infinity)
Others	independent actions	Synchronized actions

6 The Second Example Problem

This section addresses, using another example, how the proposed learning method identifies new explanations so as to quickly understand non-local data and also identifies the priorities of actions and messages in cooperative problem solving.

6.1 The Second Example (Part I)

We assume the same network environment as Fig. 4. The problem symptom is that, from HostA in Net1, telnet to HostB does not work (the error message "Timeout" is displayed in the HostA's display). Suppose that the cause of this problem is that the network interface in HostB has a hardware problem. In this case, the 'user' reports the symptom to L1[15] and the diagnostic process of L1 is started. In the first stage of the diagnosis, L1 determines that the problem is not local and coordination with L7 is necessary, so L1 activates L7 by reporting this problem. Fig. 15 and Fig. 16 show the traces of the diagnosis by L1 and L7. For example, in L7, first the plan **Understand-Current-Problem** is selected to get a more detailed identification of the problem that L1 reported to L7, then based on the acquired data in this plan, some possible causes are proposed as the HP. Next, the highest rated HP **Interface-broken-or-poweroff** is selected (state A2), and a high-level plan is created to verify this HP. The Controller in L7 selects the mid-level plans and operations based on this high-level plan while this HP is still under consideration. We assume that L1 and L7 diagnose this problem under S-coordination. In the following sections,

[15]The 'user' means the user of HostA or the network manager. This problem is reported to L1 because the user using HostA is near L1.

we describe only the learned coordination rules and omit the detailed description of the learning process. In this discussion, we use the notion (V = A,L) to express that a variable V has a value A in an agent L.

6.2 Getting Explanation Between Local and Non-Local Data

During the plan **Check-ARP-reply**[16] in L7, the value of the variable **ping-remote**[17] in L1, which is NIL, arrived with a high rating, and was thus analyzed. Since this data is explainable from the local observed value **ping-local** in L7 and the network domain model, and is consistent with the current HP, it does not offer any important information. Furthermore, this analysis is computationally expensive because it must use a non-local domain model in the calculation. If explanation-based learning is applied to this situation, L7 can directly conclude that

ping-local = NIL in the local-end $\longrightarrow$ **ping-remote** = NIL in the remote-end.[18]

(or If (**ping-local** = NIL, local-end) then (**ping-remote** = NIL, remote-end))

Then, the agent can quickly understand whether or not this received message is worth analyzing. This is not a coordination rule but is useful for efficient coordination; thus, it is kept as an explanation in domain theory. Moreover, this value can be deduced only from the universal domain theory (see Appendix 2), and thus is useful in all LODES agents.

6.3 Autonomous Processing (A-Coordination)

Although the meaning of (**ping-remote**(HostB) = NIL, L1) can be quickly understood after the explanation rule is learned as discussed in the previous subsection, it is still true that L7 is distracted by this message. Looking at the trace of L7, the verification of the HP **Interface-broken-or-Poweroff** was strongly supported by the value of **ping-local**(HostB) = NIL in L7, and it was unnecessary to refer to non-local information to verify this HP. Applying our proposed learning method to this case, the coordination rule (we assume that "no coordination is necessary" is also one of the coordination rules) derived here is that, if the HP **Interface-broken-or-Poweroff** is strongly supported and selected, and if the plan sequence to verify this HP is identical to this example problem, then a reference to any message from other agents is not necessary to process immediately. That is, A-coordination should be selected in this case.

[16] ARP stands for Address Resolution Protocol. This plan verifies that the host responds to the ARP request packet.

[17] The variable **ping-remote** expresses whether the remote host (HostA if the observer is L7, and HostB if the observer is L1) can reply to an ICMP echo request packet. The variable **ping-local** expresses whether the local host can reply to an ICMP echo request packet. For clarification, these variables are often expressed with the observed host. For example, (**ping-remote**(HostA) = T, L7) means that L7 sent a ICMP echo request packet and could observe the response from HostA.

[18] L1 and L7 are substituted for "remote-end" and "local-end" respectively.

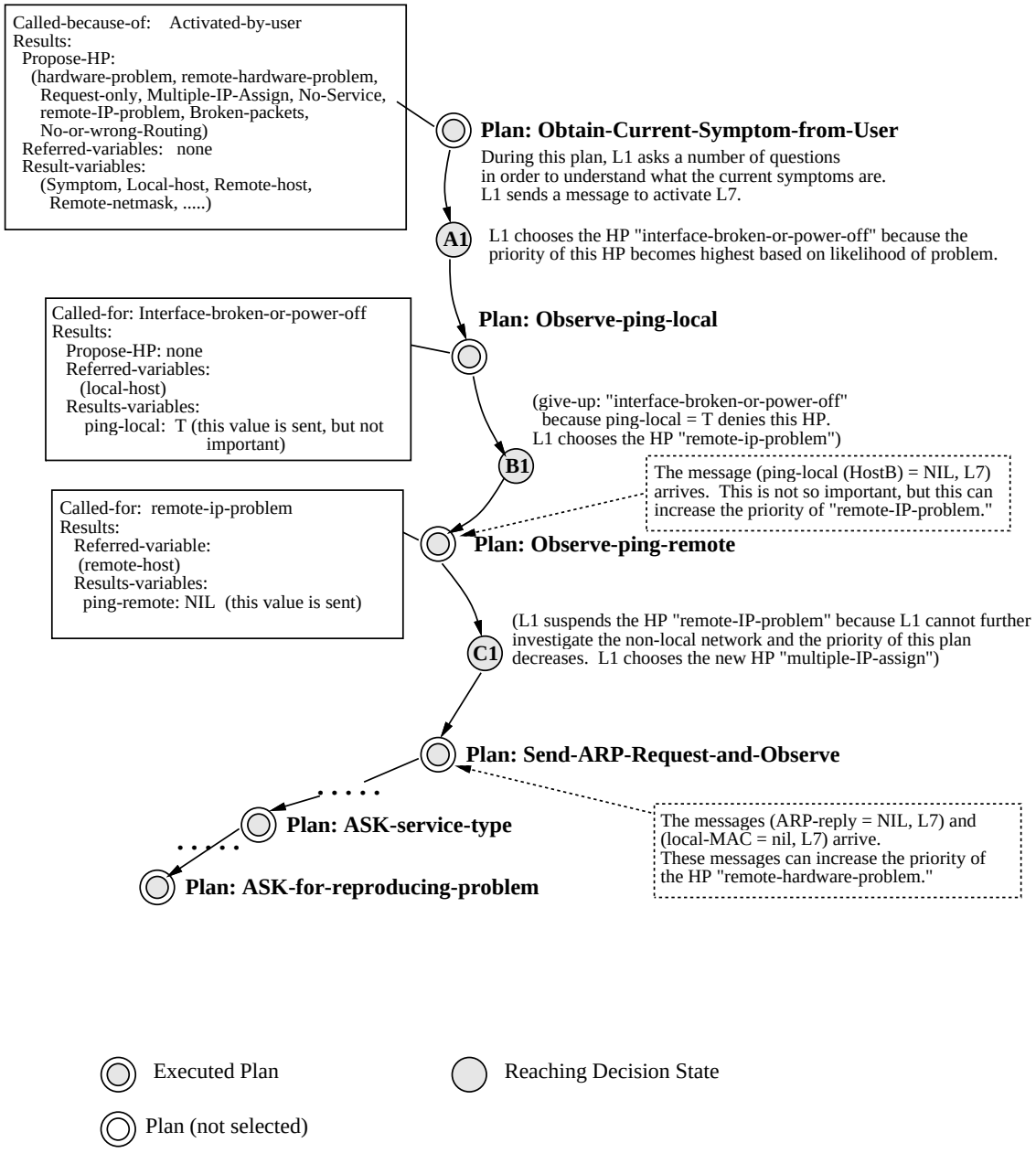


Figure 15: Trace of L1 (the second example — part I)

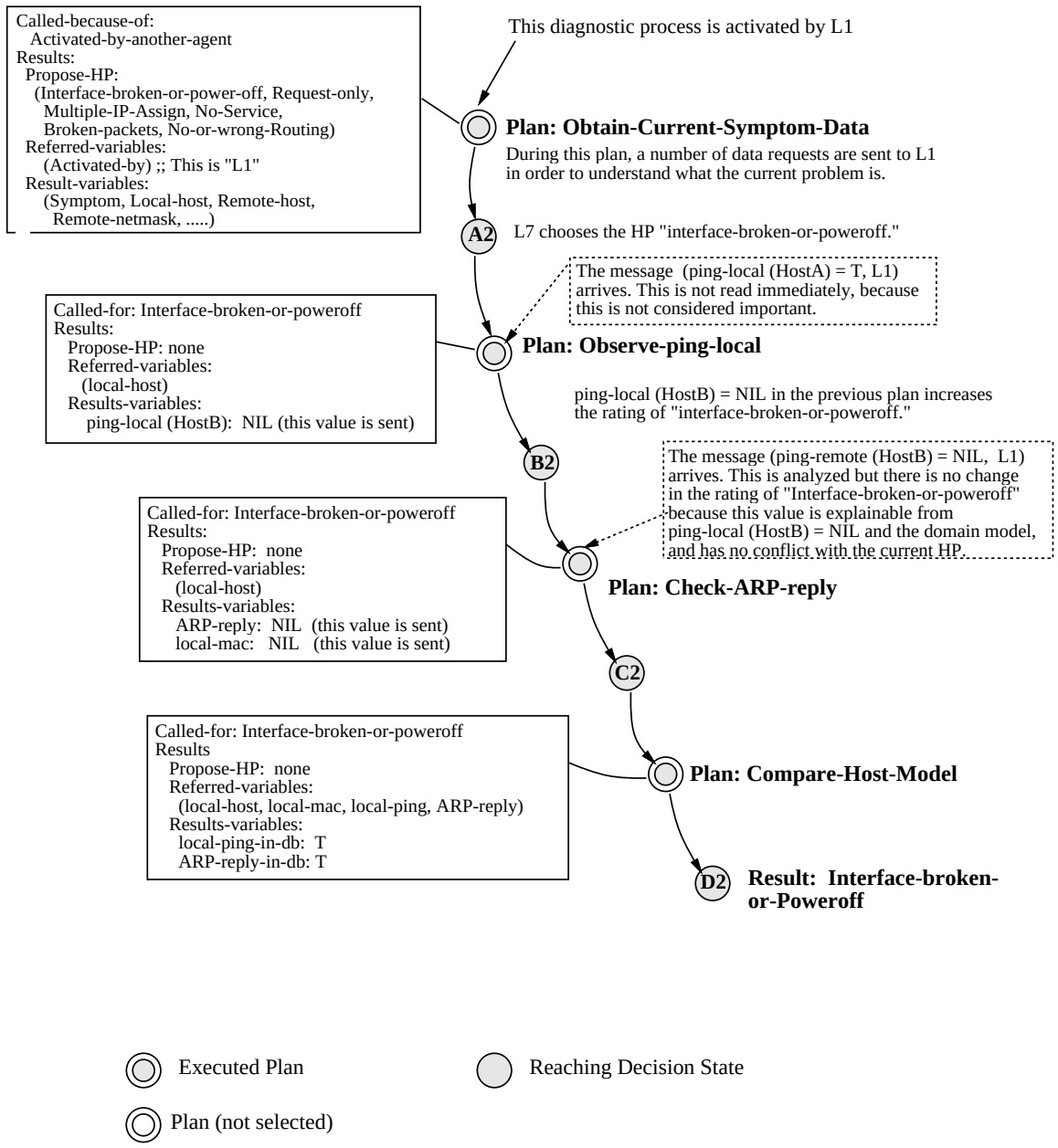


Figure 16: Trace of L7 (the second example — part I)

We must discuss here the differing perspectives among agents. For example, L7 selects A-coordination in a different problem according to the learned rule because of the lack of non-local information available to identify the difference between the problems. However, by applying our learning method again after this failure, agents can understand which non-local information is necessary before agents shift to A-coordination. We must also consider the cost of coordination here. If much information is necessary to select A-coordination, it may be better to give up shifting to A-coordination and to infer under S-coordination. More precise reasoning about their own controls is necessary as discussed in Section 5.3.2.

6.4 Avoiding Wasteful and Expensive Tasks – Find a new situation where coordination is required

If a selected operation in an agent is expensive in terms of CPU resources used or time requirement, or has interactions with users, the agent should take into account the possibility that other agents can diagnose this problem without resorting to this expensive operation. Let us focus on L1's trace. Eventually L1 could not efficiently diagnose the problem and was also not requested to do any job for L7 during its diagnosis. Thus, L1 selected one HP after another until the message indicating that L7 had found the cause arrived. During this process, L1 may select a plan that includes a resource- and/or time-consuming operation or a special operation interacting with the user such as **Ask-Service-Type** (ask the user what kind of protocol was used when the problem occurred) and **Ask-for-Reproducing-problem** (ask the user to reproduce the problem). If neither agents can find any highly-rated HP, these tasks may be helpful, but obviously these are wasteful in this example because the problem can be diagnosed without asking the user. This kind of expensive operation should be postponed while one of the other agents chooses a highly rated HP. In such situations, an agent should look at other agents' plans. That is, before executing an expensive operation, an agent should take into consideration states of other agents' inferences. This action may be done under S-coordination where the agent sends a message asking whether or not other agents have an HP that is rated more than a predefined threshold. However, D-coordination is sometimes required to understand proposed HPs, plans to verify them, their ratings, and logical connections between these plans and local actions. For example, a highly rated plan in another agent has to refer to the result of this local, expensive operation. As another case, the in-progress execution of highly rated plan in other agents makes this local plan impossible. In these cases, through positive and negative experience, an agent can understand the necessity of D-coordination by the proposed learning method.

To understand whether or not an operation is expensive, there are two methods: First, by syntactically analyzing an operation, its time requirements are estimated from the predefined primitive functions of which the operation consists. However, it is not easy to completely understand how much CPU resource is needed and how much time can be consumed by sophisticated operations under this method. The second method is to estimate the resource and time requirements from the history of executions. In LODES, the second method is applied because it is applicable to and appropriate for continuous systems like LODES.

6.5 The Second Example (Part II)

It is not true that the message (**ping-remote**(HostB) = NIL, L1) is unimportant in every situation. To clarify how to evaluate the importance of this message, a slightly different example is introduced. In this new example, the network environment and the symptom of the problem are the same, but the cause of the new example is that HostB has no routing definition to Net1. From the viewpoint of L1, both examples look like the same problem. However, from L7's perspective, HostB is working correctly because L7's interaction with HostB does not force HostB to communicate to Net1. Fig. 17 and Fig. 18 show the diagnostic traces of L1 and L7 for this problem.

6.6 Identification of Importance of Messages

In part I of the second example, a message (**ping-remote**(HostB) = nil, L1) is not important but in part II, it is. Differences between these cases are the value of **ping-local**(HostA) in L7; in part II these observational values conflict, but in part I they do not. Understanding this conflict is an expensive task as described in Section 6.2, but the conflict is explainable from the network model. Thus, by applying the EBL algorithm, the explanation to quickly identify this conflict can be derived and thereafter the inference becomes more efficient in this situation as described in Section 6.2.

Even under S-coordination, deciding appropriate ratings of messages and job requests is quite important. Discussion in the previous paragraph suggests that ratings of messages and job requests are not static because the ratings may vary according to values of other variables. In this example, L1 could understand how important this message is if it had looked at the message (**ping-local**(HostB) = T, L7). L1 did not immediately analyze this message because of the low rating set by L7; from L7's perspective, this message suggests the correct behavior rather than a problem so its rating is set low by L7.

Before sending the value of **ping-remote**, L1 should check whether or not the value of **ping-local**(HostB) in L7 has arrived in order to understand the importance of this message. This is possible because, after identifying the derived explanation to detect the conflict, L1 can quickly understand the importance of this message depending on the value of **local-ping**(HostB) in L7. Of course, L1 should not wait for this message indefinitely. If L1 realizes that the value of **ping-remote**(HostB) in L1 is not important, and if L7 is currently overloaded or communication cost is high, L1 can just not send the message since L7 probably does not need this result to arrive at the correct conclusion.

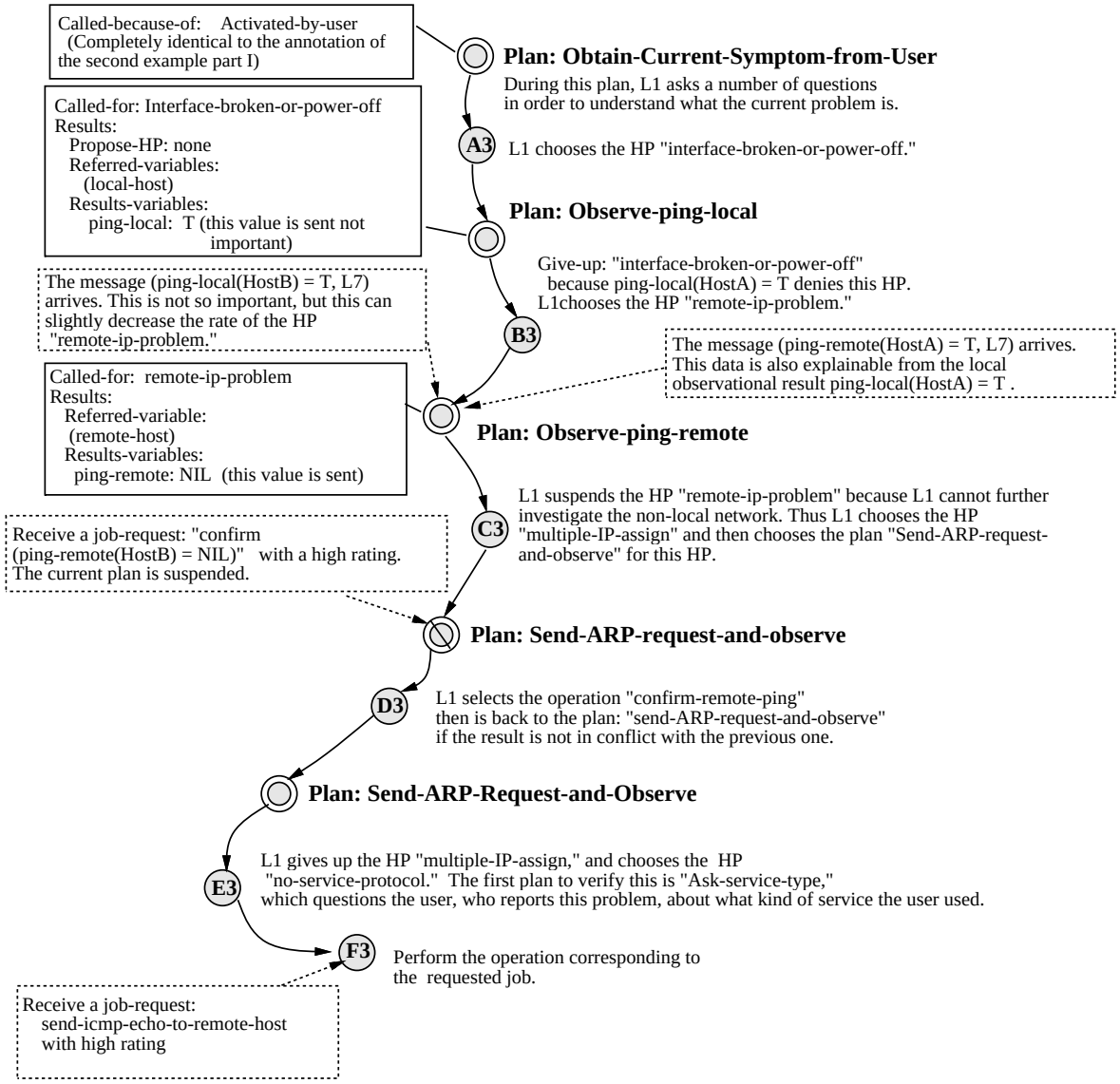


Figure 17: Trace of L1 (the second example — part II)

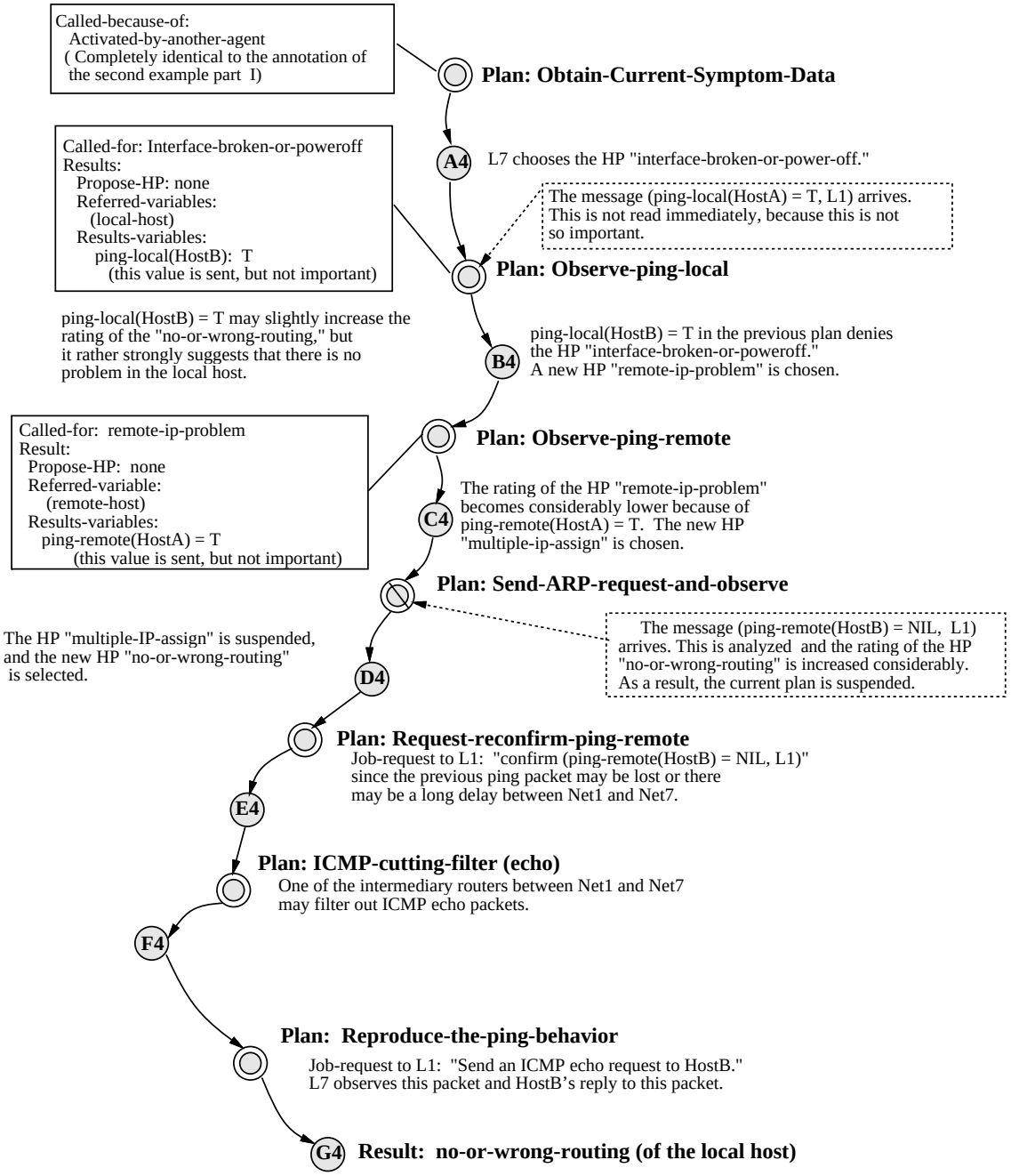


Figure 18: Trace of L7 (the second example — part II)

7 Discussion and Future Research

Coordination is certainly essential in distributed problem solving but it is not always necessary; excessive coordination leads to inefficiency and redundant communications. Thus, it is important to identify how and when to coordinate and this is the motivation of our research. We believe that our proposed learning is important for all CDPS systems, especially continuous, systems because it can identify situation-specific coordination rules which are adaptive for their own domain problems and environment. Other research has addressed this topic through dynamic organizational design [8, 13] and partial global planning [6, 7] where dynamic meta-level information is used to achieve an appropriate level of coordination. These coordination algorithms are always applied to all situations and usually involve extensive communications and inferences by agents. Our research is also related to the previous work on the development of a diagnosis module (DM) [12] that reasons about the behavior of a single-agent problem-solving system. The DM compares an actual behavior with a model of problem-solving behavior through a technique called comparative analysis. This permits the DM to identify correct and incorrect behavior when there is a weak domain theory. [14] also discusses what to learn from a failure posed on which component of their chess-playing program has failed. In [24], cooperative learning is proposed in which there is an interaction board where all agents can interact and enumerate positive/negative examples and intermediary results of a rule being learned. However, this research assumes consistency of data in all agents and identifying coordination rules is not discussed. Shoham and Tennenholtz discuss another interesting approach to learning cooperation rules in multiple agent systems [22, 23]. In this work, agents adapt themselves to their current work and environment on a game theoretical basis. A simple rule for deciding strategy called 'cumulative best response' is introduced. They demonstrate that, using this rule, all agents can reach a common cooperation strategy after many trials. They also experimentally illustrate how efficiently agents reach the common strategy under various configurations of system parameters such as memory size and communication restrictions.

In the future, we see our research being extended in the following directions. If there is no strong domain theory, case-based learning for planning and meta-level control [10, 28] is useful and also applicable to learning coordination rules; the past positive control for the similar situation is modified and employed to guide the coordination controls for the current problem solving. Another direction is to understand appropriate coordination actions by statistical analysis [4]. From a number of examples and analysis of problem-solving activities, agents statistically estimate when a specific type of coordination is useful. We feel that both of these directions can be merged into the EBL and comparative analysis methods proposed in this paper, depending on the kinds of problems and kinds of situations. We also feel that our proposed method can be extended to the development of coordination strategies that deal with more indirect interactions among agents due to resource requirements of independent problems. For example, when multiple agents handle different problems, a number of their actions can be merged or cannot be performed simultaneously because they use the same resources [17]. In the event of multiple problems occurring simultaneously, we have to investigate the possibility that the multiple problems may arise from a single cause and agents may observe it at different points — an issue

especially important for internetwork diagnosis. Agents must identify the relations among these problems using local data, acquired data and the domain theory, but this is an expensive task. We hope that the learning method proposed here can be extended to derive rules for quickly and accurately understanding these relations.

There are also a number of other issues that need to be investigated. Explicit reasoning about the timing of operations is not currently used in LODES. If LODES agents can reason about time, they can, in the first example, find another coordination rule in which each agent performs the action of 'sending test packets' sequentially. Furthermore, the perception of activities of hosts in the network, including other LODES agents, has some delays; these delays may lead agents to inappropriate activities. More generally, we believe that the proposed learning is applicable to identifying the appropriate activity based on time-constraints in distributed real-time systems. Asynchronism also occurs in distributed systems. The order of actions and messages is often different in the same problem, and sometimes these are not commutative. Comparative analysis may identify the important order of these actions and messages from positive and negative examples of problem solving, but if agents can simulate other possible problem-solving scenarios by changing the order of actions, a critical order of actions can gracefully be predicted. When learned coordination rules can be shared and with whom is also another issue. In the proposed method, rules learned only from universal domain theory or troubleshooting knowledge, which are identical in every agent, can be shared; otherwise, knowledge in agents may have conflicts. However, there may be some coordination rules which are not shared according to our definition but are still useful for another agent in a specific group or in a similar environment. "Problem decompositions and allocations" are also an important issue. We conjecture that how to decompose a problem and who should solve these decomposed subproblems are predictable from past examples, even though it is not discussed in this paper.

Creating a more formal model for analyzing distributed problem-solving behaviors is another important topic. For example, we define some notions such as "commutative plans," and "similar situations." We must discuss these notations using a formal model so that they can be applied in more general distributed problem solving. Additionally, in our learning, an agent identifies some inefficient activities and then creates control and coordination rules such as postponing unimportant messages, changing the order of actions that may be requested by other agents, and obtaining needed non-local information. It is necessary to discuss, in a more general manner, what kind of control (adding/eliminating/re-arranging plans, operations, and messages) can scale the solution up.

A number of LODES agents are currently working at NTT Laboratories and at the Department of Computer Science, University of Massachusetts at Amherst. They can detect and diagnose many network problems, though LODES is still under development. We find that LODES is actually useful for network managers and computer network users. The learning method discussed here is currently being implemented. The LAP detection and LAP analysis steps have already been implemented. The learning method will be evaluated in the near future.

8 Conclusion

This paper presents an approach to learning coordination plans. Many AI applications require cooperative distributed problem solving where coordination is an essential technique. An agent must choose globally coherent activities based on local and acquired non-local information. Furthermore, understanding when and which non-local information should be exchanged is important for efficient coordination inferences since understanding everything about other agents is impractical.

This paper discusses the introduction of a learning component into a CDPS system that can identify, in a situation-specific manner, what type of coordination is required, what priorities to associate with messages and actions, and what non-local information is necessary. Our proposed learning method is invoked when the system finds a problem in its own diagnosis or a problem that is caused by the agent's activities is reported by other agents or external components. To find these problems (LAPs), the system must be able to monitor itself. LAPs occur, for example, when: (1) diagnosis fails; (2) diagnosis cannot finish within a requested time; (3) the execution of a plan takes much longer than the expected time; and (4) an unexpected side-effect occurs. Our learning method keeps traces of inferences and analyzes them after problem solving. It then uses the EBL technique and comparative analysis (if there is no strong domain theory) in order to identify important and redundant activities, missing information, and situations that occurred as the result of agents' activities. Using the information, our learning component derives the coordination control rules for deciding: what information should be sent immediately; the priorities of messages; and the local or non-local action that should be taken next in order to promote global coherency. These can be seen as organizational rules in the sense that they decide which agent does what and when.

We also propose three coordination levels: nearly autonomous, shallow, and deep coordination. Agents must choose the appropriate coordination level depending on features of the problem, situation, and environment in order to achieve efficient inferences. The learned coordination rule is stored with organizational and coordination knowledge as a control option that corresponds to a hypothesis, a plan, or an operation. A hypothesis, plan, or operation is chosen based on the local plan, requested jobs, and acquired non-local data. Before it is executed, the Coordinator and the Organizer refer to these learned rules to decide the appropriate coordination level and coordination actions.

Two example problems are used to describe how our proposed learning method works in actual situations. The first example problem involves the transmission of redundant test packets through a slow communication link because of a lack of coordination among agents that are simultaneously diagnosing the same problem. Our method can determine what non-local information is necessary and when agents should initiate the derived coordination activity for preventing similar problems from occurring in the future. The second example problem illustrates inefficient diagnosis. Our method can identify (1) the explanations that allow the system to quickly and accurately understand non-local data, (2) the appropriate coordination level, and (3) the priorities of messages for efficient reasoning. Finally, related research and further research

issues are discussed.

References

[1] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," RFC1157, 1990.

[2] D. Comer, *Internetworking with TCP/IP, Volume I; Principles, Protocols, and Architecture* (Second Edition), Prentice Hall, Englewood Cliffs, NJ, 1991.

[3] K. S. Decker and V. R. Lesser, "Generalizing the Partial Global Planning Algorithm," *Int. Jour. of Intelligent Cooperative Information Systems*, Vol. 1, No. 2, pp. 319-346, 1992.

[4] K. S. Decker and V. R. Lesser, "Analyzing the Need for Meta-level Communication," to appear in *Proc. of AAAI-93*.

[5] G. Dejong, "Generalizations Based on Explanations," *Proc. of 7th IJCAI*, pp. 67-69, 1981.

[6] E. Durfee and V. R. Lesser, "Using Partial Global Plans to Coordinate Distributed Problem Solvers," *Proc. of 10th IJCAI*, pp. 875-883, 1987.

[7] E. Durfee and V. R. Lesser, "Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formulation," *IEEE Trans. on System, Man, and Cybernetics*, Vol. 21, No. 5, pp. 1167-1183, 1991.

[8] E. Durfee and T. Montgomery, "Coordination as Distributed Search in a Hierarchical Space," *IEEE Trans. on System, Man, and Cybernetics*, Vol. 21, No. 6, pp. 1347-1362, 1991.

[9] L. Gasser, "Social Conceptions of Knowledge and Action," *Artificial Intelligence*, Vol. 47, pp. 107-138, 1991.

[10] K. J. Hammond, *Case-Based Planning — Viewing Planning as a Memory Task*, Academic Press, 1989.

[11] E. Hudlická, V. R. Lesser, J. Pavlin, and A. Rewari, "Design of a Distributed Diagnosis System," COINS Tech Report 86-63, Univ. of Massachusetts, 1986.

[12] E. Hudlická and V. R. Lesser, "Modeling and Diagnosing Problem-Solving System Behavior," *IEEE Trans. on System, Man, and Cybernetics*, Vol. 17, No. 3, pp. 407-419, 1987.

[13] T. Ishida, M. Yokoo, and L. Gasser, "An Organizational Approach to Adaptive Production Systems," *Proc. of AAAI*, 1990.

[14] B. Krulwich, "Determining What to Learn in a Multi-Component Planning System," *Proc. of the Cognitive Science Conf.*, 1991.

[15] V. R. Lesser and D. D. Corkill, "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks," *AI Magazine*, Vol. 4, Fall, pp. 15-33, 1983.

[16] V. R. Lesser, "A Retrospective View of FA/C Distributed Problem Solving," *IEEE Trans. on System, Man, and Cybernetics*, Vol. 21, No. 6, pp. 1347-1362, 1991.

[17] F. von Martial, *Coordinating Plans of Autonomous Agents*, Lecture Notes in AI 610, Springer-Verlag, Berlin, 1992.

[18] K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II," RFC1213, 1991.

[19] R. Michalski and R. Stepp, "Learning from Observation: Conceptual Clustering," in *Machine Learning — An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Tioga Publishing Company, 1983.

[20] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli, "Explanation-Based Generalizations: A Unifying View," *Machine Learning*, Vol. 1, pp. 47-80, 1986.

[21] J. R. Quinlan, "Learning Efficient Classification Procedures and their Application to Chess End Games," in *Machine Learning — An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Tioga Publishing Company, pp. 463-482, 1983.

[22] Y. Shoham and M. Tennenholtz, "Emergent Conventions in Multi-Agent Systems: Initial Experimental Results and Observations," *Proc. of KR-92*, Boston, 1992.

[23] Y. Shoham and M. Tennenholtz, "Co-learning and the Evolution of Coordinated Multi-Agent Activity," in preparation, 1993.

[24] S. S. Sian, "Adaption Based on Cooperative Learning in Multi-Agent Systems," *Decentralized A. I. 2*, Y. Demazeau and J.-P. Müller (Eds.), Elsevier Science Publishers, pp. 257-272, 1991.

[25] T. Sugawara, "A Cooperative LAN Diagnostic and Observation Expert System," *Proc. of IEEE Phoenix Conf. on Comp. and Comm.*, pp. 667-674, 1990.

[26] T. Sugawara and K. Murakami, "A Multiagent Diagnostic System for Internetwork Problems," *Proc. of INET'92*, Kobe, Japan, 1992.

[27] T. Sugawara, "Using Action Benefits and Plan Certainties in Multiagent Problem Solving," *Proc. of IEEE Conference on Artificial Intelligence Applications*, pp. 407-413, 1993.

[28] M. M. Veloso and J. G. Carbonell, "Derivational Analogy in PRODIGY: Automating Case Acquisition, Storage, and Utilization," *Machine Learning*, 1992.

Appendix 1: Annotations of Nodes in Traces of Inferences

Decision-State Node:

Coordination-level:	The current coordination level
Why-current-coordination-level:	Why is the current coordination level chosen (Pointer to corresponding organization knowledge)
List-of-coordinating-agents:	All agents participating in the current problem solving
Current-HP:	The selected HP
Proposed-HP-and-ratings:	All proposed HPs and their ratings
Why-current-HP:	Why is the current HP selected (the list of variables, their values and pointers to the referred knowledge)
Current-high-level-plan-sequence:	The sequence of the current high-level plan to verify the current HP.
Supporting-variables:	The facts supporting the current high-level-plan sequence
Higher-level-plan:	The current high-level plan
Current-plan:	The selected (mid-level) plan.
Proposed-plan-and-ratings:	All proposed plans and their ratings
Why-current-plan:	Why is the current plan selected (the list of variables, their values, and pointers to the referred knowledge)
Request-jobs:	The requesting jobs for this decision-making and pointers to referred coordination knowledge which causes these requests.
Request-data:	The requesting non-local data for this decision-making and pointers to coordination knowledge which causes these requests
Referred-domain-theory:	Pointers to referred domain theory

Plan Node:

Name:	The name of the corresponding mid-level plan to this node
Operator-sequence:	The current operator sequence
Possible-operator-sequences:	Other possible operator sequences
Supporting-local-variables:	Why this operator sequence is selected
Higher-level-plan:	The high-level plan proposing this mid-level plan
Proposed-plans-and-ratings:	The name of plans which are proposed during the execution of this plan.
Why-plan-proposed:	Why are these plans proposed.
Proposed-HP-and-ratings:	The name of HPs which are proposed during the execution of this plan.
Why-HP-proposed:	Why are these HPs proposed.
Referred-knowledge	The pointers to the referred knowledge during this plan
Referred-data:	The referred local data during this plan
Referred-nonlocal-data:	The referred non-local data during this plan
Calculated-data:	The calculated data during this plan
Sent-data:	The sent data during this plan
Received-data:	The received data during this plan

Operator Node:

Type-of-operations:	a local operation, cancellation or resuming of the selected operation, an analysis of the received data
Proposed-by:	The plan or the message that causes this operation
Referred-variables:	The referred variables to execute this operation
Defined-variables:	The defined variables as the results of this operation
Referred-coordination-knowledge:	Referred coordination knowledge for executing this operation
Referred-domain-theory:	Pointers to referred domain theory
Referred-troubleshooting-knowledge:	Pointers to referred troubleshooting knowledge
Communication-activities:	Communication history during this operation

Appendix 2: Models of Network and Other Agents

The following describes the universal and local network domain theory model used by a LODES agent. Note that the model that describes the "structure of network," the "basic behaviors of packets," the "connection non-local actions and local observations," and the "model of behaviors of network nodes based on the protocol knowledge" are identical in any agent. Other models are different depending on the local network environments and the problems. Thus, the former models are called *universal domain theory* and the latter are called local domain theory.

Basic definitions: These are defined for simple descriptions.

- Functions for a packet ∗d
  - The source of ∗d (Src(∗d)), the destination of ∗d (Dst(∗d)), and the protocol type of ∗d (Type(∗d)) are defined in ∗Lodes if Observable(∗d,∗Lodes), where ∗Lodes is an LODES agent.
  - The network address portion of a host (NetAddr(∗Host)) is always defined if ∗Host is bounded, where *Host is a host node in the network.
- Logical definitions

  SrcDst(∗Host1,∗Host2,∗d) $\leftrightarrow$Src(∗d) = ∗Host1 $\wedge$ Src(∗d) = ∗Host2

  BelongTo(∗Host1,∗Net) $\leftrightarrow$NetAddr(∗Host1) = ∗Net, where ∗Net is a network address

  Recv(∗Host1,∗Host2,∗d) $\leftrightarrow$Send(∗Host2,∗Host1,∗d)

  Send(∗Host1,∗Host2,∗d) $\rightarrow$

  Manage(NetAddr(∗Host1),∗Lodes)$\wedge$ Observable(∗d,∗Lodes)$\wedge$ SrcDst(∗Host1,∗Host2,∗d),

  where Manage(∗Net,*Lodes) is true if *Lodes manages the network ∗Net.

Structure of network: This is the model of a hierarchical network structure described in Fig. 19, which means that any network consists of a number of host nodes, a set of subnetworks is also a network, and any network is a subnetwork.

Local network model: This is the model of the local network including the network that the local agent should manage, physical connections of networks, the maximum throughput, the maximum transfer unit size, the number of local hosts, the addresses of the local routers, the addresses of adjacent networks, and routing definitions in the local routers.

For example, if Self = L5 in Fig. 4

- Managing network

  BelongTo(Self,∗Net) $\rightarrow$ Manage(∗Net,Self)

  (Manage(∗Net,Self) Optional)

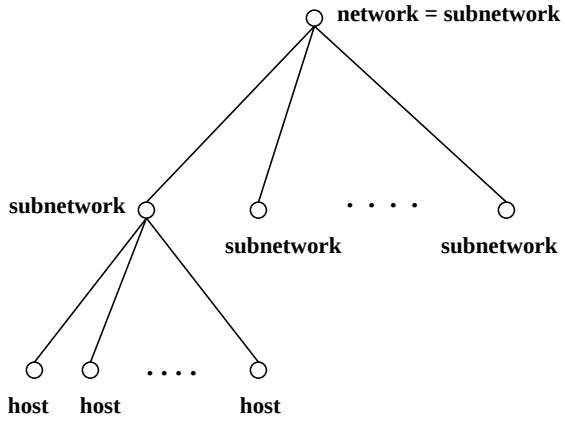


Figure 19: Hierarchical Structure of Networks

- Netmasks

  NetMask(∗Net,∗Mask) (which means that ∗Mask is the netmask of the network ∗Net)

  (Example: NetMask(129.60.41.0, 255.255.255.0))

- Adjacent networks (long-term refresh cycle)

  Adjacent(Net4,Self), Adjacent(Net6,Self),

  which means that Net4 and Net6 are physically connected with the local network segment.

- Local routing definitions (short-term refresh cycle)

  Route(∗DestNet,∗AdjacentNet,Self)

  (For example, Route(Net1,Net4,L5) means that if a packet whose destination is Net1 is found by L5, then the packet will be forwarded to Net4)

This model is automatically created by observing flowing packets and communicating with the local routers (most of these data must be given by network managers if the local router does not look at SNMP (Simple Network Management Protocol [1, 18]), since these are essential for network management). Note that "Adjacent Networks" expresses the physical connectivity so this data is refreshed in a long time cycle, although "Local Routing Definitions" is refreshed in a short time cycle because it is logical and frequently varies.

Basic behaviors of packets: This knowledge expresses the routing of packets. Depending on their destination address, they will pass to an appropriate adjacent router and network according to the local routing definitions. These expressions also mean that all of the passed packets are observable by one of the adjacent LODES agents.

Observable(∗d,Self) ∧ SrcDst(∗Anyhost,∗Host,∗d)

$\wedge$ BelongTo(*Host,*DestNet) $\wedge$ Manage(Self,*DestNet)

$\rightarrow$ Reach(*Host,*d)

Observable(*d,Self) $\wedge$ SrcDst(*Anyhost,*Host,*d)

$\wedge$ BelongTo(*Host,*DestNet) $\wedge$ Route(*DestNet,*AdjacentNet,Self)

$\rightarrow$ PassTo(*d,*AdjacentNet,Self)

PassTo(*d,*AdjacentNet,Self),$\wedge$ Manage(*Adjacent,*Lodes)

$\rightarrow$ Observable(*d,*Lodes)

Connection of non-local actions and local observations: This formula describes that if a packet is observed, all other packets whose source and destination address are identical to it are also observable. Note that "IsObserved" is valid only for a very short time, because this knowledge assumes no changes of routing definitions in routers. Since a dynamic routing method is used in actual network, the result of this formula may not be correct after a certain time.

IsObserved(*d1,Self)

$\rightarrow \forall$*d [Src(*d1) = Src(*d) $\wedge$ Dst(*d1) = Dst(*d)$\rightarrow$ Observable(*d,Self)]

Participating agent network model: This model expresses the agents participating in the current problem solving. When a problematic packet is detected, agents in its source and destination network become "end nodes" (End). According to the destination of this packet, adjacent agents of the route of this packet (CAdjacent) are identified.

(0) Who participates in the current problem solving
L1, L2, ..., L7

(1) Each agent knows Both Ends of the causing packets thus
End(L1), End(L7)

(2) Each agent knows both neighbors of the current agent network
CAdjacent(L4,Self), CAdjacent(L6,Self)
;;Assuming Self = L5

Model of other participating agents: Each agent has the partial model of other agents including current plans and requested jobs of the local agent. If necessary, agents can get current operations executed and values of any variables to make more global views.

Currentplan(*PlanName,*agent)

Requested(*Job,*agent)

(this means that *agent requested *Job of the local agent.)

Model of observed local network: During problem solving, an agent makes a model of the local network. This model expresses the current state of the local network although the local network model expresses the normal state. This model is more detailed than the local network model.

Model of other networks: During problem solving, agents can indirectly observe other networks by sending test packets. For example,

NoResponse(∗Host,ICMPecho)

means that there is no response to ICMP (Internet Control Message Protocol) echo request packet [2] from non-local host ∗Host.

Model of behaviors of network nodes based on the protocol knowledge: This model describes that protocol-related behaviors of network nodes. For example, when a host receives an ICMP echo request packet, it must reply to it, and when a host receives an unknown protocol packet, it must send back an ICMP port unreachable packet to the source of the packet.

$\forall$∗d [Type(∗d)= "ICMP Echo Request" $\wedge$ Send(∗Host1,∗Host2,∗d)

$\rightarrow$Observable(∗d',Self)$\wedge$ Send(∗Host2,∗Host1,∗d')$\wedge$Type(∗d')="ICMP echo reply"

Observed(∗d',Self)$\wedge$ Send(∗Host1,∗Host2,∗d')$\wedge$Type(∗d')= "ICMP port unreach"

$\rightarrow \exists$∗d [∗d = Orig-packet(∗d') $\wedge$ Send(∗Host2,∗Host1,∗d) $\wedge$ unknownproto(∗Host1,Proto(∗d))]