

Applying coordination Mechanisms for Dependency Relationships under Various Environments *

Wei Chen
Computer and Information Sciences
University of Delaware
Newark, DE 19716
wchen@cis.udel.edu

Keith Decker
Computer and Information Sciences
University of Delaware
Newark, DE 19716
decker@cis.udel.edu

ABSTRACT

Coordination is a key functionality in multi-agent systems, and mechanisms for achieving coordinated behaviors have been well-studied. One important observation has been that different mechanisms have correspondingly different performance characteristics, and that these can change dramatically in different environments (i.e., no one mechanism is best for all domains). A more recent observation is that one can describe possible mechanisms in a domain-independent way, as simple or complex responses to certain dependency relationships between the activities of different agents. Thus agent programmers can separate encoding agent domain actions from the solution to particular coordination problems that may arise. This paper explores the specification of a large range of coordination mechanisms, for the common hard “enablement” (or “happens-before”) relationship between tasks at different agents. It also explores the impact of task environment characteristics on the choice/performance of these mechanisms. Essentially, a coordination mechanism can be described as a set of protocols (possibly unique to the mechanism), and as an associated automatic re-writing of the specification of the domain-dependent task (expressed as an augmented HTN). The idea about the separation of general knowledge and domain-dependent knowledge is explained. A general method to address the relationships between application domains and agent coordination is introduced. This paper also presents a concrete implementation of this idea in the DECAF agent architecture and an initial exploration of the separation of domain action from meta-level coordination actions for eight simple coordination mechanisms.

1. INTRODUCTION

Generally speaking, there are three ways to coordinate multi-agent systems: arrange the problem so as to avoid the need for coordination, associate coordination mechanisms with

each individual agent, or construct a special agent that functions as a centralized coordinator. All of these solutions have good and bad points, and many possible realizations in different environments. If we are to make progress toward a general theory of coordination, we must look at what information is needed to enumerate and choose between these alternatives. If we are to build intelligent multi-agent systems, we must look at how to represent and reason about this information computationally. If we did this, each agent would be able to learn over time which coordination mechanism is the best for a specific situation, according to its knowledge of its capabilities and beliefs towards other agents in a dynamic environment.

Previous approaches towards coordination have explored different perspectives. The study of coordination over external resources is popular, and has produced several demonstrations where complex or knowledge-intensive mechanisms can be detrimental for both individuals and a system as a whole [16, 17, 20]. Other work has focussed on particular classes of coordination mechanisms, such as contracting [3] or organizational approaches [19, 23, 12]. However, not all systems depend on external resource coordination alone, and it is not clear how to integrate multiple approaches (since no single approach is likely to be good all the time in many realistic environments). In this paper we present a general approach to explore the relationships between application domains and agent coordination.

An important observation, made independently by Decker [5] and Castelfranchi [1], is that the need to coordinate comes about because of the relationships of agents goals, the planned tasks to achieve those goals, and required resources (including not only external resources but also private resources and capabilities). Such relationships can be both positive or negative, hard (necessary) or soft. Thus an important way to express coordination mechanisms in a more general way is to express them with respect to these task interrelationships.

Our work involves a general solution to this problem. It is based mainly on two previous achievements, TÆMS and GPGP. TÆMS (Task Analysis, Environment Modeling, and Simulation) [6] proposes a formal approach to represent the tasks and their relationships in a domain-independent, quantitative way. It is based on annotating HTN (Hierarchical Task Networks) [8] with both basic information about

*This work is supported by NSF Grant No. 9733004.

the characteristics over which an agent might express utility preferences (e.g. cost, quality, duration, etc.) but also how the execution of some action or subtask *changes* these characteristics at another subtask—thus setting up the potential for some coordination to occur (or not).

The problem is not only to represent potential coordination in a general way, but to develop an algorithmic approach by which many different coordination mechanisms can be integrated. GPGP (Generalized Partial Global Planning) [5] is a domain-independent, extendible approach for expressing coordination mechanisms. Although inspired by PGP, GPGP is not tied to a single domain, allows more agent heterogeneity, has a wider variety of coordination mechanisms, and uses a different method of mechanism integration. The key to the integration used by GPGP is to assume each agent is capable of reasoning locally about its schedule of activities and possible alternatives. Thus coordination mechanisms of many different styles can be thought of as ways to provide information to a local scheduler that allow it to construct better schedules. This is especially useful in practice where considerable work has been done in tuning an agent's local scheduler for a particular domain. Furthermore, it has been shown that it is "... useful to separate domain-specific problem solving and generic control knowledge" [2]. Our approach allows for such a separation of concerns: the behaviors of domain problem solving, versus behaviors for coordination.

This paper concerns an implementation of these ideas using DECAF (Distributed Environment Centered Agent Framework). DECAF [11] is an architecture and set of agent construction tools for building real (non-simulated) multi-agent systems, based on RETSINA [7]. DECAF is being used to build applications in electronic commerce, bioinformatics, and plant alarm management. Unlike many API-based agent construction toolkits, DECAF provides an operating system for each agent—a comprehensive set of agent support behaviors to which the casual agent programmer has strictly limited access. The GPGP module described here is one of these agent OS-level subsystems.

Section 2 analyzes the hard dependency relationship, "enables". Section 3 describes eight specific coordination mechanisms in detail. Section 4 discusses the implementation of the mechanisms in the DECAF architecture. Section 5 states some initial experimental results. Section 6 presents a general approach to understand the relationships between application domains and agent coordination.

2. TASK STRUCTURES AND THE ENABLES RELATIONSHIP

To achieve its desires, an agent has to select appropriate actions at suitable times with the right sequence. These actions are represented by an agent's task structures. Task structures might be created in agent architectures by various means: table lookup, reactive planning, task reduction, classical HTN planning, etc. While the TÆMS representation is more completely and formally specified elsewhere (e.g [21, 6]), the basic idea is as follows. A *task* represents a way of achieving some objective, possibly via the execution of some set of subtasks. Each task is related to a set of subtasks via a Quality Accumulation Function (QAF) that indicates how

the execution of the subtasks affects the completion of the super task. In this paper we will stick only to AND/OR trees: an AND task indicates a set of subtasks that must be accomplished, and an OR task indicates a set of possible alternatives to be considered. Eventually these task trees end in leaf nodes that represent executable methods or *actions*. Actions are described by a vector of characteristics (e.g. quality, cost, and duration) that allow predictive scheduling based on some dynamic agent utility function (i.e. maximize quality subject to meeting a deadline). The result is a well-defined calculus that allows the calculation of the utility of any particular schedule of actions.

The agent scheduler can be implemented with any number of algorithms (cf. comparisons in [11]). However, because of the inevitability of non-local dependencies, and the associated uncertainty of the action characteristics, the ability of the scheduler is severely limited if it can not acquire information about when and how the non-local dependencies are to be handled. It is this information that is provided by coordination mechanisms.

If the execution of a task affects, or is affected by, another task, we say there exists relationship between these tasks. These relationships among agent tasks can be classified as hard or soft relationships. For example, if action Act1 cannot start until action Act2 is finished, we say Act2 *enables* Act1, and this is a hard relationship. On the other hand, if Act1 and Act2 can be executed in either order, but doing Act2 before Act1 is "better" for Act1 (it acquires a lower cost, higher quality, faster duration), then we say Act2 *facilitates* Act1, and that this is a soft relationship. In any case, if a task structure (or some alternative within) extends over tasks/actions located at different agents, then these *non-local* relationships (subtask, enablement, facilitation, etc.) that cross agent boundaries become potential coordination points. We will concentrate on the enablement relationship as our main concern in this paper, although these mechanisms presented here will also work for facilitation.

Previous work on GPGP coordination mechanisms [5] had described them in an abstract way, which made implementation and analysis difficult. Our more recent observation is that we can specify a specific coordination mechanism generally as a set of protocols (task structures) specific to the mechanism, and a pattern-directed re-writing of the HTN. For example, if Act2 at Agent 2 enables Act1 at Agent 1, then one coordination mechanism (out of many) might be for Agent 1 to ask Agent 2 to do Act2, and to commit ahead of time to a deadline by which Act2 will be completed. Here the protocols are a reservation and a deadline commitment protocol, and the re-writing changes "Act2 enables Act1" into "reserve-act enables deadline-cmt enables Act2 enables Act1". To support this activity, an agent architecture must provide a facility for examining patterns of relationships in the current task structures between local tasks and *non-local* tasks, and re-writing them as required by the mechanism. This approach enables the cataloging of potential coordination mechanisms for a relationship, much more clear comparisons, and the real possibility of supporting automated coordination in an agent architecture such as DECAF (leaving aside for the moment the important question of *which* coordination mechanism to use for any particular relation-

ship and context)

3. COORDINATION MECHANISMS FOR ENABLING TASK RELATIONSHIPS

We have catalogued at least seventeen coordination mechanisms for enable relationships. For example, if a task TB at agent B enables task TA at agent A, one could:

- Have A request that B complete TA by some deadline;
- Have B commit to a deadline of its own choosing for TB (the original PGP-inspired mechanism [5]);
- Have B send the result of TB (“out of the blue”, as it were) to A when available;
- Have A poll for the completion of TB (Our model of current hospital practice [6]);

The seventeen mechanisms are not an exhaustive list, and many are simply variations on a theme. They include avoidance (with or without some sacrifice), reservation schemes, simple predecessor-side commitments (to do a task sometime, to do it by a deadline, to an earliest-start-time, to notify or send result directly when complete), simple successor-side commitments (to do a task with or without a specific EST), polling approaches (busy querying, timetabling, or constant headway), shifting task dependencies by learning or mobile code (promotion or demotion), various third-party mechanisms, or more complex multi-stage negotiation strategies. This paper will focus on both the absence of coordination, and seven implemented mechanisms of various types.

In order that these mechanisms be applicable across task structures from any domain, the result of the mechanism is some alteration of the task specification. This might be a true structural alteration (i.e. removing or adding tasks) or an alteration of the annotations on the task structure. As an example of the latter, consider the scheduling problem imposed by a task structure that includes a non-local task. In general the local agent may have no knowledge about the characteristics of that non-local task. Thus even though the agent may have perfect knowledge about all of its local tasks, it cannot know the combined characteristics of the complete task structure. Coordination mechanisms that rely on commitments from other agents remove this uncertainty and allow the local agent to make better scheduling decisions.

3.1 Avoidable Dependency

Even the simplest coordination mechanisms take some time and effort. In the case where alternatives are available (an OR task node) then one way to deal with the dependency is to remove it from consideration (all other things being equal). Only if the non-dependent alternative were to fail would the agent attempt the dependent alternative.

As an example, two agents A and B have task structures as in figure 1. Since agent A has two alternatives for completing TaskA, and one of them does not involve the dependency, one response would be to prune the dependent alternative

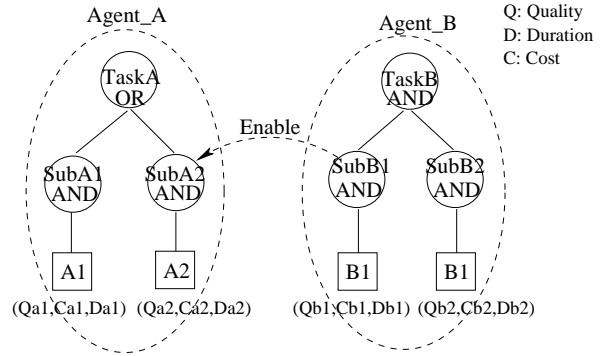


Figure 1: Avoidable Dependency.

SubA2, assuming that the local utility of completing alternative SubA1 is no less than that of completing SubA2. This utility will be some function of the various characteristics (here: quality, cost, and duration) that accompany each alternative.

Although this seems like a trivial mechanism that should be used whenever possible, remember that the decision is still made from a local perspective. One can construct environments where this behavior is not in the best interest of the agents, for example, if the agents are quality maximizers only, and SubA1 accesses a shared resource¹ that produces poor quality when over-utilized (i.e. the “Tragedy of the Commons”).

3.2 Sacrifice Avoidable Dependency

Simply choosing task SubA1 to avoid the hard dependency could result in a somewhat lower utility because this alternative has a different mix of performance characteristics. For example, Agent A may desire a pizza, and may have all the ingredients and capability to make one itself, or it could call for take-out. The first alternative may (for good cooks) result in a higher-quality result at lower cost, but take considerably more time than the second alternative. A subtle variation on the avoidable-dependency mechanism is then to avoid the dependency even if it means sacrificing perceived local utility. Although we focus on enablement in this paper, note that this mechanism may also be used for facilitation (ignoring the relationship).

The mechanisms for handling Avoidable/Sacrifice-Avoidable dependencies are very fast. They have much shorter coordination time compared with more complex mechanisms, and are of course easy to implement.

3.3 Coordination by Reservation

This mechanism is named after the real world activity of reservations. For example, if you want to have dinner in a restaurant, before you go there you’d better make a reservation so that at some agreed future time, you will be there and the people there will be ready to serve you. This mechanism includes both a rewriting of the local task structure and some external communication protocols.

Imagine Figure 1 with task SubA1 is removed (so Agent A

¹TÆMS represents such shared resources explicitly[6]

has no alternative). The reservation mechanism includes a new protocol, instantiated as a new task structure (called “Wait”) at Agent B, that processes a message indicating a request to do a task sometime in the future. The result is a message indicating when (if ever) the task will occur. The reservation mechanism rewrites the task structure at Agent A so that a new subtask (“GPGPReservation”) is executed before SubA2 and invokes the “Wait” protocol at Agent B and then processes the return message. The result is an annotation on the non-local task that allows Agent A to predictively schedule SubA2 locally at a time after the enabling task has been completed.

Coordination by Reservation works very well in complex task structures with many task levels, although it requires extra communication overhead. It is advantageous in exploiting the best schedule, with the lowest rate of deadlines missed and the highest quality achieved.

3.4 Demotion Shift Dependency

Under the same assumption of removing SubA1 in figure 1, we name AgentB as Predecessor and AgentA as Successor. *Demotion Shift* is so named because the task structure is transferred to the successor from the predecessor.

The communication protocol is: AgentA detects the dependency relationship with AgentB’s task, SubB1; AgentA sends a request coordination message to AgentB for task demotion. AgentB receives the message and sends back the task information and the object code for SubB1; AgentA gets the returning message and dispatches it to GPGPModule again; The GPGP module in AgentA unwraps the message and executes the object code to get the result, at the same time modifying the task structure so that the result is directed to the next execution module.

Demotion Shift Dependency performs efficiently when the same tasks are requested for execution over and over again. For example, in information gathering applications, very complex wrapper code can be demoted to the requester, especially when the wrapper is unpredictably loaded. Although we do not implement it in this paper, *Promotion Shift Dependency* is also possible in information gathering applications to upload tasks from a handheld agent or thin client to a large server.

3.5 Coordination by Sending Result

Instead of sending coordination meta-information back and forth, the enabler sends the actual execution result to the enablee. This mechanism differs from previous mechanisms in that the coordinated agents do not have to spend extra time on re-scheduling and exchanging coordination information. On the other hand, the enablee cannot do any detailed predictive scheduling, since they only have a commitment that it will be done, not *when*.

This mechanism is often used for simple load-balancing purposes. For example, in figure 1 assume that Agent A chooses to do SubA2 because it represents the offloading of almost all the processing to Agent B. This mechanism is the default in DECAF.

3.6 Coordination by Polling Result

Polling has been widely used in many areas such as network management, remote network monitoring, etc., to attack the problem of unreliable communication protocols. The enablee agent, AgentA, asks for the result of SubB1 with polling. The enabler agent, AgentB, replies with the result only if the requested task is already completed. Here, *Polling* means AgentA starts a new query to AgentB at some fixed periodic time interval. For example, AgentA continuously asks the result from AgentB every ten seconds until the result is received. During the time between queries, AgentA is flexible to execute whatever other tasks it has so that it is not idle.

Polling is good for communication protocols that are connectionless and have no guarantee of returning service. On the other hand, communication costs are clearly higher because of the continuously periodic queries before acquiring results.

3.7 Constant Headway / Timetabling

In certain situations AgentA may know or detect that AgentB executes routine tasks periodically in some fixed manner and the repeated tasks provide inputs to some task of AgentA. Similarly, the enabling task may be executed according to some stable timetable. In either case AgentA can deduce either a maximum bound on task completion (constant headway) or a precise schedule (timetabling) without explicit coordination.

We can think about this mechanism as how a subway train or bus system works. Subway trains are scheduled to arrive at some known periodicity (constant headway), buses arrive according to a timetable. In this situation, the enablee, which could be the traveler, acquires the bus schedule in advance by picking up a bus-schedule timetable. At scheduled time, traveler comes to the bus stop right before the arrival of the bus and waits to be picked up.

One problem is still present, which is how to determine the number of services in the period between each enabler’s execution. Continuing the previous example, the buffer could be the the number of empty seats, which is a constraint on carrying travelers. It is important to balance the buffer size so that the efficiency, correctness and completion of executed tasks are achieved.

4. IMPLEMENTATION

The execution of coordination mechanisms can be activated either under programmer direction or left up to the agent’s coordination component² The mechanisms can be selected in any combination. Although the ordering of the execution of mechanisms matters in terms of time/performance efficiency, the functionality is ensured correct.

The detection of the coordination relationships is domain-independent, which is advantageous compared to the earlier approach taken in [5]. The structure of DECAF tasks explicitly reveals abstract dependency information. By parsing the KQML/FIPA messages during program execution,

²This will be useful for eventual learning and exploration control, see section 6.

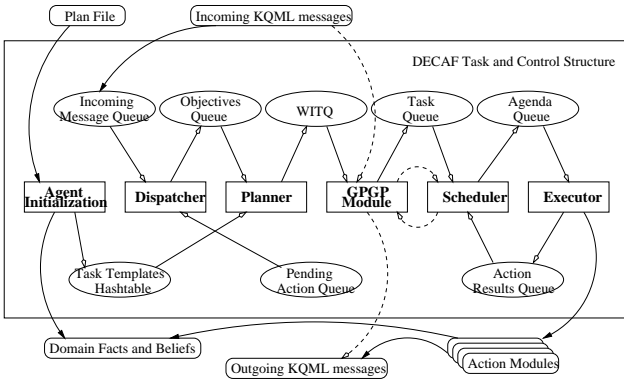


Figure 2: DECAF architecture.

even the specific dependency related agents are known to the GPGP component.

Previously, a single agent in DECAF system has message dispatcher, planner, scheduler, Agenda Manager and executor. These components work together to keep track of an agent's current status, plan selection, scheduling and execution. Now we put a GPGP component between the planner and scheduler (Figure 2).

GPGP analyzes the structures of the tasks in the WITQ ("What-If Task Queue"). According to specific characteristics of the task structure, GPGP exploits some coordination mechanism. Then, the modified tasks will be sent into Task Queue, from which the local scheduler chooses tasks for efficient scheduling. The dotted lines between GPGP Module and Scheduler indicates that GPGP takes advantage of the local scheduler's scheduling ability to evaluate the features of actions for a remote agent (for example, when making a commitment for either a predecessor task or a reservation request). The dotted lines, from Incoming KQML Message Queue to GPGP module, and the one from GPGP module to Outgoing KQML Message Queue, show that GPGP takes request from remote agents to do task evaluation work, and then sends the evaluation result back to the requesting agents.

The DECAF task structure representation is composed of Task, Action, Non-Local Task (NLT—explicitly representing an inter-agent dependency), Provision Cell, Characteristic Accumulation Function (CAF) and Action Behavior Profiles. It is a fairly straightforward transformation from the abstract TÆMS task structures (Figure 1) to the actual DECAF task structures (Figure 3) In DECAF (based on RETSINA), we make the abstract structures concrete by explicitly indicating the provisions (inputs) needed before a task runs, and the possible outcomes that can occur. A *Provision Cell* is a data structure which stores the required inputs before action can be executed, or the outcomes of the action execution. The line between them, such as the outcome provision cell OK from Ask and the input provision cell IN of the NLT (Non-Local Task), means the value of the outcome of action Ask is transported to the input of NLT by a KQML message, so that the NLT can be instantiated. The NLT can not begin execution until the input value is

filled by an incoming message. In this way, the relationships among tasks can be represented naturally with DECAF task structure, and the data flows allow an agent to intelligently use local resources such as multiple processors.

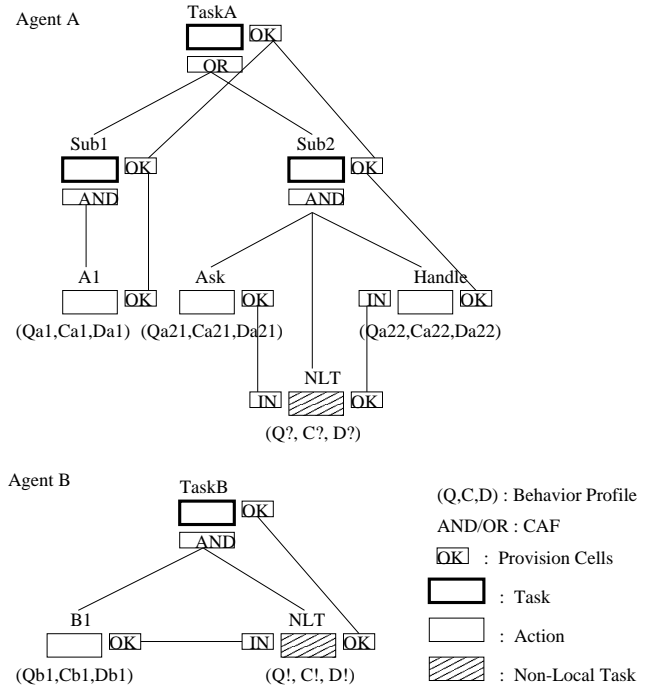


Figure 3: Actual DECAF agent Task Structure.

5. EXPERIMENTAL RESULTS

5.1 Experimental Framework

We created an experiment framework to test the performance of these mechanisms under various environment factors. Since our coordination mechanisms focus on the non-local dependencies, we reduce the complexity by using only two agents, and concentrate on the relationships between their non-local tasks. Both agents have an average of nine multiple subtasks with alternatives (OR-nodes) per request. They may choose any selected subset of the task alternatives to be executed based on various environment factors.

This experiment framework is constructed as a simulation testbed for coordination mechanisms. It can be also applied to real applications, for instance, query planning and execution in an information gathering system. For information gathering tasks, a query planning agent takes user input and decomposes it into sub-tasks distributed to multiple wrapper agents. Each wrapper agent queries remote databases, extracts the expected information, and sends the sub-results back to an agent for result composition. One wrapper agent may need the results of other wrapper agents' queries for either further sub-queries or further information extractions. The scheduling for a wrapper agent to access multiple information resources, the explicit enablement relationships between the reasoner and the wrapper agent, and the interdependencies among wrapper agents, provide coordination mechanisms with a perfect stage to perform on.

The abstract experiment structure is similar to figure 1, except that the task structures and task characteristics are selected randomly during experimental execution. In this framework *Error Rate*, *Task Repeat Rate*, and *Agent Load* are some of the changing input factors that describe an environment. Multi-agent system performance is characterized by *Quality Change*, *Communication Load*, *Task Execution Time*, *Idle Time*, *Deadlines Missed* and *GPGP Coordination Time*. In this paper we concentrate on the analysis of execution time and coordination time under various error rates and task repeat rates.

5.2 Experiment Result Analysis

Each experiment with respect to a different coordination mechanism was repeated 20 times for each environmental condition, and we report the mean performance scores. We first show the change in final quality and the communication load qualitatively.

Env.	Mechanisms						
	A	SA	RES	D	SR	P	CH
QC	≥ 0	Dep	> 0	> 0	> 0	> 0	> 0
CL/CT	0	0	Low	Dep	Low	High	0

Mechanism performance on different environments.

As shown in the table, the characters have the following meaning: *A* = Avoidable Dependency; *SA* = Sacrifice Avoidable Dependency; *RES* = Coordination by REServation; *D* = Demotion Shift Dependency; *SR* = Coordination by Sending Result; *P* = Polling; *CH* = Constant Headway; *QC* = Quality Change; *CL* = Communication Load; *CT* = GPGP Coordination Time.

Since quality is domain-dependent, it is reasonable to use qualitative expressions to show the result of Quality Change with Positive, Non-Negative or Depends. The Sacrifice Avoidable mechanism may sacrifice quality achievement but ensures faster execution, and it is thus suitable to be used in time constrained environments. Other mechanisms usually achieve higher quality but have longer coordination time. Because Avoidable and Sacrifice Avoidable mechanisms just alter the task structures without any non-local communication, their CL is 0. The period/headway is the key information for Constant Headway; if this information is available, agents do not need coordination communication if there is sufficient service available. Reservation and Sending-Result are similar, differing in whether coordination meta-information (the reservation) or the actual domain result is sent back, approximately two non-local communications. Polling obviously has the highest communication load value since it issues new queries periodically, but it provides better reliability and connectionless communication. We list communication load and coordination time together because coordination time is mainly composed of coordination communication time, which usually takes seconds (more for true Internet connections) while the GPGP reasoning module only uses less than dozens of milliseconds.

As the above table shows, the communication among the coordinating agents is not necessary for these three mech-

anisms: *Avoidable Dependency*, *Sacrifice Avoidable Dependency* and *Constant Headway / Timetabling*. While we are more interested in the mechanisms which agent communication is required. As a result, we choose the other four mechanisms for further analysis: Reservation, Demotion, Send-Result and Polling. Figure 4 shows how the average task execution time changes according to the coordination task repeat rate. *Task repeat rate* characterizes how often an enabler task is requested for execution by the enablee. As expected, demotion shift dependency outperforms the others if the coordination task repeat rate is higher than 0.4, or costs extra time when below 0.2. Coordination by reservation and coordination by sending result share similar performance values here. The frequency of each request for Polling is selected appropriately so as to not degrade the agent's performance by polling too often. However, the overhead of the polling requests still creates a longer execution time than the reservation and result sending mechanisms.

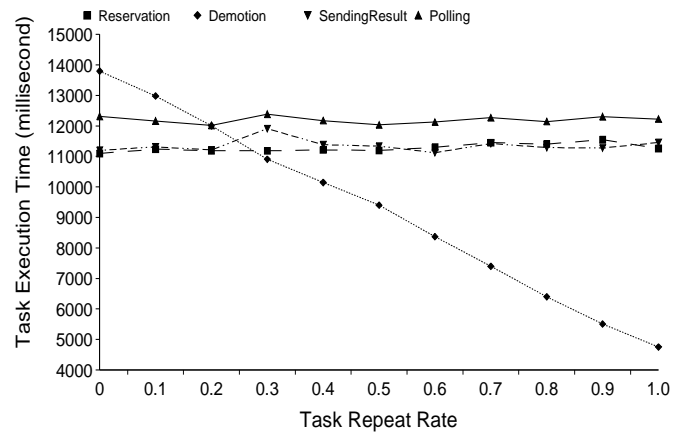


Figure 4: Performances of coordination mechanisms.

Next graph Figure 5 shows the performance of a particular mechanism, Polling, based on various poll gaps. We define *Poll Gap* as the length of the time period between each polling request. Poll gap reflects poll frequency, since a small gap results in higher frequency and a larger gap stands for a lower frequency. As Figure 5 indicates, the dashed line is for the situation that the task is finished already before any further poll request, that is, the poll gap is longer than the task execution time. The graph points out the overhead cost associated with too-frequent polling.

Polling is more useful as a mechanism when error rates increase. As in Figure 6 shows, we see that Polling mechanism finishes the tasks much faster than other mechanisms as the error rate gets higher. Here, “error rate” could be either task or communication error (in the experiments, it is simulated task error rate). This is because the Polling mechanism keeps requesting for results leading to much earlier re-execution of the failed task, while the other mechanisms have to wait for a reply until they timeout. As the error rate rises, the other mechanisms’ performance gets worse much faster, while Polling only degrades in an approximately linear way.

From Figure 5 and Figure 6, we conclude that Polling outperforms other mechanisms if system is unreliable, but costs much more in communication overhead. If one carefully chooses the poll gap based on different error rates, one can optimize the performance for Polling.

Different coordination mechanisms are better in different environments and there is no single best mechanism for every situation. To carefully choose a coordination mechanism based on different environment conditions improves multi-agent system performance.

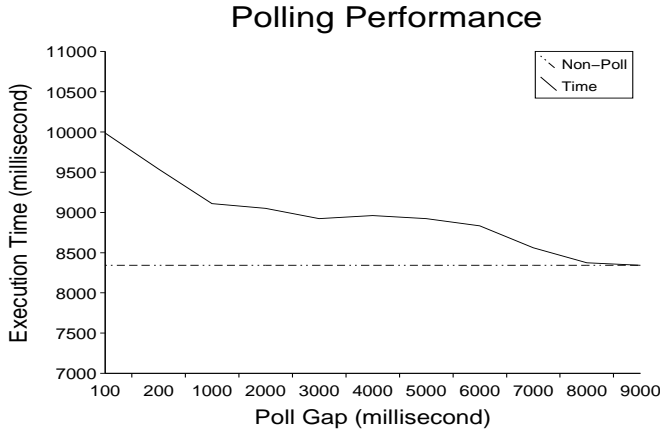


Figure 5: Performances of coordination mechanisms.

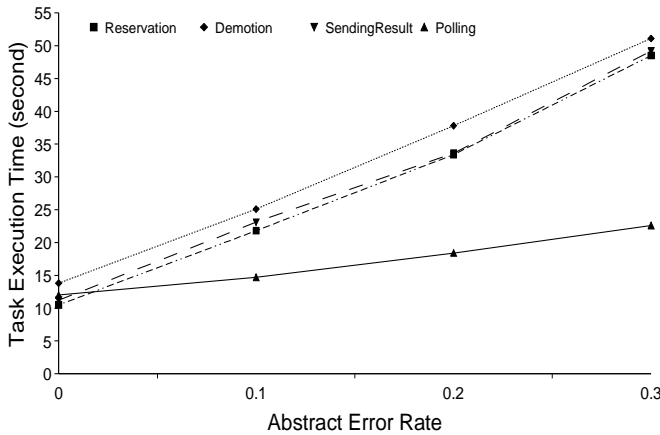


Figure 6: Performances of coordination mechanisms.

6. APPLYING THE MECHANISMS UNDER VARIOUS ENVIRONMENTS

Our first goal is to associate a set of possible coordination mechanisms with every agent. However, as we have discussed, various application environments have different features that directly affect agents' coordination behaviors. Our research objective is to enable our agents to autonomously select appropriate mechanism(s) according to various factors in different environments. Next we will introduce our approach about how to model various application domains,

and our plans to modify a boosting algorithm so that agents may adapt themselves for different environments.

There are some previous learning approaches in multi-agent systems [14, 18, 9]. But these approaches are domain-specific and treat various environmental factors as being static. As a result, they are not appropriate for our objective, which is to study different problem spaces to understand the relationship between problem domains and coordination mechanisms. [13] is an initial approach that respects the domain knowledge and develops *situation-specific coordination*. But that approach still sticks to a specific domain without presenting a general method for our objective. Here we present a general method to evaluate the domain's effect upon coordination. We have stated that there is no single best coordination mechanism for all environments, but at the same time coordination behaviors do share common features. Based on the relationships between general and domain knowledge, we claim that *it is necessary to separate general coordination knowledge and domain knowledge to model the coordination behaviors and the various environments*.

General knowledge describes the basic facts possibly impinging on coordination in agent systems. These facts are natural attributes about the system itself. For example: the number of agents in a multi-agent system, whether communication between certain agents is available or not, and if communication is available what is the communication load and communication frequency, agent failure rate, coordination task repeat rate, communication bandwidth, average message size, etc. These factors are basic for every kind of multi-agent system and they reflect general features that may impact the choice of mechanisms. We represent the general coordination features with a vector $\mathcal{V}_g = \langle g_1, g_2, \dots \rangle$. Element g_i represents one general feature of the coordination system, and vector \mathcal{V}_g is the collection of all general environmental features.

Domain-dependent knowledge represents the featured environmental factors in different application areas. Coordination mechanisms selected vary from different environments. For example, in Bioinformatics the most heavy calculations and time consumed are at remote gene/protein databases or analysis tools [4]. But for agent system developers, remote databases are not under direct control. What we really care about are the factors like how our wrapper agents extract information from the remote databases, what the size of each query/result message is, and how these wrapper agents coordinate their query tasks so that the system provides fast and expected query results. The domain factors in a Bioinformatics system (some of which are common to most information gathering systems, but not to any arbitrary MAS) include the number of gene databases to query, number of external analysis tools, query repeat rate, average/maximum analysis pathways (measured in number of NLTs), query plan features, bandwidths of the communication channels, availability of local caching, response time of remote databases, etc. Since there is no database holding all the information biologists may need, it is inevitable to query multiple different databases such as GeneBank, SwissProt, etc. The number of databases to query does matter for the system, because querying more databases may create more hits and improve the query accuracy, but with longer query

time as a side effect. The bandwidth of a communication channel is also a domain-dependent factor, since in Bioinformatics the size of the query result message varies from several KB to more than 10 MB, and possibly even bigger. A higher bandwidth shortens the response time for the end users, the biologists.

We explain more about the domain-dependent factors by discussing the potential relationships between these factors and the coordination mechanisms. For example when doing a BLAST query against the NCBI database, after the query is submitted, a query ID is displayed for this specific query together with hint as “the results are estimated to be ready in 12 minutes ... “ Our mechanism of Predecessor Deadline Commitment fits this situation very well in that the long sequential query time is hidden by multiple queries at the same agent and the results are picked up at a fixed time in the near future. Another factor is repeated queries. Although biologists from different institutes/universities/organizations usually query absolutely different data against the databases, it is very likely that people (such as professors and their students) from the same place make the same queries. In this case Coordination by Demotion Shift Dependency (for query task structure) and Local Caching (for final result) will save much time for repeated queries. In Bioinformatic systems the accuracy and the number of hits, which relate to quality, are important, while for other areas the domain-dependent factors are different, such as EMS (Emergency Medical Service) systems. The main concern of an EMS is to deliver ambulances to emergency sites and take victims to the nearest hospitals as soon as possible. Time is the most important characteristic to evaluate in an EMS system. The domain-dependent factors in EMS are the number of ambulances and hospitals, the alternative routes to the emergency site and the hospitals (topology of the EMS geographical area), the time taken for the hospitals to get prepared, the traffic pattern of the area, etc. From the above examples we can see that the featured factors under different environments are different.

We represent the domain-dependent knowledge with another vector $\mathcal{V}_d = \langle d_{i,1}, d_{i,2}, \dots, d_{i,1}, \dots \rangle$. Element $d_{i,j}$ represents the featured factor j of the application domain i . Now the coordination universe is represented with vector

$$\mathcal{V} = \langle \mathcal{V}_g, \mathcal{V}_d \rangle.$$

We explicitly differentiate these in order to clarify that *general knowledge and domain knowledge could be studied separately*. While there are virtually unlimited factors that may be used to describe even a single application area, we select in consultation with domain experts the main features and rule out the trivial/irrelevant features, such as the salaries of the people who use the query system.

After we vectorize the coordination universe, we need to normalize the vector so that the elements can be formatted as input for learning algorithms. It is easy to normalize the elements based on their minimum and maximum instance values. For example, the bandwidth of the Internet Information Gathering system ranges from zero to 10M bps, an instance of 3M bps could be normalized as the value of 0.3. We represent coordination mechanisms with vector $\mathcal{M} = \langle m_i \mid i = 1..17 \text{ mechanisms} \rangle$. Now our problem is

reduced to as follows: Given an input vector \mathcal{V} and an output vector of \mathcal{M} , how can we find a hypothesis that predicts which m_i should be selected according to the selected \mathcal{V} for the best system performance. We apply the mechanisms one at a time under certain combinations of environmental values in the coordination framework, and then modify the environmental values and apply the mechanisms one at a time again. We repeat this process until we acquire a set of mappings π which selects which mechanism to use under which combination of environmental values. We take this set of mappings as a training set. Finally the problem reduced to as follows:

Given (v, m) , where $v = \text{one instance of } \mathcal{V}$ and $m = \text{one instance of } \mathcal{M}$, find a hypothesis \mathcal{H} to predict the relationship between the input and output.

The abstract problem described above represents the concrete real world problem, which is how to map out particular application domains and selecting appropriate coordination mechanisms for dealing with the optimization problems present in the domains. We are currently adapting the AdaBoost [15] algorithm for this abstract problem. The algorithm is shown as below:

*Given : $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \dots, T$:*

- *Train weak learner using distribution D_t ;*
- *Get weak hypothesis $h_t : X \rightarrow \{-1, +1\}$ with error $\epsilon_t = P_{r_i \sim D_t} [h_t(x_i) \neq y_i]$;*
- *Choose $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$;*
- *Update*

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right).$$

Boosting refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb in an accumulative manner. It is a general method to improve the accuracy of given learning algorithm. AdaBoost is advantageous because of these features: It is easy to implement; It only has one parameter, the round T , to tune; It requires no prior knowledge about any component weak learner and can be combined with any method for finding weak hypotheses; It was shown to achieve accurate prediction from accumulative weak learners. Boosting technology is helpful for our objective. Using domain-dependent elements from instances

of \mathcal{V}_d as input, we can find the hard domain-dependent factors and pay special attention to them, so that appropriate coordination mechanism is selected. We can also include the general factors as input to the algorithm. In this case, comprehensive knowledge is taken into consideration and some general factors may be found particularly important for the application domain as well.

There are still some points that need to be clarified when applying the above procedure to study the relationships between coordination behaviors and domain applications. First, as the above AdaBoost algorithm shows, it is a *binary* case, which means the label Y 's value is either true or false, while the label Y for our approach represents one of seventeen coordination mechanisms. This problem can be handled by reducing the multiclass problem to a larger binary problem [22]. Second, the acquisition of the training data is key part of our approach. It requires a large set of data from the execution of the experimental coordination framework. The good point is that the system developer does not have to explore all the possibilities for all environments, but to concentrate on an individual application domain. Finally, the developer does need some domain knowledge from expert to start the learning approach.

The approach presented above attacks the problem of understanding how application domains and coordination/control techniques relate. Given the ideas above, research tasks only need to follow the steps and concentrate on different application domains. It is a logically correct and practical approach, and more experiments are needed.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

We have introduced a new way of thinking about coordination mechanisms—as re-writings to hierarchical task networks—that brings us closer to providing general coordination services for an agent architecture in a domain-independent manner. We also introduced several implemented coordination mechanisms and discussed the applicability of them to different environments. The mechanisms we implemented above are on a promising path to explore coordination for dependency relationships. The mechanisms implemented so far can be applied to agent programs written with DECAF for different domains. The agent programmer does not have to explicitly code for coordination (although without learning, the programmer does have to select which mechanisms to automatically instantiate for each NLT). We presented a general approach towards the understanding and automated learning of the relationships between application domains and agent coordination, however more experimental results and a larger scale experimental framework are needed and are under development.

Different coordination mechanism may outperform others within different kinds of environments. *Avoidable and Sacrifice Avoidable* mechanisms cost very little in terms of meta-level coordination time, so programmers are encouraged to apply them first unless they are not applicable as described earlier. The more complex coordination mechanisms result in higher quality, but require longer times. If deadlines are the main factor agents must respect, *Coordination by Reservation* outperforms others. If the repeat rate of a requested

task is high, the *Demotion Shift* mechanism saves the transmission time by sending object code for further reuse. *Promotion Shift* has this advantage as well and saves one round of coordination communication, but with the constraint of extra domain knowledge. If the enablee agent is heavily loaded, *Coordination by Sending Result* distributes the computing task to the enabler agent. If the underlying network protocol is connectionless, *Polling* ensures the reliability of coordination protocol with continuous queries for the earliest presence of a result, or takes recovering actions early in case of failure. By publishing a well known timetable, the mechanism of *Constant Headway* unburdens the enabler and enablee agents from tightly coupled coordination.

We identified agent-architectural properties that are necessary for our approach: a local scheduler or other way of reasoning about non-local commitments, and a way to generate such commitments by “what-if” questions or other means. Viewing the eventual end-product of coordination mechanisms as information for making scheduling decisions gives a consistent and general way of integrating many different coordination mechanisms in many styles. The most difficult aspect of implementation is in altering an agent architecture so that planned activities can be examined and possibly altered before they are executed.

7.2 Future Work

In our original blueprint there are at least seventeen mechanisms for the hard dependency relationships, some of them only subtly different from each other. Most of the mechanisms can be implemented with task structure re-writing directly. Some of them may require arbitrarily complex new tasks, such as the Multistage Negotiation Mechanisms, but they can be implemented through DECAF anyway. There are some other mechanisms which require other technologies, such as mobile agent implementations. With the introduction of other technologies, coordination mechanisms will become more complex and more functional as well.

We explored the hard relationship, *enables*, among agents. The soft relationships are a simple extension to our current research. A more complex extension is to complex subtask alternative relationships across multiple agents. We have already begun representing soft relationships within DECAF. We believe all kind of non-local relationships can be explored within the general DECAF HTN task structures.

Current research concentrates on the analysis for individual coordination mechanism, but under certain environments it is highly possible that combinations of mechanisms may result in better performance. For example, Polling and dynamic event-driven mechanisms can be combined so that the new mechanisms not only ensure reliability requirements, but also provide a flexible solution for time constrained systems.

Error tolerance is another important issue associated with the mechanisms. Because the coordination mechanisms here are in execution together with a real time scheduling component, it is more desirable to make the mechanisms dynamically selected according to the changes of the factors in the environment so that the MAS system reaches optimal overall performance over a period of time.

Finally, learning approaches will allow agent designers (in non-critical action fields, such as personal information gathering) to build agents that can experiment with various known coordination mechanisms to find the best ones to use (or avoid) in certain common situations. More experiments and large training data set are needed to feed in the learning algorithm we present in this paper.

8. REFERENCES

- [1] C.Castelfranchi and R.Conte. Distributed artificial intelligence and social science: Critical issues. In *Foundations of Distributed Artificial Intelligence, Chapter 20*, 1996.
- [2] C.Dellarocas and M.Klein. An experimental evaluation of domain-independent fault handling services in open multi-agent systems. In *Proceedings of ICMAS'00*, Boston, MA, USA, 2000.
- [3] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, Jan. 1983.
- [4] K. Decker, S. Khan, C. Schmidt, and D. Michaud. Extending a multi-agent system for genomic annotation. In M. Klusch and F. Zambonelli, editors, *Cooperative Information Agents IV*, pages 106–117. Springer-Verlag, 2001.
- [5] K. S. Decker and V. R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems(ICMAS-95)*, San Francisco, 1995.
- [6] K. S. Decker and J. Li. Coordinating mutually exclusive resources using gppp. *Autonomous Agents and Multi-Agent Systems*, 3, 2000.
- [7] K. S. Decker and K. Sycara. Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9(3):239–260, 1997.
- [8] K. Erol, D. Nau, J. Hendler, and R. Tsuneto. A critical look at critics in htn planning. In *Proceedings of the IJCAI95*, Montreal, Canada, Aug. 1995.
- [9] C. B. Excelente-Toledo and N. R. Jennings. Learning to select a coordination mechanism. In *Proceedings of AAMAS02*, 2002.
- [10] G.Weiss. *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*, pages 88–92. MIT Press, 1999.
- [11] J.Graham and K.Decker. Towards a distributed, environment-centered agent framework. In *Intelligent Agents IV, Agent Theories, Architectures, and Languages*. Springer-Verlag, 2000.
- [12] M.Tambe. Teamwork in real-world, dynamic environments. In *Proceedings of the International conference on multi-agent systems(ICMAS96)*, 1997.
- [13] M. N. Prasad and V. Lesser. Learning situation-specific coordination in generalized partial global planning. In *AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, Stanford, Mar. 1996.
- [14] M. S. S. Sen and J. Hale. Learning to coordinate without sharing information. In *Proceedings of AAAI'94*, Seattle, WA, USA, July 1994.
- [15] R. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.
- [16] S.Rustogi and M.Singh. Be patient and tolerate imprecision: How autonomous agents can coordinate effectively. In *Proceedings of IJCAI'99*, Stockholm, Sweden, Aug. 1999.
- [17] S. S.Sen and N. Arora. Effect of local information on group behavior. In *Proceedings of the International Conference on MAS*, pages 315–321, 1996.
- [18] T. Sugawara and V. R. Lesser. On-line learning of coordination plans. Computer Science Technical Report 93–27, University of Massachusetts, 1993.
- [19] S.Willmott and B.Faltings. The benefits of environment adaptive organizations for agent coordination and network routing problems. In *Proceedings of IJCAI'99*, Stockholm, Sweden, Aug. 1999.
- [20] T.Hogg and B.Huberman. Controlling chaos in distributed systems. In *IEEE Transactions on Systems, Man, and Cybernetics*, pages 1325–1332, 1991.
- [21] T. Wagner, A. Garvey, and V. Lesser. Complex goal criteria and its application in design-to-criteria scheduling. Providence, July 1997.
- [22] Y.Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. 55(1):119–139, 1997.
- [23] Y.So and E.Durfee. Designing organizations for computational agent. In K. C. In M.J. Pritula and L. Gasser, editors, *Simulating Organizations*, pages 47–64. AAAI press, 1998.