Integrating High-Level and Detailed Agent Coordination into a Layered Architecture ⋆

Shelley XQ. Zhang, Victor Lesser, Anita Raja, and Thomas Wagner
Department of Computer Science
University of Massachusetts at Amherst
xqzhang@cs.umass.edu
phone: 413-545-0675

Abstract. Multi-agent coordination is an important and complicated process. This paper proposes a layered approach to coordination in which low-level domain independent coordination and scheduling modules deal with detailed temporal and resource constraints and high-level controllers focus on domain issues and domain state. A general agent architecture is described based on this agent control model. The integration of the low-level controllers with high-level JIL process programming language is used to explore the model. The possibility of integrating with other frameworks such as domain planners and BDI-based controllers is also explored.

1 Introduction

Coordination, which is the process by which an agent reasons about its local actions and the (anticipated) actions of others to try to ensure the community acts in a coherent fashion [7], is an important issue in multi-agent systems. There are three main reasons why coordination is necessary. First, there are dependencies between agents' tasks; second, there is a need to meet global constraints such as time and cost limits; and third, no individual agent has sufficient competence, resources, or information to solve the entire problem.

There are several frameworks, such as the BDI-based agent architecture [10, 1], the JIL process programming language [16, 8], and the teamwork model [11], that deal with high-level coordination issues. They logically reason about which task should be performed collaborately or the resources required for certain tasks, but they do not quantitatively reason about detailed coordination issues like the concurrent scheduling and selection of multiple goals, the temporal sequencing of subtasks, hard/soft deadline constraints, complicated resource constraints and utility differences between different options for performing a goal, etc. We feel that a coordination and control module that can provide these types of detailed quantitative reasoning would significantly augment the capabilities of these higher-level frameworks.

⋆ Effort This material is based upon work supported by the National Science Foundation under Grant No. IIS-9812755 and the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contract F30602-97-2-0032. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory/IFTD, National Science Foundation, or the U.S. Government.

Coordination is a complicated process that typically consists of several operations: exchanging local information; detecting interactions; deciding whether or not to coordinate; proposing, analyzing, refining and forming commitments; sharing results, and so on. We argue that facets of these different operations can be separated and bundled into two different layers, each relating to different classes of operations. In this view, the lower-layer pertains to *feasibility* and *implementation* operations, i.e., the detailed analysis of candidate tasks and actions, the formation of detailed temporal/resource-specific commitments between agents, and the balancing of non-local and local problem solving activities. In contrast, the upper-layer pertains to *domain specific* coordination tasks such as the formation of high-level goals and objectives for the agent, and decisions about whether or not to coordinate with other agents to achieve particular goals or bring about particular objectives. Detail domain state is used at this level to make these high-level coordination decisions. In contrast, decisions at the lower-level do not need to reason about this detailed domain state. However, reasoning about detailed models of the performance characteristics of activities, such as their temporal scope, quality, affects of resource usage on performance, is necessary at this level. In this view, the layers are interdependent activities that operate asynchronously. The upper-layer provides the performance criteria and the set of candidate tasks for consideration by the lower-layer. Whereas the lower-layer, as the result of its deliberation (in constructing the detailed commitments and temporal sequencing necessary to implement the desired objectives) may provide feedback to the upper-layer about the feasibility of certain tasks. In response to this feedback, the upper-layer may alter the set of candidate tasks or the performance objectives. This obviously can be a multi-step negotiation process.

The focus of our research is on a domain-independent low-level coordination module that deals with the detailed analysis and sequencing aspects of agent coordination, i.e., the "how to coordinate" issues. This coordination module includes the Design-to-Criteria (DTC) [13] scheduler and the Generalized Partial global planning (GPGP) [5] coordination system. By introducing this domain-independent coordination module into an agent, the high-level agent controller's coordination burden is significantly reduced (and the agent designer's task is simplified). Since the high-level agent controller does not need to deal with the details of the coordination process, it can put more concentration on domain-related coordination reasoning. Furthermore, because this coordination module is domain-independent, it can work with different high-level coordination frameworks. We present a general agent architecture based on this coordination module, so that agent developers need not do redundant work on detailed coordination actions.

In this paper, we describe the layered coordination model and introduce a general agent architecture based on the model (Section 2). In Section 3, we explore the layered model by integrating the low-level domain-independent module with a high-level JIL problem solver. We also explore the possibility of integrating the low-level module with other frameworks such as domain planners and BDI-based controllers in Section 4.

2 Integrating Coordination Approaches

Figure 1 presents a general agent framework based on the layered model. Before we explain how the agent performs the coordination process, we describe the functionality of each component.

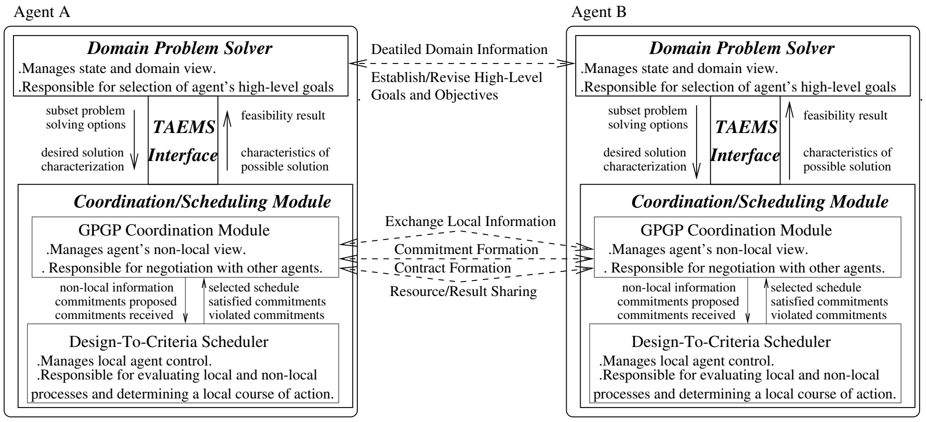

Fig. 1. Integrating Coordination Agent Framework

2.1 Domain Problem Solver Layer

The Domain Problem Solver Layer is dedicated to solving domain related problems, involving both local agent activities and collaborative activities with other agents. Different domain problem solvers may use different description languages, modeling structures, reasoning and analyzing strategies to solve problems. For example, JIL [16, 8] uses a process programming language perspective to describe the sequence of agent activities whereas the BDI framework uses logical expression of beliefs, desires, intentions to model domain problems and reasons about them. The domain problem solver layer models the domain problem, manages the system state, reasons and plans on how to solve problems, and establishes the performance criteria for the agent. The domain problem solver may communicate with other agents in the generation of potential activities.

2.2 TÆMS task modeling language and TÆMS interface

The TÆMS task modeling language [4] is a domain-independent framework used to model the agent's candidate activities. It is a hierarchical task representation language that features the ability to express alternative ways of performing tasks, statistical characterization of methods via discrete probability distributions in three dimensions (quality, cost and duration), and the explicit representation of interactions between tasks. Figure 4 contains an example of a TÆMS task structure. The TÆMS framework serves as a bridge that we use to connect the Domain Problem Solver Layer and the domain independent Coordination/Scheduling Layer. As mentioned, Domain Problem Solvers model processes using different languages, but the DTC scheduler and GPGP coordination components reason on TÆMS models, thus a TÆMS translation interface is needed between the domain problem solver and the coordination/scheduling module.

2.3 Coordination/Scheduling Layer

The coordination/scheduling layer evaluates the feasibility of performing goals/subgoals recommended by the domain problem solver layer, and, based on detailed resource constraint analysis, sets up the detailed temporal sequence and choice of local activities so that the multi-agent system meets its performance objectives. It provides the following functions for the domain problem solver:

- **Reasoning about the feasibility of activities** The domain problem solver has a plan to fulfill an objective, the coordination/scheduling module reasons about if it is feasible to accomplish the goal given the targeted performance criteria (duration limits, cost requirements, etc.). If the goal is a joint goal, the coordination/scheduling module should communicate with other collaborative agents and consider their activities in its plan.
- **Choosing from and sequencing possible activities** There may be multiple possible methods to accomplish the objective, the coordination/scheduling module chooses one solution that meets the criteria requirement. There also may be multiple activities the agent need to do, the coordination/scheduling module sequences them considering the task priority, the resource limitations and the interactions with other agents' activities.
- **Assisting the task allocation** The domain problem solver may have an intention to share a plan with other agents; the coordination/scheduling module will communicate with other agents and find a feasible task allocation pattern that meets the criteria requirement considering other agents' capabilities and current commitments.
- **Assisting the resource allocation** Resource sharing is a common issue in a multi agent system. By carefully arranging local activities, some resource conflicts can be avoided. The coordination/scheduling module will reason about local agent's resource requirement, communicate with other agent and help to allocate resources to avoid, or minimize the effects of, resource conflicts.

The coordination/scheduling module includes the Design-To-Criteria Scheduler and the GPGP Coordination system. The Design-To-Criteria scheduler [13] uses a domain-independent real-time, flexible computation approach to task scheduling. DTC efficiently reasons about the quality, cost, duration of interrelated methods, and constructs a set of satisfying schedules for achieving high-level goals. Resource requirements and proposed commitments are also taken into account when the scheduler selects a schedule. The scheduler provides the agent with ability to reason about the trade-offs of different possible activities while respecting the criteria requirement. The GPGP coordination module is a domain independent coordination framework that is based on Decker's original GPGP ([5]) and our recent extensions (GPGP2) [12]. It communicates with other agents, it also communicates with DTC scheduler, but these two kinds of communication are transparent to the domain problem solver. The main functions of the GPGP system include:

- **Exchange Non-Local Viewpoints** Agents exchange information about the activities that they are planning to perform and information about candidate activities that may be performed sometime in the future. In some domains (e.g., the interpretation domain as described in Section 4.3) this enables the local agent to determine which of its local tasks interact with other agents' non-local tasks. In other domains (e.g., the JIL process program as described in Section 3), the interactions of tasks are determined by the domain problem solver.
- **Contracting Mechanisms and Commitment Formation/Negotiation** If the domain problem solver want to assign tasks to other agents, the GPGP coordination module will communicate with other agent to verify the feasibility and finally accomplish the task allocation by establishing commitments with other collaborative agents. Different coordinate protocols are built in this coordination module, and the agent can select one appropriate protocol for the current problem solving context. The proposed commitment is analyzed given the

agent's current set of commitments and current set of scheduled actions, then a decision is made whether to accept/reject this commitment or refine it. Thus this commitment formation process is also a negotiation process.
– **Resource Acquisition** If certain resources are required for a task, the GPGP coordination module will communicate with the resource manager or coordinate with other agents to make sure the resource is available when the task is executed.
– **Results Sharing** Often commitments originate with the need to share results between agents. When results are produced, the producer agent must convey the required information to the consumer agent.

This agent framework works as follows: the domain problem solver analyzes its current problem solving situation and establish high level goals it want to achieve; also through the communication with other agents, it may decide some tasks need to be cooperatively performed. The coordination/scheduling module reasons about possible solutions to achieve the goal and sequences local activities. In this reasoning process, the criteria requirements such as the balance between achieving a good result quickly versus achieving a high quality result in a longer time, resource requirement and interaction with other agents are all been considered. The communication and reasoning process in the coordination/scheduling module are transparent to the domain problem solver. The GPGP coordination module communicates with other potential participant agents and builds proposed commitments for the common goal. The DTC scheduler reasons about local activities and these proposal commitments and verifies the feasibility of these commitments. If the proposed commitments are not suited for current objective, the GPGP module refines the commitments after negotiating with other agents. The GPGP module may also receive requests from other agents to establish commitments to achieve a particular result. The DTC scheduler also reasons about these requests given the current scheduled activities and find if these commitments are feasible. The scheduled activities and established commitments are returned to the domain problem solver for execution. In conclusion, the idea is that the domain problem solver decides what to do, the coordination/scheduling module decides how to do it and when to do it.

3 Coordination in JIL

3.1 Description of JIl Language

JIL [16, 8] is a process-programming language that is used to describe software development process and other processes. JIL represents processes as compositions of steps, which may be divided into substeps. The specification of a step is defined in terms of a number of elements. Each element defines a specific aspect of step semantics, such as data, control, resource usage, or consistency requirements. Briefly, the elements of a step specification are as follows:

– **Objects and declarations:** Parameters and local data used by the step.
– **Resources:** Specifications of resources needed by the step, including agent, software, hardware.
– **Steps:** Identification of the substeps of a step (which are themselves steps).
– **Step execution constraints:** Restrictions on relative execution order of substeps.

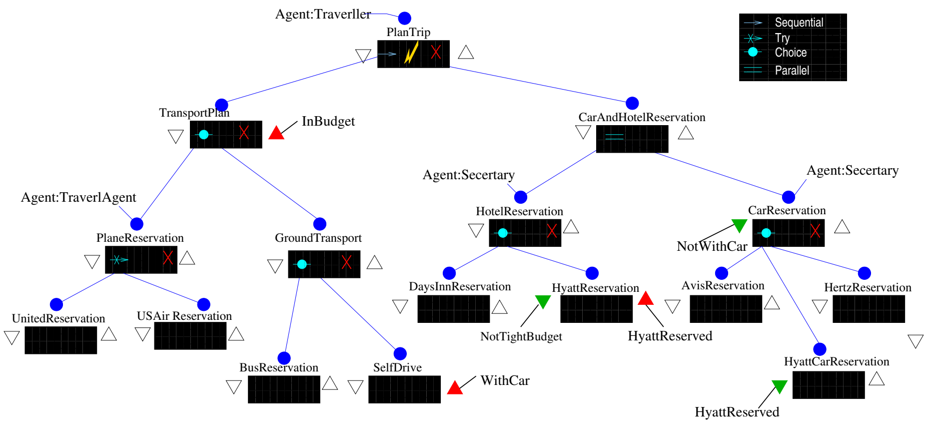

Fig. 2. Trip Plan Example

- **Preconditions, constraints, postconditions:** Consistency conditions that must be satisfied (respectively) prior to, throughout, and subsequent to the execution of the step.
- **Handlers:** Identification of handlers for local exceptions. Handlers can invoke substeps, thus exception handlers can use the full power of JIL to recover from errors.

Figure 2 contains an example of a "PlanTrip" process in JIL. To plan a trip(*PlanTrip*), the traveler needs to do two things in sequence: first plan the transportation (*TransportPlan*), then reserve a car and a hotel (*CarAndHotelReserveration*). To plan the transportation, the traveler has two choices, either ask a travel agent make the plane reservation (*PlaneReservation*) or plan the ground transport by himself. To reserve a car and a hotel, the traveler may ask a secretary to do both the car reservation and the hotel reservation, or assign these two steps to two secretaries so these two tasks can be processed in parallel. To make the plane reservation, the travel agent will try United Airline(*UnitedReserveration*) first, if it fails, then try USAir (*USAirReserveration*). To plan the ground transportation, the traveler has two choices, either take a bus (*BusReservation*) or drive a car (*SelfDrive*). Similarly, there are two choices to reserve the hotel - *DaysInnReservation* or *HyattReservation*; there are three ways to reserve car - *AvisReservation*, *HertzReservation*, and *HyattCarReservation*.

3.2 Our Work With JIL

JIL language provides a high level description of a process. It describes control flows, data flows, resource requirements of a process, this information constructs an overview of what this process should do, but there is not enough detailed information about how to perform this task to meet certain objective functions. For example, how the traveler could make a trip plan to meet his quality, cost, duration requirements? This problem is more difficult when this process is distributed among multiple agents, agents need to coordinate with each other to find a solution that meets the global criteria function. Furthermore, there are interactions among steps that need agents to coordinate over them, but JIL language does not represent them explicitly (which poses problems for multi-agent coordination). For example, if the traveler chooses to drive a car to the

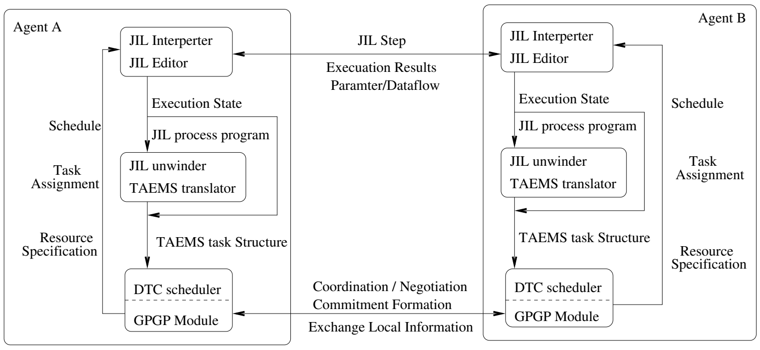

Fig. 3. JIL Agent Architecture

destination, there is no need for the secretary to reserve a car. Another example is the hyatt car could be reserved only if the hyatt hotel has been reserved.

Our solution to these problems is to integrate our coordination module with the JIL process problem solver. Figure 3 describes the infrastructure of an agent that works on a JIL process program. The JIL process program is generated by the JIL editor or is received from another agent(task assignment). The JIL interpreter should execute this process program. As we discuss above, the JIL interpreter can not reason about quantitative issues nor interactions among steps. This JIL process program is sent to the JIL unwinder, which opens this process program, discovers those interactions among agents, and extracts the resource requirement information. In the unwinding process, some steps are detected as non-local tasks (for example, the *PlaneReservation* step should be executed by a TravelerAgent), these non-local tasks are treated as virtual tasks: they will be analyzed but not executed locally. The TÆMS translator takes this progress program and the information provided by the JIL unwinder, generates a TÆMS

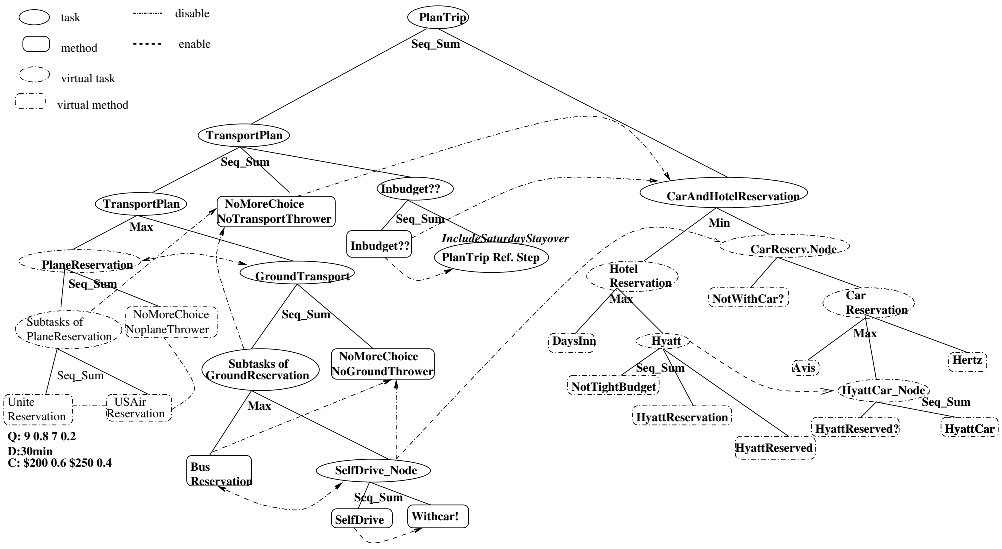

Fig. 4. TÆMS Task Structure for PlanTrap Example

task structure, as shown in Figure 4. The Design-To-Criteria scheduler works on this TÆMS structure and generates a schedule to meet the global criteria requirement. The schedule includes a set of methods with starting time, expected quality, cost and duration. Table 1 provides three schedule examples for different criteria requirements. Based on this first round scheduling, the agent tries to find appropriate agents to perform those non-local tasks though GPGP module. This task allocation process is a negotiation process because the agent which is assigned the non-local task may not be able to perform the task as the local scheduler specified, re-scheduling or re-assignment may be needed to achieve a satisfactory task assignment pattern. In this negotiation process, the agent also should take the resource requirements into consideration, making sure resources are available when they are needed. Resource acquisition is also a function built into the GPGP module. The "final" schedule with task assignments and resource specifications are returned to the JIL interpreter and the process is executed as scheduled. If, during the execution process, something unexpected causes the current schedule to be no longer valid, e.g., a step failing or over-running its costs or time expectations, the agent must communicate with other agents and/or reschedule its future activities through the GPGP module and the DTC scheduler.

	1	2	3
Quality Requirement	800	300	600
Cost Constraint($)	1000	400	700
Duration Constraint(Hours)	4	10	20
Transportation	UnitedResrvation	Self_Drive	BusReserv.
Hotel	HyattReserv.	DaysInnReserv.	HyattReserv.
Car	HyattCar		HyattCar
Expected Quality	720	350	630
Expected Cost	820	309	618
Expected Duration	4	9	14

Table 1. schedule examples

3.3 Benefits

The first benefit we gain from this organization is the reduction of the communication at the domain problem solver layer because the lower layer will automatically handle many issues. Coordination communication such as exchanging local information about execution state, building non-local commitments, are handled by the GPGP module, so the JIL problem solver does not have to deal with these issues.

The second benefit is that agents have global quantitative view of the process. Figure 5 shows a possible task assignment pattern for the PlanTrip process. Based on Figure 5, if the traveler wants to make a low cost trip plan, the schedule C is selected to meet the requirement. Secretary A will know the CarReservation is not necessary because of the disable relationship from the "SelfDrive" node to the "CarReservation" node (See Figure 4). Also secretary B will be informed that it is better to reserve the DaysInn hotel to meet the global criteria requirement.

The third benefit is that agents coordinate their actions and solve their conflicts easily though GPGP/DTC. Because we have discovered those inter-relationships among

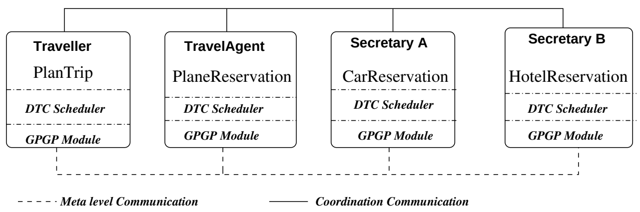

Fig. 5. Task Allocation for PlanTrip Example

steps and explicitly represented the non-local-effect (NLE) edges such as enables, disables, in TÆMS task structure, the GPGP/DTC module can handle these NLEs. For example, there is a disable edge from the GroundTransport to the PlaneReservation node, so once the traveler chooses SelfDrive, the TravelAgent will not have to do the PlaneReservation since it has been disabled.

4 Other High-Level Views of Agent Coordination

4.1 BDI Frameworks

The beliefs-desires-intentions (or BDI) model has been proposed as an agent architecture for quite a long time [10, 1]. Using this architecture, an agent has certain mental attitudes of belief, desire, and intention, respectively, the information, motivational, and deliberative states. The belief component represents the agent's information about the environment state, the desire component represents the objectives to be accomplished, and the intention component represents the currently chosen course of action, it is the deliberative component of the agent. These mental attitudes determine the agent's behavior. Usually, these mental states and their properties are formally defined using logical frameworks that allow agents to reason about them.

Some research work has been done to make BDI-agents behavior as social agents. GRATE* ([6]), was the first implemented BDI architecture treating joint mental states. It utilizes the concepts of joint intentions and joint responsibility to establish a collaborative activity and monitor the execution of joint activity. However, there is a gap both between the underlying descriptive theory and the architecture and between the functional and the implementation architecture. As Jennings himself points out: " ... the architecture does not specify how intentions are represented, how commitment is described, what circumstances are used to obtain agreements nor how to develop the common solution." The architecture of GRATE* does not sketch the details of the coordination such as how an agent can choose from multiple candidate goals or how the agent decides which action to perform at a given time. Our GPGP/DTC coordination module complements to this architecture on the lower level, it provides the ability to reason about the value of actions associated with the goals, to choose from alternatives, to work on the goals concurrently by interleaving actions and so forth.

Figure 6 shows a BDI agent architecture as implemented in GRATE* integrated with our coordination module. Events occurring as a result of local problem solving or change in the environment in addition to the events occurring elsewhere in the community, are monitored by *the monitor events* process. Events signify a potential need for a fresh activity and therefore a new objective. When a new objective is established

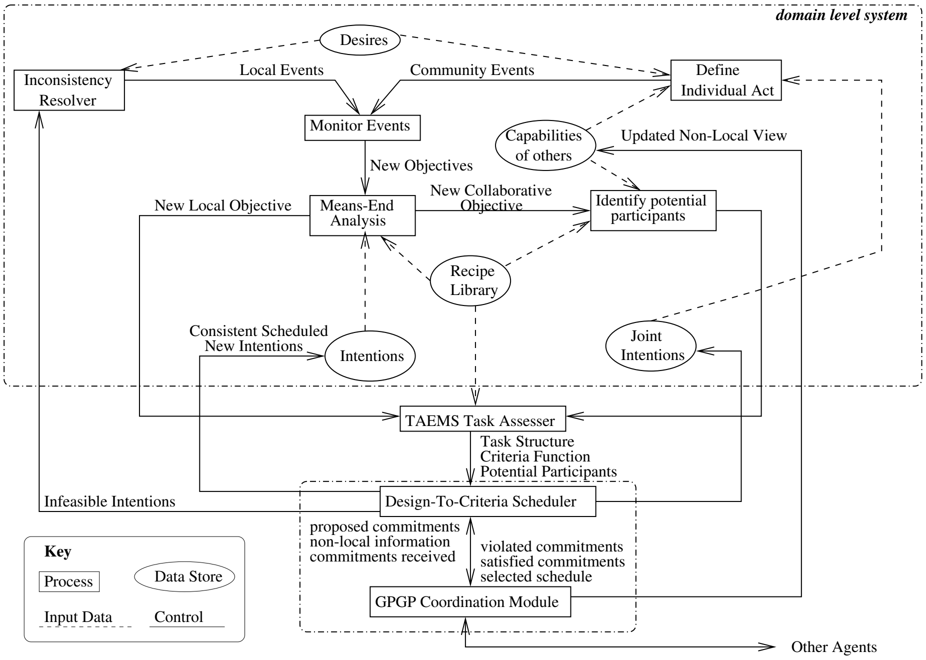

Fig. 6. BDI Agent Architecture with coordination module

for consideration, *the mean-end analyzer* refers to the *recipe library* to find appropriate plans to fulfill the objective. These plans indicated whether the objective should be satisfied locally, collaboratively or make a choice between the two. If the decision is to fulfill the objective collaboratively, a collaborative objective is established, and those agents in the community who are potentially interested in being involved are identified as potential members. The *TÆMS task assessor* takes the new local objective or the new collaborative objective, potential participants and possible alternative plans to achieve the objective as input information and builds a TÆMS task structure with potential task assignments and criteria requirements. The DTC scheduler schedules on this task structure and select an appropriated schedule. The GPGP coordination module communicates with other agents to determine if they can meet the collaborative objective as scheduled. If this is true, social commitments are built; otherwise, new proposed commitments are send back to the scheduler and a re-scheduling process is performed to find a proper schedule based on these new commitments. The scheduling-coordination process may be iterately refined to find a solution to fulfill the objective. If such a solution is found, the intention data store will be updated with a set of consistent scheduled local activities and the joint intentions will be updated with a set of social commitments; otherwise, infeasible intentions are detected, the *inconsistency resolver* attempts to either modify the existing commitments or alter the objective to remove the infeasible intention.

4.2 Organizational Experts

The integration path used to combine GPGP/DTC with high-level controllers, such as JIL, can also be used to expand and extend agent capabilities. Consider an organizational context expert of the form [15, 14] that views the agent as a socially situated problem solving entity. In this world view, agents belong to multiple different organizations, have multiple different roles, have different relationships with different agents, and interact in cooperative and self-interested styles.

Figure 7 shows a network of organized interacting information agents (in the WARREN style [3]). While a complete description of the network is beyond the scope of this paper, the agents in the network are associated with different corporate entities and interact in a way that reflects their different associations. Additionally, within a given organization, relationships may differ due to power relationships or differences in organizational structure, e.g., certain divisions may work closely together and cooperate freely while others may require intercompany transactions or payables to motivate cooperative activity. Agent control in such open environments, characterized by dynamic and complex agent relationships, requires reasoning about the different motivations for coordination and cooperative action with other agents.

This class of reasoning is inherently different than the type of detailed, temporal analysis performed by the DTC/GPGP feasibility experts. At this level, tasks are characterized in terms of *motivational quantities* (MQs) that are consumed or produced through task performance. Each agent has a set of MQs and agent utility is defined in terms of the MQ set held by the agent at any given point in time. Agents thus reason at this level about tasks from a utility perspective, but the temporal component is minimal (pertaining to opportunity costs) and detailed inter-agent interactions (and chains thereof) are not represented nor evaluated.

Integrating this social/organizational selection of high-level tasks and actions may take place using a vehicle similar to that used to integrate JIL and other high-level controllers. As with the other integration scenarios, the interface must be two-way as the organizational-view of the world is incomplete and the controller may select a set of candidate tasks that are difficult to implement in any desirable fashion. This component, however, differs in that the utility computations used to select high-level tasks

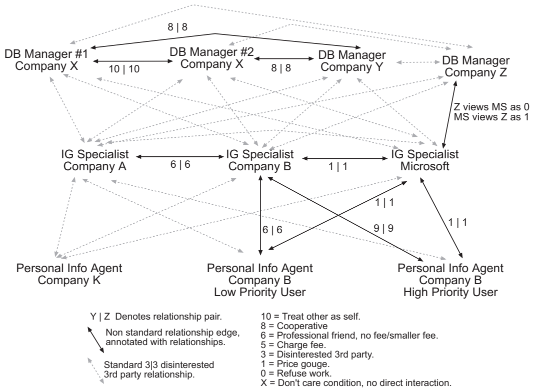

Fig. 7. An Organized Network of Interacting Agents

must also be reflected in the quality, cost, and duration characteristics of the primitive actions represented in TÆMS and reasoned about by DTC/GPGP. In this scenario, organizational context must *influence* the statistical characterization of tasks and methods. In contrast, in the JIL integration, the characteristics of task and methods are part of the domain model and thus the high-level selection process is independent from the characterizations of primitive actions. The question of what level of influence the organizational component should have on the inherent properties of tasks and actions is a current research issue.

4.3 Interpretation Problem Solvers

Still another example of this integration approach is the combination of interpretation-style (blackboard) domain problem solving experts with the feasibility and detailed agent control tools. Integrating problem solvers like RESUN [2] with the lower-level agent control tools gives the problem solvers the ability to meet real-time deadlines and to perform detailed analysis of their problem solving process. However, the integration is slightly more complicated than it is in the JIL case as blackboard-based problem solvers typically employ an opportunistic control mechanism; problem solving is both top-down (expectation driven) and bottom-up (data driven). In contrast to the JIL view of the world, there is no high-level scenario or global view of the process and *predictability* is, by definition, difficult. It is important to note, however, that this problem solving model is well suited to distribution as communication from other agents resembles, in some sense, the same bottom-up processing that occurs locally, i.e., interactions are data driven. The added complexity takes the form of communications and processing overhead and the possibility of the local agent being distracted by the additional hypothesis and information.

The reactive / opportunistic problem solving style, however, complicates integration and requires that the problem solvers be able to constrain their opportunism for windows or intervals of some temporal scope, enabling agents to coordinate and schedule. The interface requires a careful balance as these islands of stability constrain or limit the strength of opportunistic problem solving. One solution, used in the BIG information gathering agent [9] is to schedule from a certain level of abstraction, i.e., the primitive actions modeled in TÆMS are not actually primitives to the agent's problem solver, but are instead black boxes during which it will perform a particular class of operations. The scheduling/coordination tools thus operate from a coarse temporal view and the problem solver remains free during these intervals to perform opportunistically. As with the other integration examples, this example too requires that the components have a two-way interface, enabling major changes in problem solving to drive rescheduling and recoordination as well as feasibility analysis affecting or changing the goals of the problem solver.

5 Conclusion

Coordination is a complicated process. We propose to extract the domain-independent coordination issues and solve them through our lower-level, domain-independent, GPGP/DTC coordination module. We have done some work on the integration of the JIL problem solver, and an interpretation problem solver (Section 4.3), with the lower-level tools as a first step in the verification of this idea. We have also considered the possibility of integrating the low-level tools with other higher-level agent control frameworks. Our goal is to establish a domain-independent, powerful, easy-to-use, plug-in coordination module that solves the generalizable aspects of the coordination problem in an efficient way.

References

1. M.E. Bratman. *Intention Plans and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
2. Norman Carver, Zarko Cvetanovic, and Victor Lesser. Sophisticated cooperation in FA/C distributed problem solving systems. In *Proc. of the 9th National Conf. on AI*, pages 191–198, Anaheim, July 1991.
3. K. Decker, M. Williamson, and K. Sycara. Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9:239–260, 1997.
4. Keith S. Decker. Task environment centered simulation. In M. Prietula, K. Carley, and L. Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*. AAAI Press/MIT Press, 1996.
5. Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In *Proc. of the 1st Intl. Conf. on Multi-Agent Systems*, pages 73–80, June 1995. AAAI Press.
6. N.R. Jennings. Specification and Implementation of a Belief-Desire-Joint-Intention Architecture for Collaborative Problem Solving. *Intl. Journal of Intelligent and Cooperative Info. Systems*, 2(3):289–318, 1993.
7. N.R. Jennings. Coordination Techniques for Distributed Artificial Intelligence. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. Wiley Interscience, 1996.
8. David Jensen, Yulin Dong, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil, Stanley M. Sutton Jr., and Alexander Wise. Coordinating agent activities in knowledge discovery processes. In *Proc. of Work Activities Coordination and Collaboration Conference (WACC)*, 1999.
9. Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. BIG: A resource-bounded information gathering agent. In *Proc. of the 15th National Conf. on AI (AAAI)*, July 1998.
10. A.S. Rao and M.P. Georgeff. Modelling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. of the 3rd Intl. Conf. on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann, 1991.
11. Milind Tambe. Agent Architecture for Flexible, Practical Teamwork. *Proc. of the 14th National Conf. on AI (AAAI)*, 1997, pages 22–28.
12. Thomas Wagner, Brett Benyo, Victor Lesser, and Ping Xuan. Investigating Interactions Between Agent Conversations and Agent Control Components. In *Agents 99 Workshop on Conversation Policies*, 1999.
13. Thomas Wagner, Alan Garvey, and Victor Lesser. Criteria-Directed Heuristic Task Scheduling. *Intl. Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. A version also available as UMASS CS TR-97-59.
14. Thomas Wagner and Victor Lesser. Relating Quantified Motivations for Organizatioanlly Situated Agents. In *Under review*, 1999.
15. Thomas Wagner and Victor Lesser. Toward Generalized Organizationally Contexted Agent Control. In *AAAI Workshop on Reasoning in Context*, 1999.
16. Alexander Wise, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil, and Stanley M. Sutton Jr. Specifying Coordination in Processes Using Little-JIL. UMASS CS Technical Report UM-CS-1998-038, 1998.