Toward Interaction-Oriented Programming

Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA

singh@ncsu.edu

+1.919.515.5677 (voice)
+1.919.515.7896 (fax)

Abstract

Although much progress has been made in agent theory and practice, bottlenecks remain in the construction of complex multiagent systems. We introduce *interaction-oriented programming (IOP)* as an approach to orchestrate the interactions among agents. IOP is more tractable and practical than general agent programming, especially in settings such as open information environments, where the internal details of autonomously developed agents are not available. IOP facilitates multiagent system design by enabling declarative specification and enactment of agent interactions, thereby channeling the intellectual energies of designers into the most amenable and effective design tasks. We develop an event algebra to specify interactions among agents. We automatically compile these declarative specifications into executable temporal logic constraints. These are efficiently processed at run-time to produce the desired behavior in a distributed manner. We have implemented the above modules in a (concurrent) actor language.

Contents

1 Introduction

Multiagent systems are finding important applications over the expanding computing and communications infrastructure [Wittig, 1992; Liu & Sycara, 1994; Oates *et al.*, 1994]. The chief characteristics of modern information environments are distribution, heterogeneity, and autonomy. Multiagent systems apply naturally in such settings, and help convert the above necessities into the virtue of modularity.

We propose *interaction-oriented programming* as a class of formalisms and techniques to develop multiagent systems. Interaction-oriented programming (IOP) focuses on what is between, rather than within, agents. Briefly, IOP is concerned with

- the semantics of interactions among distributed agents
- languages for expressing the desired properties of interactions
- techniques for programming systems
- tools for realizing them.

IOP is a family of formalisms and techniques, with a wide range of abstractions. These include abstractions for (a) a rigorous understanding of events in a multiagent system, (b) message passing to implement control and data flow [Hewitt, 1991], (c) patterns of interactions, (d) knowledge-level communication constraints [Singh, 1994], and (e) social constructs [Gasser, 1991]. Much research has been conducted into these abstractions. What is new here is the focus on programmability and common structure or "design-patterns" without, however, any loss of rigor. This is a significant challenge. For this reason, we are able to report only on the first three aspects here. In this sense, this paper only scratches the surface of IOP. However, nontrivial progress has been made on a formal theory based on process algebra and a successful implementation in an actor language.

IOP shamelessly borrows from previous work in concurrent programming, distributed computing, heterogeneous databases, and distributed artificial intelligence. Thus, IOP naturally bridges the gap between the open systems and intelligent agent heritage of multiagent systems [Hewitt, 1991; Hewitt & Inman, 1991]. The most relevant previous techniques for developing and verifying multiagent systems are either not formal, or are designed for traditional distributed systems, or do not fully exploit the modularity inherent in multiagent systems. For example, AgenTalk [Kuwabara *et al.*, 1995] gives a powerful programming environment, but no formal semantics. By contrast, [Kick, 1995] has a theory, but no associated implementation. Further, Kick's theory is a variation on traditional process algebras and violates autonomy by requiring full details of the internal structure of agents. The present approach develops a semantics and realizes it in an activity management infrastructure that enhances ideas from extended database transaction models.

This paper presents our theory and implementation in the context of cooperative information searches. Section 2 motivates and presents our conceptual approach. Section 3 describes our algebra for specifying interactions and uses it to formalize an example from section 2. Section 4 reviews the pertinent multiagent literature.

2 Conceptual Description of Approach

Although our approach is generic, we consider information search applications for concreteness. In such applications, agents cooperate to perform combinations of tasks such as resource discovery, information retrieval, information filtering, querying heterogeneous databases, and information fusion.

Example 1 Consider a ship on the high seas. Suppose an engine spare-part, a valve, runs low in the ship's inventory. This simple fact can lead the maintenance engineer to a complex search for information. (a) Have additional spares already been ordered?—access an on-board text log. (b) Are any in transit—access an on-shore legacy database. (c) Are such valves available at the next sea-port to be visited?—access the bridge to find the next sea-port and call up the databases of the part suppliers located there. (d) Can other available valves be substituted for this valve?—bring up the engine manual, the online spec-sheets on the given valve and other likely valves, and view animations of how to install them. Ideally, for (c) and (d) we should conduct the searches concurrently while protecting the user from extraneous detail. ∎

Roughly, information search is to open information systems what queries are to databases. For reasons such as heterogeneity, autonomy, dynamism, and so on, information searches must be extensible and flexible. Agents can be a useful means to satisfy these requirements. Unfortunately, searches—agent-based or not—in open information systems are often performed in an unprincipled and rigid manner. The interactions among the agents or modules are hardwired.

2.1 Patterns of Interaction

Such procedural encodings are not appropriate for a number of reasons. They are difficult to specify, modify, reason about, optimize, and build tools for. Therefore, a semantically perspicuous declarative representation of searches and a means to execute them is required. Our interest here is not in the application-specific representations (although those are obviously essential), but in the *patterns of interaction.* We define search paradigms as formalized classes of searches, or the patterns of interaction that arise in cooperative search.

Example 2 The searches of Example 1 fall into three paradigms. Searches (a) and (b) involve simple queries. Search (c) involves a query to the bridge, a directory lookup to find suppliers in the next port, and a concurrent map over the list of suppliers to ask about the desired valve. One positive response is enough, but additional responses improve reliability and help optimize other criteria, e.g., the price. Search (d) involves merging information from heterogeneous sources. ∎

What do the patterns buy for us? Although hardwired approaches can work, in open information environments, a flexible declarative mechanism for logically specifying interactions is required. Such a mechanism enables extracting the key properties

of classes of computations and executing them in a changing environment, potentially generating a different schedule each time, but preserving the semantics.

Example 3 Continuing with search (c) of Example 1, if the next port is several hours away and the directory of valve suppliers is unresponsive, it might be OK to wait and retry. Here speed is not a consideration but recall, i.e., finding as many suppliers as one can, is. ∎

Search paradigms have several advantages by virtue of being formal and declarative. They (1) enable reasoning about correctness of classes of interactions; (2) facilitate concurrency and efficiency improvements through algebraic and other reasoning about the meanings of the requests—to rewrite them or to effectively merge results (e.g., to perform cleanups like removing duplicates); and (3) enable learning and adaptation (e.g., as a consequence of iterative refinement of requests).

Clearly, a variety of patterns are needed to handle the interesting cases, partly because interactions may have varying effectiveness depending on the application and the underlying computing and communications infrastructure. A generic declarative facility to specify and schedule interactions would obviate designing specialized systems for each of them. This motivates our approach, which is to (a) develop a formal language and semantics for specifications of interactions and (b) automatically schedule computations that respect the desired interactions.

2.2 Architecture and Execution Model

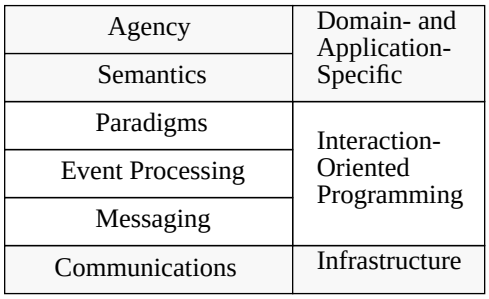

Figure 1: Layering of functionality and focus

We propose a layered architecture as shown in Figure 1. At the bottom is the infrastructure, which we assume. At the top are the application-specific multiagent systems, which we wish to build. In between are the IOP layers of the interaction specification and management. They include the patterns of interaction, their translation into events, and their scheduling through appropriate messages.

Each agent deals with the *interaction manager* in terms of *events* that are significant for coordination (discussed in Example 4 below). Figure 2 shows how the manager interacts with agents. The agents tell it what events have happened unilaterally and ask permission for those it may control; it gives or denies permissions,

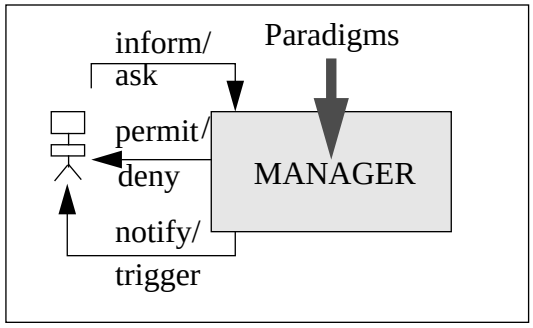

Figure 2: Execution model, logically

sends notifications, or triggers more events. Any necessary reasoning on intermediate results for decision-making is carried out through application-specific subcomputations. Thus the manager handles only the coordinational aspects of the search, which simplifies its design and implementation considerably.

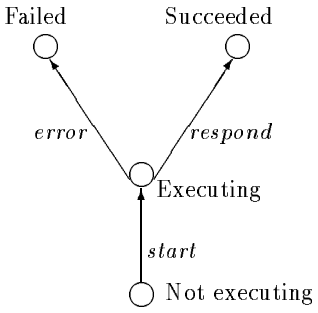

Figure 3: Skeleton for a simple querying agent

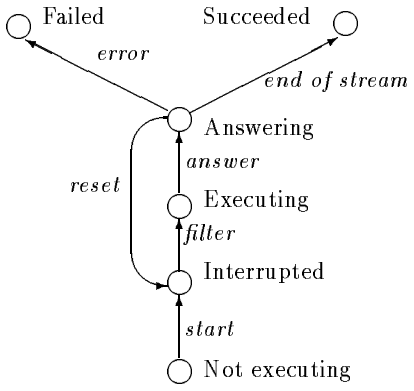

Figure 4: Skeleton for an information filtering agent

The manager is not tied to any specific set of interactions, but interprets declarative specifications of them. The specifications are abstract and involve only the

significant events of the contributing agents. The *skeleton* of an agent describes its major transitions for the purposes of coordination—it identifies the events in the agent that are exposed.

Example 4 Figures 3 and 4 show some skeletons that arise in information search. The significant events for the first skeleton are *start*, *error*, and *respond*—all the computation takes place in the node labeled "Executing." The significant events for the second skeleton are obvious—the key difference is that it can iterate over some of the events. ∎

Our approach makes no assumptions about the possible skeletons. In this way, it is more general than previous approaches [Breitbart *et al.*, 1993; Chrysanthis & Ramamritham, 1994], which are limited to single-shot transactions.

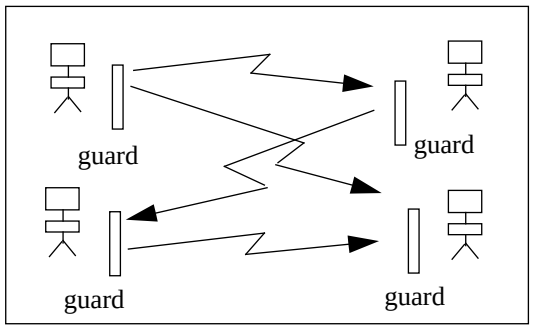

Figure 5: Execution model as implemented

Importantly, the manager is not a separate entity, but is distributed across the *guards* on the significant events of each agent. Figure 5 shows how the guards exchange messages; the message content and direction are automatically compiled.

3 Formalization of Patterns of Interactions

The formalization of interactions turns out to be quite simple. This is because it involves a straightforward reinterpretation of the mathematical theory we previously developed for relaxed transaction specification and scheduling. The mathematical results are simply inherited. Our theory and implementation are based on an abstract event-based trace semantics. Our algebraic notation, its semantics, and its processing are described in [Singh, 1996b; Singh, 1996c]; we include only a summary here.

Our notation involves event *types*, which are explicitly represented in our implementation. Specific event *tokens*, however, are attempted or triggered. Event tokens are instantiated from event types through parametrization—a tuple of all relevant parameters is included in each token. When dependencies are stated, some of the parameters can be variables, which are implicitly universally quantified. Section 3.1 describes the formal language; section 3.2 gives the highlights of its semantics, and section 3.3 applies it on Example 1.

3.1 Formal Language

$\mathcal{E}$, the language of event expressions has the following syntax. Σ is the set of significant event symbols. Γ includes all event literals or constants and Ξ includes all event atoms (constants and variables). A *dependency* or an expression is a member of $\mathcal{E}$. An *interaction* is a set of dependencies. Let $\delta(e)$ give the number of parameters of e. Here we assume a set $\mathcal{V}$ of variables and a set $\mathcal{C}$ constants that can be used as parameters.

Syntax 1 $e \in \Sigma$, and $p_1, \ldots, p_{\delta(e)} \in \mathcal{C}$ implies $e[p_1 \ldots p_{\delta(e)}], \overline{e}[p_1 \ldots p_{\delta(e)}] \in \Gamma$

Syntax 2 $e \in \Sigma$, and $p_1, \ldots, p_{\delta(e)} \in (\mathcal{V} \cup \mathcal{C})$ implies $e[p_1 \ldots p_{\delta(e)}], \overline{e}[p_1 \ldots p_{\delta(e)}] \in \Xi$

Syntax 3 $\Gamma, \Xi \subseteq \mathcal{E}$

Syntax 4 $0, \top \in \mathcal{E}$

Syntax 5 $E_1, E_2 \in \mathcal{E}$ implies that $E_1 \cdot E_2, E_1 \vee E_2, E_1 \wedge E_2 \in \mathcal{E}$

The specification $e[p_1 \ldots p_m] \in \Gamma$ means that e occurs instantiated with constant parameters $[p_1 \ldots p_m]$ ($\overline{e}$ means that the complement of e occurs). The constant 0 refers to a specification that is always false; $\top$ refers to one that is always true. The operator $\vee$ means disjunction. The operator $\wedge$ means conjunction; $\wedge$ thus refers to the interleaving of its arguments. The operator $\cdot$ refers to sequencing of its arguments. Variable parameters are treated as implicitly universally quantified. We follow the convention that $\cdot$ has precedence over $\vee$ and $\wedge$, and $\wedge$ has precedence over $\vee$.

3.2 Formal Semantics and Symbolic Reasoning

Our formal semantics of $\mathcal{E}$ is based on traces, but unlike traditional process algebras, also has a notion of *admissibility*. Admissibility encodes the knowledge of the system about specific events. We give a set of equations, which enable efficient symbolic reasoning to determine when a certain event may be permitted, prevented, or triggered. These equations are proved sound and complete in [Singh, 1996b]. Certain compile-time preprocessing is required to prevent deadlocks and race conditions. This too is carried out symbolically. We lack the space to include details.

3.3 Specification and Scheduling of Interactions

Our language allows a wide variety of dependencies to be stated. Table 1 shows some common dependencies, which are explained next. It assumes two computations U and V, whose appropriate events are u and v with parameters υ and ν, respectively (each could be a tuple). s denotes the start event.

1. U is required by V. If $v[\nu]$ occurs, then $u[\upsilon]$ must occur before or after $v[\nu]$.

Informal Relationship	Dependency
U is required by V	$u[v] \vee \overline{v}[\nu]$
U disables V	$\overline{u}[v] \vee \overline{v}[\nu] \vee v[\nu] \cdot u[v]$
U feeds V	$u[v] \cdot v[\nu] \vee \overline{v}[\nu]$
U conditionally feeds V	$\overline{s}[v] \vee u[v] \cdot v[\nu] \vee \overline{v}[\nu]$
Guaranteeing U enables V	$u[v] \wedge v[\nu] \vee \overline{v}[\nu]$
U initiates V	$\overline{u}[v] \wedge \overline{v}[\nu] \vee u[v] \cdot v[\nu]$

Table 1: Some typical dependencies

2. U disables V. If $u[v]$ occurs, then $v[\nu]$ must occur before $u[v]$.
3. U feeds V. $v[\nu]$ requires $u[v]$ to occur before, but $v[\nu]$ does not have to occur if $u[v]$ does. This suggests an enabling condition or a data flow from U to V.
4. U conditionally feeds V. If U starts, then it feeds V.
5. Guaranteeing U enables V. $v[\nu]$ can occur only if $u[v]$ has or will occur.
6. U initiates V. $v[\nu]$ occurs iff $u[v]$ precedes it.

The dependencies are defined with parameters so that the appropriate connections among activities may be effected without restricting the behavior of the system. These dependencies suffice for following examples, but more complex, multiparty dependencies are supported for other cases.

Example 5 We now formalize search (c) of Example 2. This can be formalized in a number of ways yielding different characteristics. Assume five types of subqueries: B to the bridge, D a directory lookup, Q the main queries, M to map over the responses of D, and F to fuse the results. We consider two formalizations.

Here x denotes the unique id of the information search through which the various instantiations of the relevant computations are tied together. *tup* is a variable bound to a tuple, which is processed by M. *sup* is a variable bound to a supplier. v indicates the availability of the desired valve. Subscripts r, s, a, and f respectively denote the *response*, *start*, *answer*, and *filter* events in the given skeletons.

- Assume all subcomputations except M have skeletons as in Figure 3 with D returning a tuple response containing a list of suppliers and Q being invoked on each of its members. M has a skeleton as in Figure 4. (D1.1) $B_r[x\ port]$ feeds $D_s[x\ port]$; (D1.2) $D_r[x\ tup]$ feeds $M_s[x\ tup]$; (D1.3) $M_a[x\ sup]$ initiates $Q_s[x\ sup]$; (D1.4) $Q_r[x\ sup\ v]$ conditionally feeds $F_s[x]$. (For expository ease, we don't consider the possibility that the disjunction of Q_r and Q_e (error) should enable F_s.)

  The response from the bridge feeds the directory lookup; the response from that goes into the mapper, which creates a query for each supplier. The responses are all collected into the fuser of which there is only one instance per search. This is why the fuser is parametrized only with x, the parameter of the whole search.

- Assume that B and F have the skeleton of Figure 3 and D and Q of Figure 4. There is no separate mapper. (D2.1) $B_r[x\ port]$ feeds $D_s[x\ port]$; (D2.2) $D_a[x\ sup]$ feeds $Q_f[x\ sup]$; (D2.3) $Q_a[x\ sup\ v]$ conditionally feeds $F_s[x]$.

  Now the directory lookup produces results one by one, which are used by the query task to initiate queries for each supplier. The results are fed into the fuser as before.

Both formalizations specify the structural properties of the search. ▌

Example 6 We now discuss the scheduling of the searches of Example 5. Although the two formalizations are similar, they have different scheduling properties.

- When the search is initiated, B is invoked. It returns with a port name, which is used in D, which could already have been set up, but would be waiting. D produces a tuple of suppliers. For each of these a separate query is initiated, which goes to the given supplier's database. These queries execute concurrently. However, the fuser waits until all the queries terminate.
- B feeds into D as before. However, D releases its answers piecemeal and they are fed straight into Q. The queries are executed one at a time. The fuser waits until the last one terminates.

The first search requires the entire tuple of suppliers to be computed before the next subquery can begin. By contrast, the second serializes the queries to the different suppliers. The good points of both approaches can readily be merged. Certain variations of fusion strategies (e.g., return one solution) can be encoded directly into dependencies. Other fusion strategies can be captured by adding a task that shields the fuser from the results that are input to it. ▌

The above examples show how interesting search paradigms can be declaratively captured and scheduled in a generic manner. This approach hones in on the structure of the computations by avoiding low-level details, which are encoded separately. This highlights the optimizations that can be effected through choosing a paradigm carefully to suit one's application needs and infrastructure availability.

3.4 Coordination and Negotiation

We saw how IOP applies to information applications. IOP also applies well to higher-level coordination protocols, such as the contract net [Davis & Smith, 1983]. Briefly, the contract net begins when the manager sends out a request for proposals (RFP); some potential contractors respond with bids; the manager accepts one of the bids and awards the task. Much of the required reasoning is application-specific, e.g., who to send the RFP to, whether to bid, and how to evaluate bids.

Example 7 Since all agents can play the role of manager or contractor, we assume that all of them have the same significant events. Any agent because of internal reasons can perform the $A_{rfp}[a\ t\ c]$ event. Here a is the agent id, t is the task id, and c is a potential contractor—there will be a separate event for each c. This involves a dependency (D3.1) $A_{rfp}[a\ t\ c\ info]$ initiates $A_{think}[c\ t\ a\ info]$. The receiving agents think about the RFP and autonomously decide to bid or not bid. If not, they exit the protocol. Otherwise, the following dependency kicks in: (D3.2) $A_{bid}[c\ t\ a\ bid]$ conditionally feeds $A_{eval}[a\ t\ c\ bid]$. The manager now autonomously evaluates bids, leading to an award on one of them, which triggers the work: (D3.3) $A_{award}[a\ t\ c\ task]$ initiates $A_{work}[c\ t\ a\ task]$. Because of internal reasons, this can lead to further RFP events and thus the whole procedure can repeat. ∎

Other formalizations are also possible. The contract net involves fairly simple interactions. More interesting protocols can be designed in which two parties each wait for the other to proceed, or assume that the other will not proceed. Our approach can detect and preprocess such protocols, so that additional "promissory" or "prohibitory" messages can be generated at run-time to obtain the desired behavior.

4 Discussion

Much present work on information search concentrates on techniques for creating, maintaining, and using indexes, but in a hardwired manner, e.g., [Bowman *et al.*, 1994]. Many "agent" approaches, e.g., [White, 1994; Borenstein, 1994], provide scripting languages through which agents can be transported, but provide no significant abstractions to program agents or structure their computations. The distributed AI approaches are the most promising. [Oates *et al.*, 1994] propose an approach for planning searches. However, their approach does not have an explicit representation of search paradigms, and does not apply generically. The search techniques are captured as different search paradigms in our approach. Some of the other agent techniques have focused on ontologies [Arens *et al.*, 1993; Huhns *et al.*, 1994] and knowledge communication [Labrou & Finin, 1994]. Our present contribution is orthogonal to, and compatible with, them.

High-level abstractions for agents have been intensively studied [Shoham, 1993; Rao & Georgeff, 1991; Cohen & Levesque, 1990; Singh, 1994]. These approaches develop formal semantics, but do not give as precise an operational characterization of the computations involved. The present work has a formal semantics along with an operational interpretation, but needs to be enhanced with some of the abstractions from the above works.

Some previous research has related communications to the agents' internal specifications [McCarthy, 1991; Singh, 1994], but not to the extent necessary to define and enact reusable patterns of interactions. Formal research on interactions among agents includes [Singh, 1991; Haddadi, 1995]. Other highly valuable work on coordination includes [Decker & Lesser, 1995; von Martial, 1992], which however is not formal.

[Rosenschein & Zlotkin, 1994] develop powerful game-theoretic approaches for negotiations among agents. We see the above research as complementary to the present paper for the following reason. These approaches have developed powerful semantic representations of domains, plans, and interactions, which can capture many of the intricacies of complex applications. Our activity management infrastructure can facilitate the above approaches and use them to provide the higher-level meanings.

There has been much work on social abstractions for agents [Castelfranchi, 1993; Gasser, 1991; Singh, 1996a], but a careful computational analysis of it is still awaited. We believe that the present infrastructure will facilitate the development of a computational treatment of the social constructs by capturing the mechanics of possible interactions in a succinct manner.

To incorporate the above sophisticated concepts of communications and social interactions are our main challenges in future work.

5 Conclusions

Open information environments require solutions that marry AI and traditional techniques. Multiagent systems are a result of this marriage. Our approach adapts techniques from formal methods in database activity management for the interaction-oriented programming of multiagent systems. We emphasize, however, that a plain traditional approach would not be as effective here. Traditional formal approaches preclude encapsulation of the component computations as agents; they do not accommodate the notion of admissibility, which captures the knowledge of the scheduler; they (in the case of databases) are limited to single-shot transactions and not applicable to arbitrary, nonterminating, complex computations that characterize agents. Design patterns are drawing much attention in object-oriented programming, but they are not formalized and focus on program structure rather than interactions [Pree, 1995]. Closer connections remain to be investigated.

Although IOP is a generic approach, the applications are of course highly important. For search applications, we are building a case-law of examples and search paradigms to be categorized using features such as (1) the cost, urgency, precision, recall, and delay requirements, (2) properties of the available resources, and (3) the organizational and autonomy characteristics of the agents. This categorization will yield a "predictive model" of when a given paradigm is most appropriate. Another important class of applications involves workflows. Workflows perform business functions and maintain consistency among heterogeneous databases. Information searches often involve *relevance* and *meaningfulness* constraints on the results, rather than *consistency*. However, the structure of computations in both cases is readily captured by IOP. A generic approach is not only easier to use, it also enables application-specific experimentation.

Our main future work is the elaboration of IOP to accommodate more sophisticated kinds of interaction, based on communication constraints and social commitments, so we can ultimately remove the word "Toward" from the title!

References

[Arens *et al.*, 1993] Arens, Yigal; Chee, Chin Y.; Hsu, Chun-Nan; and Knoblock, Craig A.; 1993. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems* 2(3):127–158.

[Borenstein, 1994] Borenstein, Nathaniel S.; 1994. Email with a mind of its own: The Safe-Tcl language for enabled email. In *Proceedings of the Workshop on Upper Layer Protocols and Architectures (ULPAA).*

[Bowman *et al.*, 1994] Bowman, C. Mic; Danzig, Peter B.; Manber, Udi; and Schwartz, Michael F.; 1994. Scalable internet resource discovery: Research problems and approaches. *Communications of the ACM* 37(8):98–114.

[Breitbart *et al.*, 1993] Breitbart, Y.; Deacon, A.; Schek, H.-J.; Sheth, A.; and Weikum, G.; 1993. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *SIGMOD Record* 22(3).

[Castelfranchi, 1993] Castelfranchi, Cristiano; 1993. Commitments: From individual intentions to groups and organizations. In *Proceedings of the AAAI-93 Workshop on AI and Theories of Groups and Organizations: Conceptual and Empirical Research.*

[Chrysanthis & Ramamritham, 1994] Chrysanthis, Panos K. and Ramamritham, Krithi; 1994. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems* 19(3):450–491.

[Cohen & Levesque, 1990] Cohen, Philip R. and Levesque, Hector J.; 1990. Intention is choice with commitment. *Artificial Intelligence* 42:213–261.

[Davis & Smith, 1983] Davis, Randall and Smith, Reid G.; 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 20:63–109. Reprinted in *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser, eds., Morgan Kaufmann, 1988.

[Decker & Lesser, 1995] Decker, Keith S. and Lesser, Victor R.; 1995. Designing a family of coordination algorithms. In *Proceedings of the International Conference on Multiagent Systems.*

[Gasser, 1991] Gasser, Les; 1991. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence* 47:107–138.

[Haddadi, 1995] Haddadi, Afsaneh; 1995. Towards a pragmatic theory of interactions. In *Proceedings of the International Conference on Multiagent Systems.*

[Hewitt & Inman, 1991] Hewitt, Carl and Inman, Jeff; 1991. DAI betwixt and between: From "intelligent agents" to open systems science. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6):1409–1419.

[Hewitt, 1991] Hewitt, Carl; 1991. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence* 47:79–106.

[Huhns *et al.*, 1994] Huhns, Michael N.; Singh, Munindar P.; Ksiezyk, Tomasz; and Jacobs, Nigel; 1994. Global information management via local autonomous agents. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence.*

[Kick, 1995] Kick, Alexander; 1995. An agent algebra for the formal description and verification of multi agent systems. In *Proceedings of the International Conference on Multiagent Systems.*

[Kuwabara *et al.*, 1995] Kuwabara, Kazuhiro; Ishida, Toru; and Osato, Nobuyasu; 1995. AgenTalk: Coordination protocol description for multiagent systems. In *Proceedings of the International Conference on Multiagent Systems.*

[Labrou & Finin, 1994] Labrou, Yannis and Finin, Tim; 1994. A semantics approach for KQML—a general purpose communication language for software agents. In *Proceedings of the International Conference on Information and Knowledge Management.*

[Liu & Sycara, 1994] Liu, JyiShane and Sycara, Katia; 1994. Distributed problem solving through coordination in a society of agents. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence.*

[McCarthy, 1991] McCarthy, John; 1991. Elephant 2000: A programming language based on speech acts. Computer Science Department, Stanford University.

[Oates *et al.*, 1994] Oates, Tim; Prasad, M. V. Nagendra; and Lesser, Victor R.; 1994. Cooperative information gathering: A distributed problem solving approach. Technical Report 94-66, University of Massachusetts, Amherst, MA.

[Pree, 1995] Pree, Wolfgang; 1995. *Design Patters for Object-Oriented Software Development.* ACM Press and Addison-Wesley, Reading, MA.

[Rao & Georgeff, 1991] Rao, Anand S. and Georgeff, Michael P.; 1991. Modeling rational agents within a BDI-architecture. In *Principles of Knowledge Representation and Reasoning.*

[Rosenschein & Zlotkin, 1994] Rosenschein, Jeffrey S. and Zlotkin, Gilad; 1994. *Rules of Encounter.* MIT Press, Cambridge, MA.

[Shoham, 1993] Shoham, Yoav; 1993. Agent-oriented programming. *Artificial Intelligence* 60:51–92.

[Singh, 1991] Singh, Munindar P.; 1991. Group ability and structure. In Demazeau, Y. and Müller, J.-P., editors, *Decentralized Artificial Intelligence, Volume 2.* Elsevier Science Publishers B.V. / North-Holland, Amsterdam, Holland. 127–145.

[Singh, 1994] Singh, Munindar P.; 1994. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications.* Springer Verlag, Heidelberg, Germany.

[Singh, 1995] Singh, Munindar P.; 1995. Semantical considerations on workflows: Algebraically specifying and scheduling intertask dependencies. In *Proceedings of the 5th International Workshop on Database Programming Languages (DBPL).*

[Singh, 1996a] Singh, Munindar P.; 1996a. A conceptual analysis of commitments in multiagent systems. Technical Report TR-96-09, Department of Computer Science, North Carolina State University, Raleigh, NC. Available at `http://www4.ncsu.edu/eos/info/ dblab/www/mpsingh/ papers/agents+multiagents/ commit.ps`.

[Singh, 1996b] Singh, Munindar P.; 1996b. Formal semantics for workflow computations. Technical Report TR-96-08, Department of Computer Science, North Carolina State University, Raleigh, NC. Extended version of [Singh, 1995]. Also available at `http://www4.ncsu.edu/eos/info/ dblab/www/mpsingh/ papers/databases/ semantics-jan96.ps`.

[Singh, 1996c] Singh, Munindar P.; 1996c. Synthesizing distributed constrained events from transactional workflow specifications. In *Proceedings of the 12th International Conference on Data Engineering (ICDE).*

[von Martial, 1992] von Martial, Frank; 1992. *Coordinating Plans of Autonomous Agents.* Springer Verlag, Berlin, Germany.

[White, 1994] White, James; 1994. TeleScript technology: The foundation for the electronic marketplace. White paper.

[Wittig, 1992] Wittig, Thies, editor. *ARCHON: An Architecture for Multi-agent Systems.* Ellis Horwood Limited, West Sussex, UK.