Component-based Approach for Multiagent Coordination

Wei Chen and Keith Decker

102 Smith Hall,
Department of Computer and Information Sciences,
University of Delaware,
Newark, DE 19716, USA,
Tel: 1-302-8313183, Fax: 1-302-8314091
wchen,decker@mail.eecis.udel.edu

Abstract. [1] Coordination is a key functionality in distributed artificial intelligence and distributed systems. Previous researchers stated that it is promising to develop coordination mechanisms for software intelligent agents, so that they can adapt their behaviors in a coordinated way to solve the problems that are beyond the ability of a single agent. We developed an extended set of coordination mechanisms for our agents so that they are able to coordinate with each other. All the agents have their own decision-making components, which decide the strategy of selecting appropriate coordination mechanisms in dynamically changing environments. Our approach is a general solution for coordination problems. The idea is based on the separation of domain actions from meta-level coordination actions when creating coordination mechanisms. Component-Based Software Engineering/Development [2] is highlighted in both industrial and academic fields with features like low maintenance cost, high reusability and standard communication interface, etc. The coordination mechanisms we are studying are suitably implemented with the component-based approach with the advantages of code reuse, design pattern reuse and standard communication interface. This paper introduces general coordination problems, describes how to implement the coordination mechanisms as software components, and discusses the advantages of wrapping coordination mechanisms with component-based software. Experimental framework and initial experimental results are also presented.

1 Introduction

Coordination has been widely studied upon many different disciplines [14][15][21] including organization theory, management science, economics, linguistics, psychology, biology, sociology, and the most important field to us, computer systems.

[1] This work is supported by NSF Grant No. 9733004.

[2] Agent-Oriented Software Engineering is mentioned as the next generation of Component-Based Software Engineering [12]. Here we concentrate on developing components in coordination level, not in the agent level. So we just name our coordination mechanisms as components in this paper.

Researchers in computer science [1][5][7] opened a new vision towards the study of coordination. As more and more exploration about coordination has been practiced, it has been not only a simulation task, but also a mathematically formal work in multiagent systems. The most exciting step is that coordination technology has entered real applications.

From multiagent system researchers' point of view, there are three ways to handle coordination problems: 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. There are advantages and disadvantages about these different approaches, and many possible realizations in different environments. In order to develop a general approach for the problems in coordination theory, we should not only study these different approaches themselves, but also exploit useful knowledge from different prospects and areas, such as software engineering, especially component-based software development, to facilitate the actual implementation. Then we can start the approach with a base model, which is supported by software engineering.

Coordination problem is regarded as *managing dependencies between activities* [16]. We can learn some facts based on this definition. Firstly, in many situations coordination is inevitable. For example, a team of soccer players is more likely to win the game with better coordination. Midback holds the ball at first and at the same time observes the positions of his fellow players and the opponent players. The strikers and the players in midfield must run to find a good position, and get the midback's attention to receive a feed. Midback passes the ball to the left midfield, who runs to the best position, to start the attack. All the other fellow players move accordingly for their best attacking positions. The left striker runs to the leftfielder to receive the ball and kicks a long pass to the right striker. Finally, the right striker shoots to score. We just described a very successful coordination task among a team of soccer players. On the other hand the lack of coordination could reduce the system efficiency, leave important tasks to be incomplete, or misuse of vital resources, which would cause system failure. Secondly, to simply construct a central coordinator can never avoid the bottleneck problem in complicated, potentially high volume communicating, open system environments. Distributed agent systems could not suffer from failure of central coordinator(s), since agents usually do not have the complete image of their external environment. And it is the coordinator that handles everything at a higher level to ensure the smooth running of the entire system. Finally, if each individual agent is equipped with coordination ability and hopefully is capable of selecting different coordination mechanism(s) under different environments, the multiagent systems may achieve the expected stable functions. [19] describes how the agents can perform well based on the modeling the environments, and we try the similar idea to model the environmental factors for multiple agents' coordination as described in [3]. The path to follow for coordination problems that we think appropriate is to construct coordination mechanisms to handle the dependency relationships among multiple agents' tasks.

GPGP and TAEMS are the bases of our coordination research. The key point of GPGP (Generalized Partial Global Planning) (Both GPGP and TAEMS are explained in [6]) is to assume that each agent is capable of reasoning locally about its schedule of activities and possible alternative schedules. Thus coordination mechanisms can be thought of as ways to provide information to a local scheduler that allows it to construct better schedules. We represent agent task structures using TAEMS (Task Analysis, Environment Modeling, and Simulation). TAEMS 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 the characteristics over which an agent expresses utility preferences (e.g. cost, quality, duration, etc.) and how the execution of some actions or subtasks *change* these characteristics at another subtask — thus setting up the potential for some coordination to occur (or not).

We construct the coordination mechanisms based on Hierarchical Task Networks alteration as actual coordinating behavior and the component model as underlying technology. This paper will present the basic ideas and models about this approach. We will also explore a framework to simulate different coordination mechanisms under different environmental factors and briefly introduce how this approach would be appropriate to be applied to real world applications [3] [19], like query planning and execution in an information gathering system. Finally, we conclude with initial experimental results and a discussion of future research.

2 Developing Coordination Mechanisms to solve coordination problems

2.1 Coordination Simulation Environment For Our Approach

Multiagent system researchers deem coordination as handling the dependencies among agents' activities. The coordination mechanisms can be constructed as sub-level components for every agent. Our goal is to enable the agents to adapt themselves under different environments and to select the best instantiation of the coordination components based on proper coordination mechanism selection strategy.

We use DECAF(Distributed, Environment-Centered Agent Framework) system [11] as agent construction tool. Each DECAF agent consists of several components, which maintain the agents' functions and keep congruent to the traditional BDI (Belief,Desire,Intention) model. The coordination mechanism construction is independent from the agent construction. That is to say, DECAF agent can be equipped with coordination components that are developed beforehand. Based on the features of coordination mechanisms, as we will describe in the next section, it is advantageous to implement these mechanisms as components.

As figure 1 shows, agent's components are Agent Initializer, Message Dispatcher, Planner, Scheduler, Executor and GPGP Module. All components carry

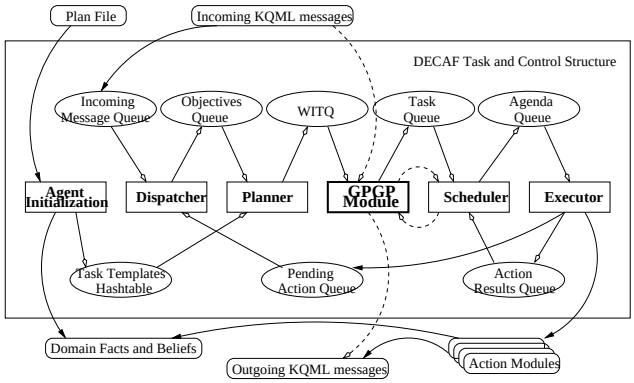


Fig. 1. GPGP Component in Agent Architecture.

the functions as their names indicate. We concentrate on one special unit, the GPGP component.

There are previous approaches towards coordination with component-based methods[17][18]. They usually construct whole agents with different roles based on the simulation environment or the analysis of coordination processes and subprocesses. Those approaches are domain specific and can not be used with even a small change of environmental factors. While our research is from a different prospect and concentrate on the low level task structure analysis. Because all the tasks in all domains can be represented with hierarchical task structure, we claim that it is a general approach to handle coordination problems by utilizing the task structure information. Our vision is to *explore dependencies into a finer indivisible granularity to analyze the agents' tasks* [2][4]. Because all the dependencies are composed of enable or facilitation relationships among agent tasks and these non-local relationships can be represented with special task structures (non-local tasks), we analyze these relationships' task structures, detect the potential coordination points, develop a set of re-usable coordination mechanisms, associate the mechanisms to every agent and finally apply the coordination mechanisms at those coordination points. Plus, with these embedded coordination mechanisms, each agent would be able to learn over time which coordination mechanism(s) are the best for a specific environment [3].

As figure 2 shows, DECAF task structure representation is composed of Task (e.g. TaskA), Action(leaf nodes of the task structure, e.g. A1), Non-Local Task (NLT), Provision Cell (the small boxes attached to the tasks and actions, e.g. the box named OK), Characteristic Accumulation Function (CAF represents the relationship between the subtasks and their common parent task, e.g. the relationship among the subtasks with common parent task Sub2 is *AND*, which means Sub2 can not be finished until all its subtasks are all finished) and Action Behavior Profile (represents the features of the task/action, e.g. the ABP of task Ask is (Quality=Qa21, Cost=Ca21, Duration=Da21)). We make the abstract structures concrete by explicitly indicating that the provisions (input values)

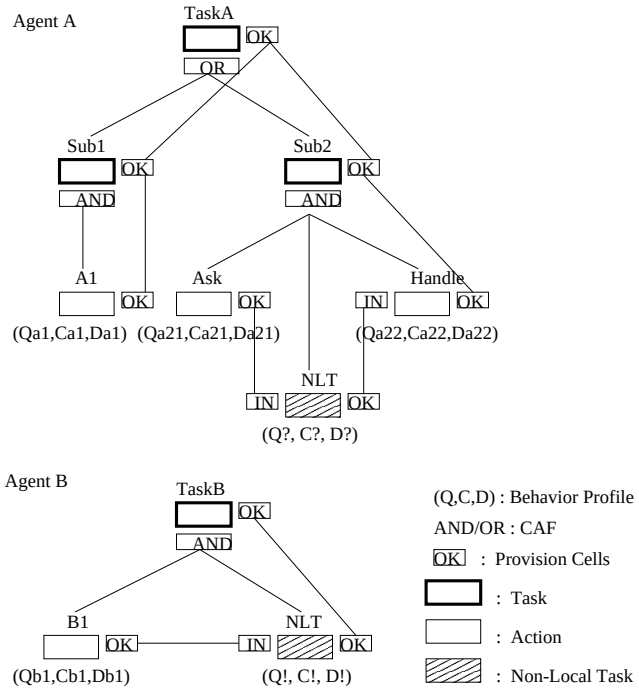


Fig. 2. Actual DECAF agent Task Structure.

are needed before the actual execution of the corresponding task. Only with the available input values in the provisions can we begin to compute the outcomes for this task. A *Provision Cell* is a data structure that receives the expected input data as the precondition before this action can be executed. The outcome cell at the right side of the task is filled with the result value after this action is finished. The line between the outcome cell and the provision cell, such as the one from the outcome provision cell OK of Ask to the input provision cell IN of the NLT (Non-Local Task), means that the value of the outcome of the action, Ask, is transported to the input of NLT by a FIPA message [9]. The task NLT is instantiated by the incoming FIPA message. The NLT can not begin its execution until the input value is filled by an incoming message. In this way, the relationships among the agents' tasks can be represented naturally with hierarchical task structure.

As figure 2 shows, Agent_A's problem is that the scheduling information of NLT in Sub2 is unknown, and this unknown value directly causes bad scheduling and leads to bad coordination. The HTN alteration will remove the unknown factors by replacing Agent_A's NLT task with Agent_B's TaskB through meta-level (the coordination level) communication between the coordination components in peer agents. With previously unknown non-local scheduling information filled,

our agents can make use of the local scheduler's scheduling power to develop better schedules [10] [11] [20].

2.2 Actual Coordination Mechanisms

We already designed seventeen coordination mechanisms as components [3]. These mechanisms can be catalogued into different groups. They are *avoidance* (with or without quality 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 transferring mobile code (promotion or demotion), various third-party mechanisms, or more complex multi-stage negotiation strategies. The input and output to the coordination mechanisms are standardized and explained in detail in the next section as component description.

Different coordination mechanism exceeds others under different environments. The later section about experimental framework gives an overall evaluation. Notably, there is no single best coordination mechanism that outperforms others under all circumstances. Every agent should learn over time which coordination strategy is appropriate based on the current environmental factors. As the factors may change in the agent systems, agent should adapt its behaviors accordingly to achieve the well coordinated states. Component-based approach makes this task easy in that the development of coordination component is separated from application program, it is convenient for the agent to switch coordination components dynamically, and the reuse of the coordination mechanisms is applicable.

3 Implementing Coordination Mechanisms with Component-Based Approach

Component-based approach needs to conform to some specific component models. The component models define two facts [13]: First, how individual component is constructed; Second, how multiple components communicate and interact with each other. We have introduced DECAF as an agent construction tool. Next we explain the implementation of our coordination components and the communication interface.

The coordination mechanisms described in this paper reach the definition of the component described in the previous paragraph in that the coordination functionality can be independently deployed and developed without modification according to their composition standard.

[3] In this paper we concentrate on how to represent coordination mechanism with component-based approach. Please read [2][3][4] for detailed explanation of the implementation of the coordination mechanisms.

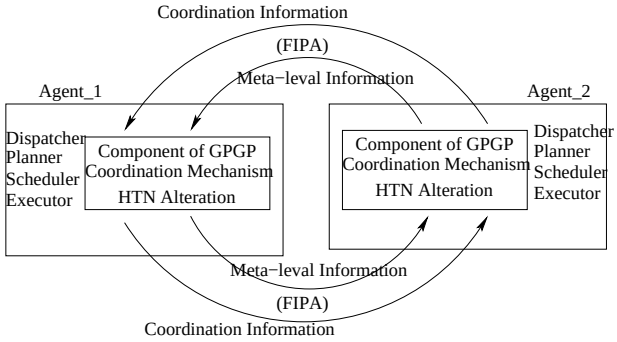


Fig. 3. GPGP Coordination Component and Communication Between Two Components.

Figure 3 shows that the primitive internal structure of GPGP component is based on HTN alteration, which is the actual coordination function and implementation of the component. It also depicts that the components communicate and interact with each other by passing FIPA messages.

The inputs to coordination components are *task group* $\mathcal{T}$, *meta-level coordination information* $\mathcal{C}$ and *environmental factor* $\mathcal{E}$. $\mathcal{T}$ is composed of *tasks* $\langle \mathcal{T}_i \mid i = \textit{the number of task cells in the task group } \mathcal{T} \rangle$ and *relationships* $\mathcal{R}$. $\mathcal{T}_i = \langle TaskID, Quality, Duration, Cost... \rangle$ is a feature vector representing the basic units of tasks/actions in the hierarchical task network tree. Quality, duration and cost (more factors may be used to represent more features about the task node) are characteristic factors for every task/action node, representing how much utility value, how long and what will be the cost of the execution of the action node. $\mathcal{R}$ represents the relationships between tasks. $\mathcal{R}(\mathcal{T}_i,\mathcal{T}_j) = \mathcal{P}$, if they have parent-child relationship; $\mathcal{S}$, if they are siblings; $\mathcal{N}$, if they are separated with each other from different tasks. Coordination relationship $\mathcal{C}$ = LE: for local effects/relationships; or NLE: non-local effects. $\mathcal{C}$ is the actual internal coordination execution, which is HTN alteration, of the coordination components. Environment $\mathcal{E}$ represents the characteristics of the agent system, such as the number of agents in the system, communication load, coordination task error rate, etc. Based on the previous definitions, we describe the coordination component conveniently as follows:

$$(\mathcal{T}', \mathcal{C}', \mathcal{E}') = \varphi(\mathcal{T}, \mathcal{C}, \mathcal{E}).$$

φ is an abstract function which takes original task structure and other parameters as input, starts the coordination task based on HTN alteration, and finally outputs the modified parameter vector and the coordinated task structure.

As we may have noticed that figure 3 also shows the communication and interface between the coordination components within different agents. The interface between the agents and the coordination components are the most important parts, since efficient and expressive communication is vital for coordinating the

agents' behaviors. It is almost impossible for the agents to coordinate well without interacting with each other in complicated environments. The agent communication language used for coordination task is FIPA, semantic-based language and more and more popular today [9].

We show a simple example here with FIPA messages. The coordination mechanism is selected as *Promotion Shift*, which is initiated by a Facilitator agent as *request* and responded by a Facilitatee agent as *inform*.

```
(request
  :sender (agent-identifier :name Facilitator)
  :receiver (set (agent-identifier :name Facilitatee))
  :content
    ((action (agent-identifier :name Facilitator)
      (inform-ref
        :sender (agent-identifier :name Facilitatee)
        :receiver (agent-identifier :name Facilitator)
        :content
          "((iota ?x ((TaskStructure
                      (:name FacilitatorTask) ?x))))"
        :ontology UDEL-coordination-mechanism
        :language FIPA-ACL)))
  :reply-with ID1
  :language FIPA-ACL)

(inform
  :sender (agent-identifier :name Facilitatee)
  :receiver (set (agent-identifier :name Facilitator))
  :content
    ((= (iota ?x (TaskStructure (:name FacilitatorTask)
              "Seat Reservation Request")))
        (:timing "immediately"))
  :ontology coordination-mechanism
  :in-reply-to ID1)
```

Because of some reasons, for example as being busy, the Facilitator agent wishes to do the task later, a message informing the delay will be transferred as an alternative way for the reply.

```
(inform
  :sender (agent-identifier :Facilitator)
  :receiver (set (agent-identifier :name Facilitatee))
  :content
    ((= (iota ?x (TaskStructure (:name FacilitatorTask)
                 BLANK)))
        (:timing delay))
  :ontology coordination-mechanism
  :in-reply-to ID1)
```

There might be additional rounds of the Facilitatee agent's message passing, confirming the receipt of coordination information. Or, the Facilitator uses *refuse*

message to reject the Facilitatee's reservation request. Here we do not bother to elaborate that procedure and just show that our coordination components have standard communication interface between each other.

Based on the above component model, we state that the implementation of the internal mechanisms is separated from agent construction. An agent may choose different coordination mechanisms as independent components according to the environmental changes without concerning the issues inside the components, e.g. the implementation and potential update. Our component is "written in a language dedicated to the coordination of processes and the flow of information among them." [1] We separate the application language and the coordination language so that we can concentrate on the complex and expensive part of the coordination. We can plug in graphic modules to give coordination components as options to our agents. This graphical design is suitable for DECAF in that this agent development tool is graphics-oriented already. The agent developers (human programmers) only need to place graphic components into the drawing area to compose task structures. More ambitiously, our goal of coordinating agents is to make it autonomous, which means that under different environments the agents are able to detect the potential changes of the external environmental factors, select appropriate coordination mechanisms accordingly and take the best actions based on the coordination selection strategy. [4]

The advantages of reusing the coordination components are as follows: The first advantage is the code reuse. Agent developers save the task of writing code by clicking and dragging the coordination components into an agent in order to equip the agent with coordination ability. The second advantage is design pattern simplification. Without the coordination component, agent developers have to explicitly design the domain task structure for the coordination tasks if the agent does need to behave coordinately. But it is difficult for the developers because they can not foresee the coordination points in the system. With the coordination component, agent developers can choose whether to use their own control of coordination by manually developing the task structures for coordination, or to simply make use of the coordination components without worrying about specifying the coordination points at design time.

4 Experimental Framework and Experimental Results

We created an experimental framework to test the performance of these mechanisms/components under various environmental factors. It can be applied to real applications as well, for instance, query planning and 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

[4] There is a fact we would like to point out that since our project is developed using Java, Java Bean is a good component model candidate for industrial applications. We also developed a general solution for the understanding of the relationships between various application domains and coordination behaviors. A paper describing this solution in detail is submitted to another workshop in AAMAS02.

wrapper agent queries remote databases, extracts the expected information, and sends the sub-results back to an management agent for result composition. One wrapper agent may need the results from other wrapper agents. These results are either from further sub-queries carried by other wrapper agents, or further information extraction from remote databases. The scheduling for a wrapper agent to access multiple information resources, the explicit enable relationships between the reasoner and the wrapper agent, and the interdependencies among the wrapper agents, provide coordination components with a perfect stage to perform on.

In this framework *Error Rate*, *Task Repeat Rate*, and *Agent Load* are some of the changing factors that describe an environment. Multiagent 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.

Each experiment with respect to a different coordination component was repeated 20 times for each environmental condition. We first show the changes in final quality and the communication load qualitatively.

Env.	*Mechanisms*						
	A	SA	RES	D	SR	P	CH
QC	>=0	Dep	>0	>0	>0	textgreater0	>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 for simulation purpose, 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 ensure faster execution, and it is 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, the CLs of these two mechanisms are the same value, zero. 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. Mechanisms of Reservation and Sending-Result are similar because they cost two rounds of non-local communication time; But they are still different from each other in that whether coordination meta-information (for Reservation mechanism) or the actual domain result (for Sending-Result mechanism) is sent back. Polling obviously has the highest communication load 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. [5]

5 Conclusion and Future Work

We have introduced the implementation of coordination mechanisms with an agent-oriented approach. We present several points as follows: We have shown that the coordination mechanisms have formal construction model and standard communication interface; It is promising and advantageous to introduce the agent-oriented software design technology into the coordination research, which means the construction of coordination behaviors is wrapped by components so that the software developers do not need to worry about the stability of the system when updating the underlying coordination mechanisms and the application behaviors; Finally, low maintenance cost and high reusability are achieved.

When developing coordination mechanisms, we not only care about the functions of these mechanisms, but also pay special attention to the reuse and efficiency issues, and especially the interaction among the agents. One future task is to smoothly integrate these coordination components into DECAF system, so that the agent developers simply select the visualized components as candidate coordination mechanisms for their agents. Another future task is to design a component pool to store all the mechanisms in a standard way, so that the agents are able to select not only one, but also a combination of multiple components under various environments. Finally, how to enable every agent with the ability to dynamically select different coordination components is an important open research task. Machine learning technology may be a suitable method to follow for this issue and more experiments are expected.

References

1. F. Arbab, C. Blom, F. Burger and C. Everaars. Reusability of coordination programs. CWWI report, CS-R9621, ISSN 0169-118X, 1996.
2. W. Chen and K. Decker. Coordination Mechanisms for Dependency Relationships among Multiple Agents. In *First Joint Conference on Autonomous Agent and MultiagentSystems*, Bologna, Italy, July, 2002.
3. W. Chen and K. Decker. Applying coordination mechanisms for Dependency Relationships under Various Environments. In AAMAS02 Workshop on Toward an Application Science: MAS Problem Spaces and Their Implications to Achieving Globally Coherent Behavior, Bologna, Italy, July, 2002.

[5] We omit the detailed quantitative analysis in this paper. Further description about the experiment can be found at [2][3][4] and the extended version of the paper.

4. W. Chen and K. Decker. Developing Alternative Mechanisms for Multiagent Coordination. Intelligent Agents and Multi-Agent Systems, P63-76, LNAI-2413, Springer-Verlag, Tokyo, Japan, August, 2002.
5. K. Decker. Distributed problem solving: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(5):729–740, September 1987.
6. K. Decker and V. Lesser. Designing a family of coordination algorithms. International Conference on Multi-Agent Systems, San Francisco, 1995.
7. E. Durfee, V. Lesser, and D. Corkill. Cooperative distributed problem solving. In A. B. Barr, P. Cohen, and E. Feigenbaum, editors, *The Handbook of Artificial Intelligence*, volume 4, pages 83–147. Addison Wesley, 1989.
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. Foundation for Intelligent Physical Agents (FIPA). FIPA 2000 Specification. *http://www.fipa.org/repository/fipa2000.html*
10. A. Garvey and V. Lesser. Design-to-time Real-Time Scheduling IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Scheduling, Planning, and Control, 23(6), 1993.
11. J. Graham and K. Decker. Towards a distributed, environment-centered agent framework. *Intelligent Agents IV, Agent Theories, Architectures, and Languages*, Springer-Verlag, 2000.
12. M. Griss. My Agent Will Call Your Agent. Software Development Magazine, Feb., 2000.
13. G. Heineman and W. Councill. Component-Based Software Engineering: Putting the Pieces Together. ADDISON-WESLEY, May, 2001.
14. C. Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.
15. B. Huberman. The ecology of computation. 1988. Amsterdam: North-Holland.
16. T. Malone and K. Crowston. The interdisciplinary study of coordination. In *ACM Computing Surveys*, pages 87–119, March 1994.
17. V. Lesser and et. al. The Intelligent Home Testbed. In *Proceedings of the Autonomy Control Software Workshop (Autonomous Agent Workshop)*, Seattle, 1999.
18. R. Menezes and A. Wood. Coordination of Distributed I/O in Tuple Space Systems. *Proceedings of 31st IEEE Hawaii International Conference on System Sciences*, Hawaii, USA, IEEE, Jan 1998.
19. J. Odell, H. Parunak, M. Fleischer and S. Brueckner. Modeling Agents and their Environment. In proceedings of AOSE 2002, Bologna, Italy, July, 2002.
20. T. Wagner, A. Garvey and V. Lesser. Complex Goal Criteria and its Application in Design-to-Criteria Scheduling. In *AAAI97*, Providence, Jun 1997.
21. M. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.