GPGP – A Domain-Independent Implementation

John Phelps and Jeff Rye
Honeywell Laboratories
3660 Technology Drive
Minneapolis, MN 55418
john.phelps@honeywell.com

Abstract

This paper describes a new, domain-independent design and implementation of TAEMS-based Java Agent Framework/Generalized Partial Global Planning agents that targets the DARPA Coordinators program challenge problem environment. It offers a description of the challenge problem environment, a mechanism utilized to provide abstract coordination problem contexts that avoids complete problem centralization, generalized coordination protocols, and some preliminary performance results.

Introduction

We use the term Generalized Partial Global Planning (GPGP) to describe a family of coordination theories and implementations that share a common underlying problem representation called Task Analysis Environmental Modeling and Simulation (TAEMS). Various implementations of GPGP have been cited including the original DVM application(Decker 1995), DECAF(Graham, Decker, & Mersic 2003), and various efforts at Honeywell(Wagner *et al.* 2004; Wagner, Guralnik, & Phelps 2003b; 2003a). In our work at Honeywell, GPGP has come to mean the coordination part of a TAEMS-based agent. More specifically, it is the coordination mechanism of such an agent that drives the Design-to-Criteria (DTC) TAEMS scheduler.[1]

We describe a new implementation of the Honeywell GPGP that is based on previous Honeywell versions of GPGP. This latest incarnation targets a class of synthetic, domain-independent challenge problems generated as part for the DARPA Coordinators program (Wagner 2005). The Coordinators program's goal is the construction of sophisticated cognitive agents capable of assisting large deployed forces adapt their mission plans and schedules in real time. To accomplish this while respecting the authority of human commanders and the inherent complexity of the problems, we are designing our agents to implement several functions, including task analysis (singe-agent planning and scheduling), coordination, autonomy control, organizational decision making, and metacontrol. In the agent architecture discussed herein, DTC addresses the task analysis requirements, the GPGP component addresses the requirements of coordination between agents. And, although autonomy control and organizational decision making are an integral part of the overall program vision, we did not address them in this program phase or, hence, our attendant solution. Our metacontroller was developed by a team at the University of North Carolina, Charlotte and is contained in this symposium's report (Raja, Alexander, & Mappillai 2006).

The agent architecture and DTC+GPGP combination described in this paper was one of several approaches investigated by the Honeywell team to solve the Coordinators challenge problems. Building on previous versions of GPGP was a natural choice because Coordinators problems are encoded in a variant of TAEMS called Coordinators TAEMS (CTAEMS). Our hope was to leverage existing agent infrastructure and techniques developed Honeywell and the UMass Multiagent Systems Lab.

Whereas previous Honeywell implementation of GPGP were coupled through explicit task model constructs to specific domains, the version of GPGP implemented for Coordinators moves much closer to true domain independence (vis-a-vis TAEMS). This required creating a more general problem context management system than we had built before, disentangling the previous GPGP distributed search protocols from their reliance on domain-specific hooks, and developing a mechanism for maintaining real-time responsiveness while relying on the soft real-time DTC scheduler.

The Coordinators program stipulated that our solutions could not completely centralize the problems we were given to solve. One approach we developed that avoided complete centralization but that also provided some quantitative problem-solving context is a proxy method mechanism. *Proxy methods* enable one agent to inform another agent about the gross expected performance characteristics of an potentially complex task structure via an approximation. The approximation is stored in the performance characteristics of a single method that is denoted "nonlocal", meaning that the agent cannot itself execute it. Proxy methods are thus used to provide problem solving context for each agent's limited view of the overall problem by adding coarse-grained, quantitative approximations of its interactions with other agents.

To maintain real-time responsiveness while generating

[1] Although there might be a less contentious name for widely varying TAEMS-based agent coordination components, GPGP has stuck and so we will continue to co-opt it for this paper.

schedules with a soft real-time scheduler, we introduce a multiprocessing model that allows the computationally expensive Design-to-Criteria (DTC) (Wagner & Lesser 2001) scheduling performed at each agent to run outside of the coordination context processing and schedule execution.

In the remainder of the paper, we discuss the DARPA Coordinators program challendge problem environment, the overall architecture and basic control flow of our JAF/GPGP implementation, the proxy method technique, the approach to maintaining real-time responsiveness while using a soft real-time scheduler, the generalized coordination protocol, and give some exploratory data on the new implementation's performance relative to a centralized scheduler provided by the program evaluators.

TAEMS, CTAEMS, and The Coordinators "Cookbook"

The Task Analysis, Environment Modeling, and Simulation (TAEMS) (Decker 1995; Lesser, Horling, & et al.) language quantitatively describes the alternative ways a mission can be achieved. A TAEMS task structure is essentially an annotated task decomposition tree. Root-level tasks represent the goals or missions that an agent may try to achieve, which may be decomposed into several levels of sub-tasks, ultimately terminating at executable primitive tasks called methods. TAEMS methods include quantitative details to describe their expected quality, cost, and duration. Each of these dimensions is described with a probabilistic distribution. Together, the three-tuple of quality, cost, and duration represents a particular outcome. Methods may have multiple outcomes, each with an associated probability. By explicitly modeling this information, the TAEMS structure can capture the uncertain and dynamic character of real-world actions. TAEMS tasks have temporal constraints such as earliest start times and deadlines. TAEMS also supports quantitative relationships that span across the task hierarchy and capture other forms of task interaction. For example, enables and disables task inter-relationships (also called nonlocal effects or NLEs) are used to denote causality or precedence requirements. So-called soft NLEs such as facilitates indicate that one task can improve another, but is not required. Consumable and nonconsumable resources are used to represent the physical or logical components required by methods. Relationships define the effects actions have on resources, and the consequences that an absence or excess of resources may have. Relationships can exist within a single agent's task model, or span those of several agents. The Coordinators program created a variant of TAEMS called CTAEMS to represent its challenge problems. CTAEMS basically removes all explicit resource concepts from TAEMS and several QAFs (such as the sequence QAFs). It also added a QAF called a SyncSum which accumulates quality like Sum QAF for those child tasks that start at the same time as the first child task.

Further, a problem "cookbook" was created to prescribe the types of CTAEMS-encoded problems that would be seen in evaluation. The cookbook was then used by the program evaluation team to create a scenario generator that automated the otherwise impossibly tedious process of creating the problems by hand. Since we used problems generated by the scenario generator to test the solution reported in this paper it is worthwhile to discuss some of the specifics of the types of problems that the scenario generator generates.

The basic structure of a scenario is as follows. Each scenario is represented as a 5-to-6-level task structure with the following basic composition:

Scenario Level — A Sum QAF task node and the root of the task structure, a "task group" in TAEMS parlance.
Problem Level — Min or Sum QAF task nodes.
Window Level — Sum QAF task nodes and the locus of the basic temporal constraints on the problem in that tasks at this level specify the release times and deadlines for all of their child tasks.
CTask Level(s) — Max or SyncSum task nodes. When a SyncSum is introduced at this level, it's child tasks are CTasks, hence if there is a SyncSum task we have a 6-level task structure whereas if we don't, it's a 5-level structure.
Method Level — This is the lowest level, where methods represent the most basic actions the agent can execute.

This is shown graphically in Figure 1. The simplicity of the basic structure somewhat belies the complexity of problems that can be generated within it. Note that we are talking only about levels here, not numbers of nodes at each level. So, for instance, you might have a scenario with 4 problems, where each problem has 6 windows, each window has 5 CTasks, and each CTask has 4 methods. This would give a total of 4*6*5*4=480 methods. I don't mean to suggest that that was a typical problem, but it would not be out of scope, either.

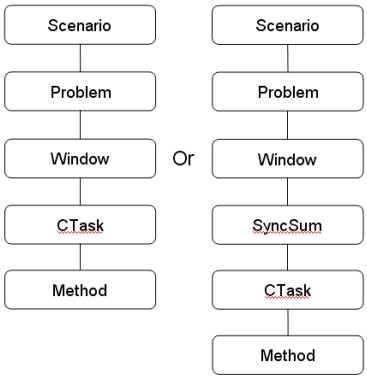

Figure 1: The basic 5-to-6 level Coordinators "Cookbook" structure for evaluation problems.

What further complicates the Coordinators evaluation problems is the addition of nonlocal effects (NLEs) within a single agent as well as between agents. NLEs come in too many esoteric flavors in the evaluation problems to discuss them at length here. The main concern for GPGP was the NLEs that spanned agents and evaluating their quantitative effects to properly evaluate potential commitments between agents.

Scenarios were tailored for distributed coordination evaluation via a procedure called the "visible-to" procedure that provided for each agent a subset of the initial task structure based on which methods the agent controlled. Each method generated in a scenario (problem) by the scenario generator would have exactly one agent that could execute it. Methods are not transferable between agents. We say that an agent "owns" a method that it alone can execute. According to the visible-to procedure, an agent would obtain from the simulation environment during an evaluation run a subset of the original scenario task structure based on the methods that it owned such that it would see all parent tasks of those methods. It would also see tasks visible to other agents based on their method ownership that are involved in inter-agent NLEs.

Architecture

Like previous Honeywell GPGPs, the Coordinators GPGP architecture is based on the Java Agent Framework (JAF) (Horling 1998). JAF provides three primary components that GPGP interacts with: the Design-to-Criteria scheduler (DTC), State, and the Executive. These are shown with their attendant interactions in Figure 2.

The Simulation Bridge is responsible for initially generating and then updating the agent's subjective view of the global task structure. Updates occur on events generated by the simulator that can change task deadlines or release times or add completely new tasks to the existing task structure. The current task structure is stored in the State component.

GPGP is responsible for creating coordinated schedules via exchange of information with other agents. Based on information from other agents in the form of commitments and proxy methods, GPGP calls DTC to analyze task structures and to create schedules which are deposited in the State component and run by the Executive.

The Executive dispatches actions dictated by the schedule and maintains consistency between the outcomes of the actions dispatched and the agent's view of the current state of the task structure. It can also perform some rudimentary rescheduling operations without invoking DTC, such as sliding the start times of a set of tasks forward in time if doing so does not violate scenario- or commitment-imposed constraints.

In conjunction with the MASS simulator, JAF was implemented to provide deterministic and repeatable results. In order to provide this guarantee, time was pulsed based on events in the simulation instead of based on wall time passing. Messages were passed synchronously. The Coordinators simulation changed both of these assumptions. Time in the Coordinators simulation is pulsed based on the passage of wall time and messages are passed asynchronously.

As a result of these simulation changes, JAF's architecture was adapted so that all processing of messages (whether from the simulator of from other agents) happens in a short period of time (apx. 1 second). Any processing that takes a longer must happen in a dedicated processing thread. Before the adaptation, JAF agents had a Control component which was responsible for pulsing the rest of the AgentComponents (Schedule, Execute, etc.). This Control component

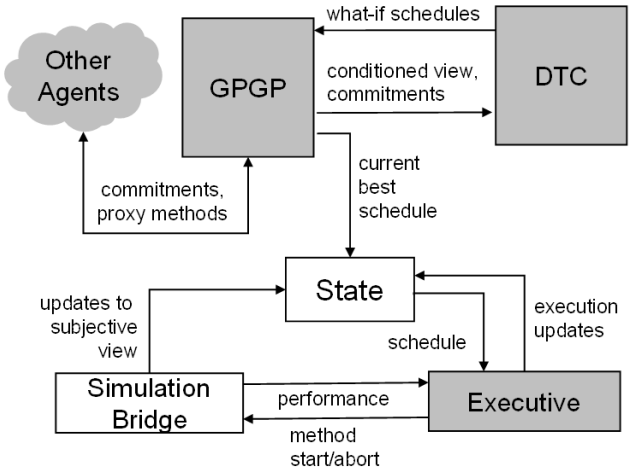

Figure 2: The Coordinators GPGP architecture.

has a thread which waited for a period of time before waking to see if it was time to execute a pulse. If it was time to do a pulse, Control would call the pulse method for each of its AgentComponents. When all of the AgentComponents had finished, Control would send a message to the simulator indicating that the agent was finished.

Managing Coordination Contexts with Proxy Methods

As mentioned previously, we assume that problem solving at a single agent begins with a subjective view of a task structure created from a scenario via the visible-to procedure outlined previously. Each agent's subjective view may thus not by default be lacking some quantitative performance characteristics of nonlocal tasks that enable or are enabled by local tasks. To provide an improved view each agent's interactions with other agents, without completely centralizing the problem, our implementation of GPGP introduces proxy methods as quantitative approximations of other agents's tasks performance. Once proxy methods are communicated to agents sharing an interaction, be it an NLE or shared task, they can be used to guide quantitative choices for commitments over the interaction.

Proxy methods are introduced on two types of interactions spanning agents: shared tasks and NLEs. For shared tasks, each agent in the shared task may have sibling subtasks of the shared task that are not visible to other agents. Each agent will communicate a proxy method for the sibling task of the shared task to the agents that it is not visible to.

Similarly, for NLE interactions, any agent that shares the source or target task communicates proxy methods that approximate the performance characteristics of the shared task's subtasks. The specific algorithm used to create the proxy methods at each agent is given below.

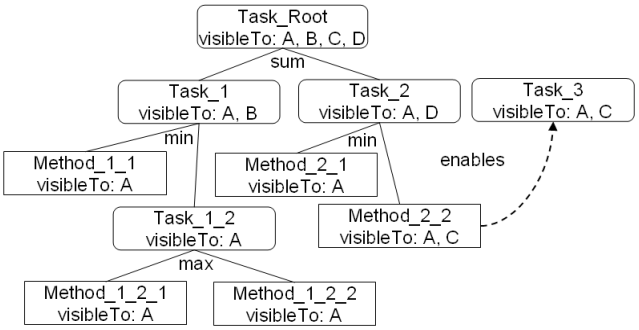

Figure 3: An example task structure with visibleTo information annotated.

Algorithm 1: CREATEPROXYMETHODS($task$)

for each $subtask \in$ SUBTASKS($parent$)
do $agentsToInform \leftarrow$ SETDIFF(
VISIBLETO($parent$),
VISIBLETO($subtask$));
SEND($agentsToInform$,
CREATEPROXYMETHOD($subtask$));
CREATEPROXYMETHODS($subtask$);

In Algorithm 1 we introduce a VISIBLETO function that returns the set of agents that a task in a subjective view is visible to according to the visible-to procedure outlined previously. CREATEPROXYMETHODS is initially called with the root of the subjective view of the task structure. It then performs a depth-first traversal of the task structure, sending proxy methods for tasks whose visibility decreases to those agents in the set difference (SETDIFF)of the task's and its parent's VISIBLETO sets. For example, consider the simple task structure in Figure 3, which shows the subjective view of Agent A.

Following the CREATEPROXYMETHODS algorithm, Agent A would generate and send the following proxy methods:

Task_1_Proxy — sent to Agents C and D, representing a performance estimate of Task_1.

Task_2_Proxy — sent to Agents B and C, representing a performance estimate of Task_2.

Creating the Approximation

A proxy method for a given task is generated by recursively joining the characteristic distributions of descendant methods according to the subtask structure's quality accumulation functions (QAFs). Each method's outcomes are first joined by applying the density of each outcome to its characteristic distributions (quality and duration distributions) to create one "global outcome". The global outcomes are then joined up the task structure according to the following heuristic formula for each QAF type:

Sum and SyncSum — return a new distribution for input distributions X, Y given by $\{(x + y, p_{X,Y}(X = x, Y = y))\}$.

Min — return a new distribution for input distributions X, Y first given by $\{(min(x, y), p_{X,Y}(X = x, Y = y))\}$ compacted.

Max — return a new distribution for input distributions X, Y first given by $\{(max(x, y), p_{X,Y}(X = x, Y = y))\}$ compacted.

$p_{X,Y}$ is the joint probability mass function (PMF) of the random variables X and Y. To make the process more concrete, let's consider an example with the following distributions, where each element of the distribution is given as a tuple $(value, P(value))$.

- $p_X = (10, 0.8), (8, 0.2)$
- $p_Y = (5, 0.6), (4, 0.4)$

marginal PMF X	joint PMF X, Y	
(10, 0.8)	(15, 0.48)	(14, 0.32)
(8, 0.2)	(13, 0.12)	(12, 0.8)
marginal PMF $Y \rightarrow$	(5, 0.6)	(4, 0.4)

Table 1: An example Sum/SyncSum QAF join.

To create the output distribution for a Sum QAF, we combine all pairs of the values of each of the input distributions to produce Table 1.

Each distribution resulting from a join of k other distributions (one for each subtask of a given task) feeds into the join at its supertask. For instance, for the Task_1_Proxy (see Figure 3 Method_1_2_1 and Method_1_2_2 would Max join at Task_1_2. Then Task_1_2 and Method_1_1 would Min join at Task_1.

Real-Time Responsiveness with Soft Real-Time Scheduling

Due JAF's origin as a client of the Multi-Agent Survivability Simulator (Vincent, Horling, & Lesser 2001), a discrete event simulator where each time unit has no relation to wall-clock time, JAF components share one thread and are given processing time in a pulse that is delivered to each component round-robin. The pulse for each component is part of a larger control pulse that originates from MASS, when JAF is connected to MASS, but that can also be generated internally. Each component pulse may take unbounded time and during one component's pulse no other component will be pulsed.

While this model of control is excellent for producing predictable behavior in distributed systems, it posed problems for meeting the real-time responsiveness challenges posed by Coordinators problems, which established a strict relationship between the units of duration in TAEMS methods, ticks, and wall-clock time. Specifically, a relationship specifying 1 duration unit = tick = 1 second was deemed acceptable by the program evaluation team.

The crux of the problem for us was that the GPGP component, which is responsible for creating coordinated schedules for the agent to execute, makes repeated calls to the

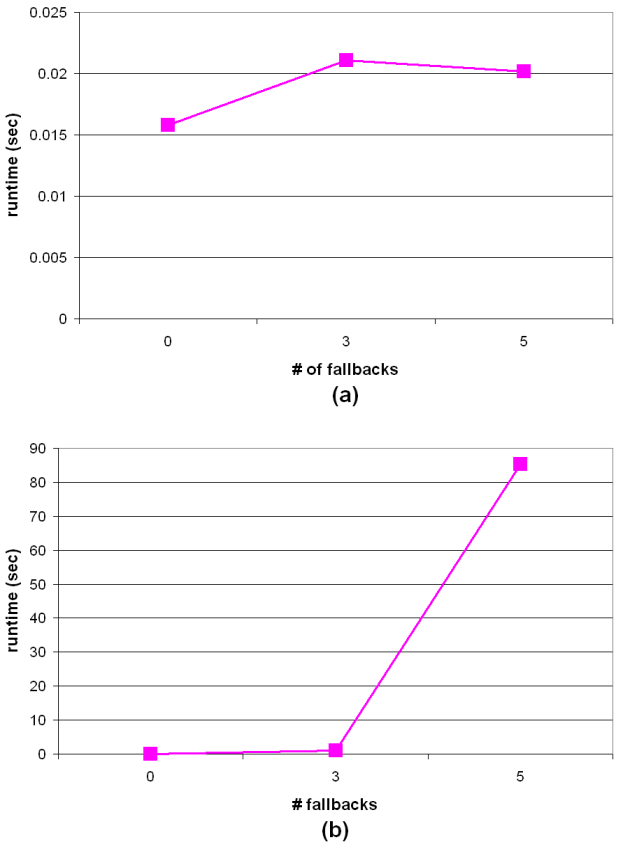

Figure 4: DTC runtime on (a) single-agent, 1 problem, 1 window scenarios with varying numbers of fallbacks and (b) the same except with 5 windows instead of 1 window.

DTC scheduler. This is a problem because the DTC scheduler was not designed to be used in hard real-time problems. It was designed to respond in soft real-time, where timely response is seen as a goal but not an inviolable requirement. So, by default, when GPGP made a call to DTC during its pulse call (when the GPGP component got access to the agent thread), all other components, including the component responsible for responding to executing a schedule and responding to events from the simulator, would block.

Although in our early investigations this performance issue was not prevalent, when we began to scale up the size of the problems we generated for GPGP+DTC we started to notice significant performance degradation. Figure 4 shows how DTC's response time increases with the number of fallbacks for a given CTask. Fallbacks are methods under a CTask owned by the same agent that are intended to be substituted in case the originally scheduled method fails. Fallbacks are generated by the scenario generator to take increasingly less time to execute but to also yield lower and lower expected quality. As problem sizes grew DTC had increasing numbers of alternatives to generate and consider, leading to longer runtimes.

Since, when they are made, the calls to DTC (during GPGP's pulse) consume nearly all the the time for a given round of pulses to all the JAF components, we decided to move the DTC scheduling to a separate thread. With that done, each pulsed component needed only to return quickly enough from the pulse call so that the sum of all the component pulse times was less than the duration of a tick.

With simulator tick durations equal to approximately 1 second, this required us to break up each major function that called DTC into pre- and post-processing that returned immediately rather than waiting for DTC to return. The preprocessing function creates the task structure for DTC to schedule and then forks a thread for the DTC processing. The processing thread assigned to DTC simply waits for DTC to return, reads in the schedules produced and set a global flag that indicates the scheduling run is done. The postprocessing function examines the schedules returned and takes whatever control actions are appropriate based on the new schedule.

In the following section, the protocols that we discuss show basically how this mechanism was used in our implementation.

GPGP Protocol

Previous Honeywell GPGP protocol implementations were tied to the domain they were targeted to, usually via hooks in the task structure created by the domain problem solver (Wagner *et al.* 2004; Wagner, Guralnik, & Phelps 2003b; 2003a). For the Coordinators implementation of GPGP, we were given an opportunity to create a version of GPGP that generalized previous versions.

For Coordinators problems, GPGP would not be given as explicit guidance about which tasks required coordination because each agent is simply given a subjective view of a shared global task structure with no annotations to guide its coordination search. Agents would thus have to first expand their local views of the problem to obtain a better understanding of their relationships with other agents. This is accomplished via the proxy methods mechanism just covered.

Once the proxy methods have been established for each agent for each potential interaction with other agents, each agent generates sets of commitment requests. This is done by first conditioning the agent's subjective view by assuming the best case for all interactions with other agents using the information established via proxy methods. The approximate quality and time performance characteristics of the proxy methods are used in conjunction with problem-dictated constraints, such as task release times and deadlines to generate sets of what-if commitments on the proxy methods (values for the proxy methods commitment variables). As a simple example, using Figure 3, if Agent C has a task that is enabled by a method owned by Agent A, Agent C assumes that the enabling method will be active at its release time.

The agent then schedules the best case what-if task structure using DTC. If DTC returns a schedule, that schedule is used in conjunction with the non-conditioned subjective view to determine the commitments required for the schedule to be valid. The times commitments are required by

are also updated to reflect the times required by the what-if schedule. For instance, returning to our example from above, if Task_3's first descendant method is scheduled at time 40, Method_2_2 (from Agent A) would not need to complete until that time.

Once the set of commitment requests is generated, they are sent to the agents owning the required tasks. If a task is shared by more than one agent, one is picked. Obviously, an extension to this protocol would be to request both tasks and then make some determination about whether redundancy would be beneficial.

GPGP's function is to determine set of commitments that maximizing quality at the task group. This is done in the UPDATESCHEDULE shown in Algorithm 3. The basic control flow of the GPGP protocol is given in Algorithm 2.

Algorithm 2: COORDINATE()

```
UPDATEPROXYMETHODS();
if needUpdate or inUpdateSchedule
  then
UPDATESCHEDULE();
  else
UPDATECOMMITMENTREQUESTS();
```

What the COORDINATE algorithm (Algorithm 2) shows is that the proxy methods and commitment requests are constantly updated with changes to the agent's subjective task structure. Often there are no updates necessary, in which case the functions return without performing any actions. The main processing of commitments to coordinate the agent's schedule according to its interactions with other agents occurs in the UPDATESCHEDULE function.

Algorithm 3: UPDATESCHEDULE()

```
taems ← SUBJECTIVEVIEW();
if not inUpdateSchedule
  then
ADDPROXYMETHODS(taems);
ADDNONLOCALCOMMITMENTS(taems);
ADDLOCALCOMMITMENTS(taems);
SCHEDULEWITHDTC(taems);
inUpdateSchedule ← true;
  else if DTCSCHEDULE() not null
  then
whatIf ← DTCSCHEDULE();
commitsSatisfied ←
 DETERMINECOMMITSSATISFIED(whatIf);
ACCEPT(commitsSatisfied);
DECOMMIT(commitsSatisfied);
inUpdateSchedule ← false;
```

The UPDATESCHEDULE function (Algorithm 3) is invoked whenever the *needUpdate* flag is set. It can be set in two ways. The first is by the Executive, which may have determined that, because of the actual execution characteristics of the scheduled methods, the schedule is no longer valid. The second is by a commitment message from another agent.

Once called, UPDATESCHEDULE begins its DTC schedule preprocessing step by setting the *inUpdateSchedule* flag so that when it is called a second time (on the next pulse), it will check to see if DTC has returned (as indicated by a non-*null* result from DTCSCHEDULE.

In the preprocessing step, UPDATESCHEDULE conditions the agent's subjective task structure by adding the proxy methods, nonlocal commitments (commitments from other agents to this agent), and local commitments (commitments from this agent to other agents). It then forks a process for DTC to schedule the task structure in the call SCHEDULEWITHDTC.

When DTC returns from scheduling, UPDATESCHEDULE performs the postprocessing on the DTC call by generating the set of commitments satisfied by the new schedule (DETERMINECOMMITMENTSSATISFIED). It then sends commitment accept messages for tasks that it had previously not made commitments for but that are in the satisfied commitment set. It also sends decommit messages for commitments that had been made previously but that are not in the set of satisfied commitments. It will also update commitments if they are in both sets, but there has been some change (say in the time that the commitment will be satisfied). Finally, the *inUpdateSchedule* flag is cleared.

Results

In this section, we present some preliminary results from some targeted investigations into the GPGP's behavior. As we noted previously, scalability issues with the DTC scheduler cropped up in our investigations which ultimately lead us to stop work on this branch of the project. (We intend to integrate the basic ideas of this work into a solution we developed in parallel that performed more robustly on the larger problems we tested on.) The following results are meant to demonstrate some of the capabilities of this solution.

The program evaluation scenario generator was used to generate each of the problems below. There are two basic parameters used in configuring the generator that are not referenced in the specific experiments: random NLE frequencies and NLE localities. Both soft and hard random NLE frequencies were set to 50% and NLE locality was set to be fully connected. That means, roughly, that there is a 50% chance for an NLE to be added to any permissible pair of nodes and that there are no special groupings of NLEs, e.g., two isolated groups. The one other aspect of the scenarios that is common throughout the experiments is the earliest release time. It was fixed at 100 ticks into the simulation to allow the agents some time to negotiate before they would be required to run anything.

The Central Solver quality reported is the maximum quality reported by the program evaluation team's centralized solver; the major advantages for the central solver are that it gets the original problem from the scenario generator, i.e., it has not been sectioned and distributed via the visible-to

procedure described previously, and it could take as long as it wanted to solve the problem.

Figure 5 shows GPGP's response to scenarios with increasing number of agents. Each scenario had 1 problem, 1 window, and CTasks with redundant methods, meaning that each CTask could be accomplished by at least two agents. Also, the failure probability was set to 50% which meant that 50% of the methods would fail 50% of the time. This was added to test the system's ability to deal with changes online. The data suggest that GPGP scales well in this type of scenario as the number of agents increases.

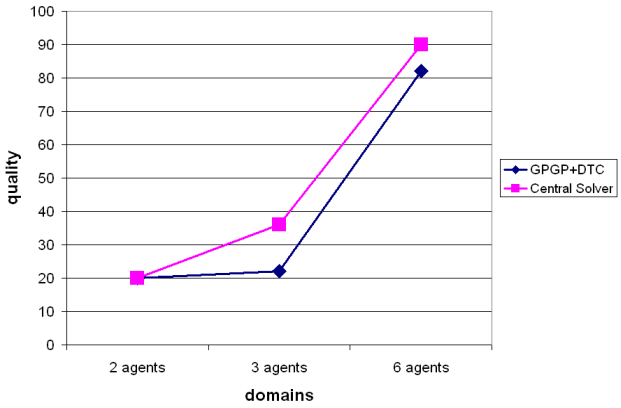

Figure 5: Effect of number of agents on solution quality on domains with 50% failure probability and 1 fallback method per CTask.

Figure 6 shows GPGP's response to scenarios with increasing probabilities of failure in both 3 agent scenarios and 6 agent scenarios. The data suggest that GPGP performance may degrade in cases where there are higher failure rates. The slightly better performance in 6 agent scenarios can be attributed to the redundancy available in those problems. Another note is that this comparison is between the maximum quality reported by the Centralized Scheduler, which makes some assumptions about the performance of methods that may be violated during the simulation.

Conclusion and Future Work

We have described a new implementation of the JAF architecture and Honeywell GPGP protocol that makes use of a domain-independent proxy method coordination context management technique, that retains real-time response time despite its reliance on a soft real-time scheduler, and that provides a generalization of previous implementations of GPGP that were targeted to specific domains.

One of the most obvious areas for future work is improving the responsiveness of the underlying scheduler technology. The performance of the current GPGP-DTC-Executive triad will degrade significantly on problems that cause DTC's processing time to explode, since either no schedule will be produced initially or none will be generated in a timely fashion on a rescheduling action once the existing

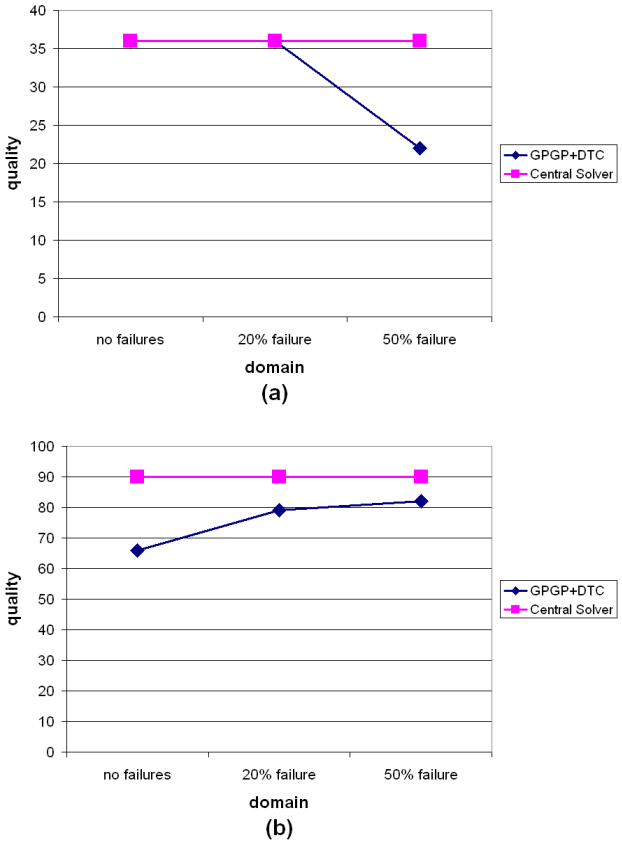

Figure 6: Effects of failure probability in (a) 3 agent scenario and (b) 6 agent scenarios.

schedule has been invalidated by some event. A separate investigation lead by Honeywell into an anytime MDP-based task analysis system as part of the Coordinators program promises to be a more fitting basis for future work on these types of problems.

Another area of future work is improving the way that agents negotiate commitments on shared tasks. Whereas currently, agents simply pick one of the agents capable of performing a shared task to provide a commitment in a pairwise fashion, a more sophisticated solution would approach this from a global constraint optimization perspective.

Acknowledgments

This material is based upon work supported by the DARPA/IPTO COORDINATORs program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

References

Decker, K. 1995. *Environment Centered Analysis and Design of Coordination Mechanisms*. Ph.D. Dissertation, University of Massachusetts, Amherst.

Graham, J. R.; Decker, K. S.; and Mersic, M. 2003. Decaf - a flexible multi agent system architecture. In *Autonomous Agents and Multi-Agent Systems*.

Horling, B. 1998. A Reusable Component Architecture for Agent Construction. Computer Science Technical Report 1998-49, University of Massachusetts.

Lesser, V.; Horling, B.; and et al. The taems whitepaper. http://mas.cs.umass.edu/research/taems/white/.

Raja, A.; Alexander, G.; and Mappillai, V. 2006. Leveraging problem classification in online meta-cognition. In *Working Notes of the AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*.

Vincent, R.; Horling, B.; and Lesser, V. 2001. An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator. In *LNAI: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, volume 1887, 102–127. Wagner and Rana (eds.), Springer,.

Wagner, T., and Lesser, V. 2001. Design-to-criteria scheduling: Real-time agent control. *Lecture Notes in Computer Science* 1887:128.

Wagner, T.; Phelps, J.; Guralnik, V.; and VanRiper, R. 2004. COORDINATORS - Coordination managers for first responders. In *AAMAS-04*.

Wagner, T.; Guralnik, V.; and Phelps, J. 2003a. A key-based coordination algorithm for dynamic readiness and repair service coordination. In *AAMAS-03*, To Appear. ACM Press.

Wagner, T.; Guralnik, V.; and Phelps, J. 2003b. Taems agents: Enabling dynamic distributed supply chain management. In *Journal of Electronic Commerce Research and Applications*. Elsevier.

Wagner, T. 2005. The DARPA/IPTO COORDINATORS program. http://www.darpa.mil/ipto/ programs/coordinators/.