AN AGENT MODEL FOR DISTRIBUTED PART-SELECTION

Timothy P. Darr and William P. Birmingham
Advanced Technologies Lab
Department of Electrical Engineering and Computer Science
The University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109
{timd, wpb}@eecs.umich.edu

Abstract

This report presents the agent model for a set of catalog, constraint and system agents. For each type of agent, the model defines the goals it can achieve; messages sent agents to other agents requesting that they achieve some goal; the commitments that they make, internally, to achieve goals in response to the messages; and messages that it sends once a goal is achieved. The messages that an agent sends and receives are intentional and are organized into protocols that define the type, sequence and content of messages exchanged among agents to achieve goals.

1.1 Agent Model Definitions

This section defines the elements of the agent model, which is based loosely on the agent-oriented programming (AOP) paradigm (Shoham 1993) consisting of agent actions, beliefs, obligations, and decisions[1]. At any time t, an agent can receive a message from another agent. Receiving message m at time t is represented by the predicate recv(m, t). For each message, there is a goal to achieve, expressed as the predicate can(m, g), where m is a message, and g is the goal.

In this model, agents wait until a set of messages are received before achieving a goal. We define τ as the (variable) time interval that agents wait until achieving some goal. Variability of the length of the time interval arises because messages are received in the context of achieving some goal, and because there is a random, finite delay in transmitting a message from one agent to another.

The set of goals to achieve as determined by the messages received during the interval τ is given by $G_\tau = \{g: \text{recv}(m, t) \wedge \text{can}(m, g) \wedge \text{able}(g, t), \text{ for all } t \in \tau\}$, where able(g, t) means that the agent has the resources to achieve goal g at time t. At the end of the time interval τ, an agent may have more than one goal in the set G_τ, each of which cannot be achieved, so a decision-making component is used to commit to achieving one of the goals in G_τ. The decision-making component takes G_τ as input and produces a commitment to achieve a goal: Decision-Making(G_τ) $\rightarrow$ commit(g_κ, τ). Thus, agents reason about which goals to achieve at the end of the interval τ. Once

[1] In AOP, these are elements of the agent's mental state. The AOP mental state is not to be confused with the ACDS agent state, which is based on the part-selection domain. We borrow concepts from AOP for purposes of describing agent capability and making decisions about which action to take.

a commitment is made to achieve a goal, an algorithm is executed to achieve the goal. Table 1 summarizes the agent model notation.

Table 1: Agent Model Notation

Term	Description
recv(m, t)	Agent receives message m at time t.
can(m, g)	Agent can achieve goal g in response to message m.
able(g, t)	Agent has the resources to achieve goal g at time t.
G_τ	Set of goals under consideration during time τ.
commit(g, τ)	Agent commits to achieving goal g after interval τ.

Figure 1 illustrates the behavior of an agent during a time interval τ. Messages are received at times $t_1, t_2, ..., t_n$, which imply a set of goals G_τ to achieve. At the end of the interval τ, a decision-making component is invoked to commit to one of the goals in the set of goals under consideration. The selected goal, g_κ, is achieved by performing an algorithm at the end of the interval, then a new interval begins.

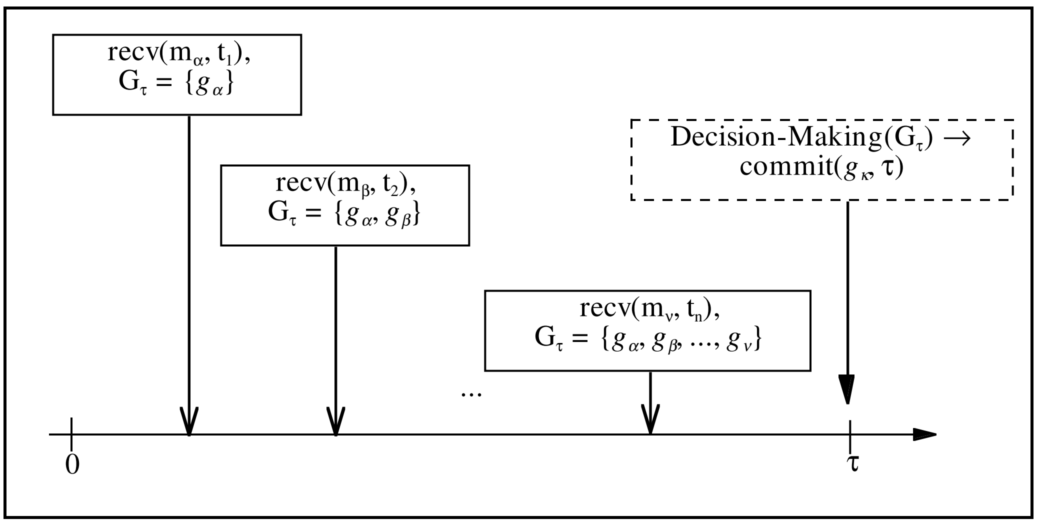

Figure 1: Synchronous Agent Commitment

Figure 2 shows the message processing and decision-making process within an agent. Messages enter the agent and map to goals to achieve using the can(m, g) predicate. Since there may be more than one goal to achieve, a the decision-making component selects one of the goals to achieve; this function is commit(g, τ). The goal is achieved using some algorithm, and a message or set of messages is returned to other agents.

In this report, we define a set of protocols for achieving the shared agent goals. The protocols prescribe the sequence of messages necessary to achieve a goal.

In the following sections, we expand each element of the agent model shown in Figure 2. The section “Agent Goals” (page 3) defines the agent goals. The section “Design Message Classes” (page 3) introduces the messages exchanged among agents. The section “Design Protocols” (page 4) concludes the report by defining the protocols that define the messages used to achieve the goals.

1.2 Agent Goals

In this agent model, goals achieve design properties[2]. Table 2 shows the mapping from properties to goals. In most of the cases, this mapping is obvious. We assume that the decomposable(ACDS-network) property requires goals to set up the network (catalog-agent respond-to-invite goal, system-agent create-network goal), and find a function-cover specification (catalog-agent remove-uncovered-parts goal).

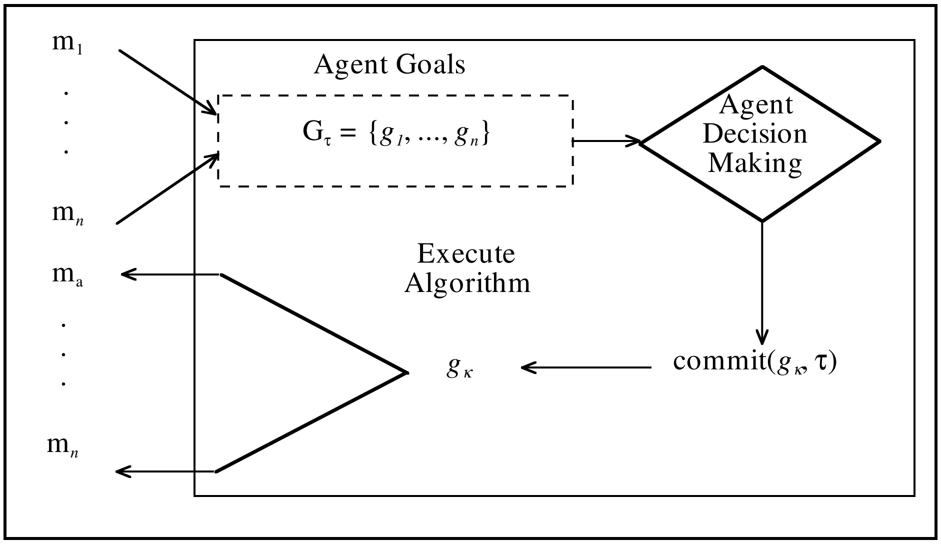

Figure 2: Agent Model

1.3 Design Message Classes

This section describes the message classes used in the design protocols defined below. These messages are based on speech-act theory, and are consistent with KQML (Neches, Fikes et al. 1991; Finin, Fritzson et al. 1992). Each message consists of a message class, which is interpreted as a request to achieve some goal, followed by a message content that provides information for achieving the goal[3].

Table 2: Agent Goals

Agent	Property	Goal
catalog	consistent(catalog)	make-catalog-consistent
catalog	decomposable(catalog)	partition-catalog
catalog	decomposable(ACDS-network)	respond-to-invite
catalog	decomposable(ACDS-network)	remove-uncovered-parts
catalog	decomposable(catalog)	report-solution
constraint	consistent(f_i(C))	create-consistent-space
constraint	decomposable(f_i(C))	create-suggested-space
system	decomposable(ACDS-network)	create-network
system	decomposable(ACDS-network)	form-solution

This section proposes three new message classes, which are extensions to KQML, specifically for distributed design: evaluate-state, assert-state and suggest-state (Darr 1996). The evaluate-state message is used by a catalog agent to request that a constraint agent evaluate an

[2] A definition of the agents and design properties can be found in (Darr 1996).

[3] (Darr 1996) describes the message-content format in detail.

attribute-space element for consistency or decomposability. Constraint agents respond to these requests using the assert-state and suggest state messages, respectively. Catalog agents use the assert-state and suggest-state messages to remove parts from their catalogs.

We use the syntax <message_name>:<content_description> to refer to a message, where <content_description> is a keyword that denotes the specific use of the message. The message content is a text string containing sets of ":keyword value" pairs, where the ':' denotes a keyword.

1.3.1 Evaluate-state

Catalog agents use the evaluate-state message to send elements of its attribute space to constraint agents for evaluation. The content of this message is a description of a subset of the agent's attributes using the attribute-space representation.

1.3.2 Assert-state and suggest-state

Constraint agents use the assert-state and suggest-state messages to provide direction to catalog agents in modifying their current space. The message content is an attribute-space element that defines the direction in which to move the catalog-agent state. How this content is used depends on the message.

1.3.3 Tell

Agents use the tell message to add knowledge to another agent's knowledge base. The system agent uses the tell:required-function message to send the required functions, contained in the content, to a catalog agent. The system agent uses the tell:covers message to send the function-cover specification, contained in the content, to a catalog agent. Catalog agents use the tell:accept and tell:reject messages to respond to requests to participate in a design. The content of the tell:accept message contains the catalog agent's function-cover multiplicities. Constraint agents use the tell:decomposable message to tell a catalog agent that its current state is decomposable. Catalog agents use the tell:solution message to send a part and part function cover to the system agent.

1.4 Design Protocols

This section defines a set of protocols that agents use to achieve goals. These protocols define the rules by which messages are exchanged. Table 3 lists the protocols and the messages they use.

Table 3: Protocol Goals and Messages

Protocol (figure)	Message
invitation (Figure 5)	tell:required-function tell:covers tell:accept tell:reject
design consistency (Figure 6)	evaluate-state:consistency assert-state
design decomposability (Figure 7)	evaluate-state:decomposability suggest-state:decomposability tell:decomposable
solution (Figure 8)	tell:solution

The invitation protocol (see section "Invitation" (page 5)) is used by the system agent to invite a set of catalog agents to participate in a design session. This protocol is similar to the Contract Net protocol (Davis and Smith 1983), with the distinction that it is instantiated specifically for design problems. The design-consistency protocol (see sections "Design Consistency" (page 6)) is based on the CSP concept of 2-consistency, and is used by both catalog and constraint agents to form a consistent design space. The design-decomposability protocol (see sections

“Design Decomposability” (page 7)) is used by a set of catalog and constraint agents to shrink a consistent space until it is decomposable. Finally, the solution protocol (see section “Solution” (page 7)) is used by a catalog agent to send parts to the system agent.

In the figures that follow defining the protocols, a state-transition graph representation is used with the conventions shown in Figure 3. This figure shows the sequence of messages sent and received from the perspective of a given agent. The unshaded circles represent a state in which the agent is sending a message to some other agent. The outgoing arrow(s) from this state is the message(s) sent by the agent. The shaded circles represent a state in which the agent receives a message. The outgoing arrow(s) from this state is the message(s) received by the agent. The double-lined circle is a termination state.

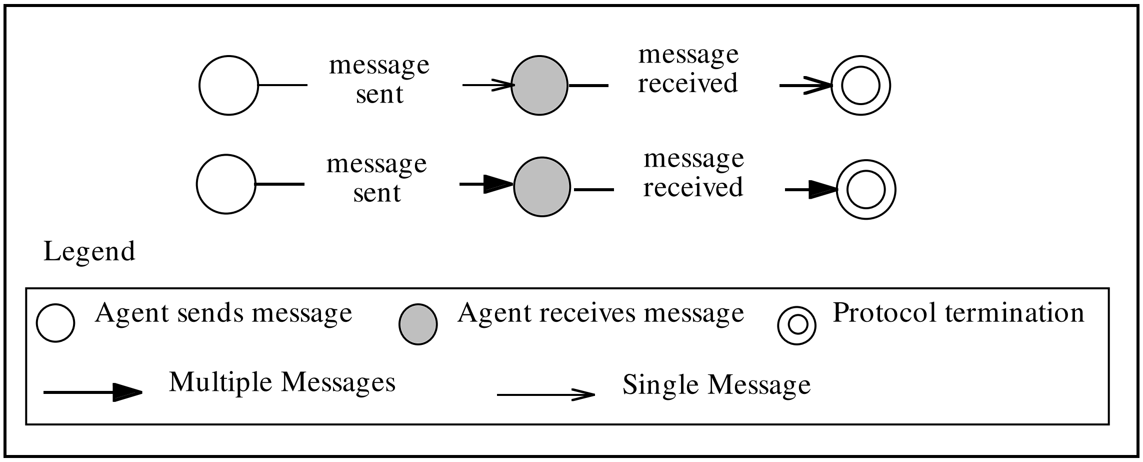

Figure 3: Protocol State Transition Diagram Convention

The protocols defined in this section, in combination with the predicates shown in Figure 4, associate messages with goals as outlined in the section “Agent Model Definitions” (page 1).

Catalog Agent
can(assert-state, make-catalog-consistent) can(suggest-state, partition-catalog) can(tell:required-functions, respond-to-invite) can(tell:covers, remove-uncovered-parts) can(tell:decomposable, report-solution)
Constraint Agent
can(evaluate-state:consistency, create-consistent-space) can(evaluate-state:decomposability, create-suggested-space)
System Agent
can(tell:accept, create-network) can(tell:reject, create-network) ca(tell:solution, form-solution)

Figure 4: Agent Capabilities

1.4.1 Invitation

The invitation protocol (Figure 5) defines the sequence of messages that the system agent uses to invite catalog agents to participate in a design. The system agent forms a list of the required functions and sends them to the invitee using the tell:req-function message. When the invitee receives the tell:req-function message, it checks to see if it contains any parts that implement the function. If the invitee contains such parts, it sends an accept message to the system agent. If it does not contain parts that implement the requested functions, it sends a reject message to the system agent.

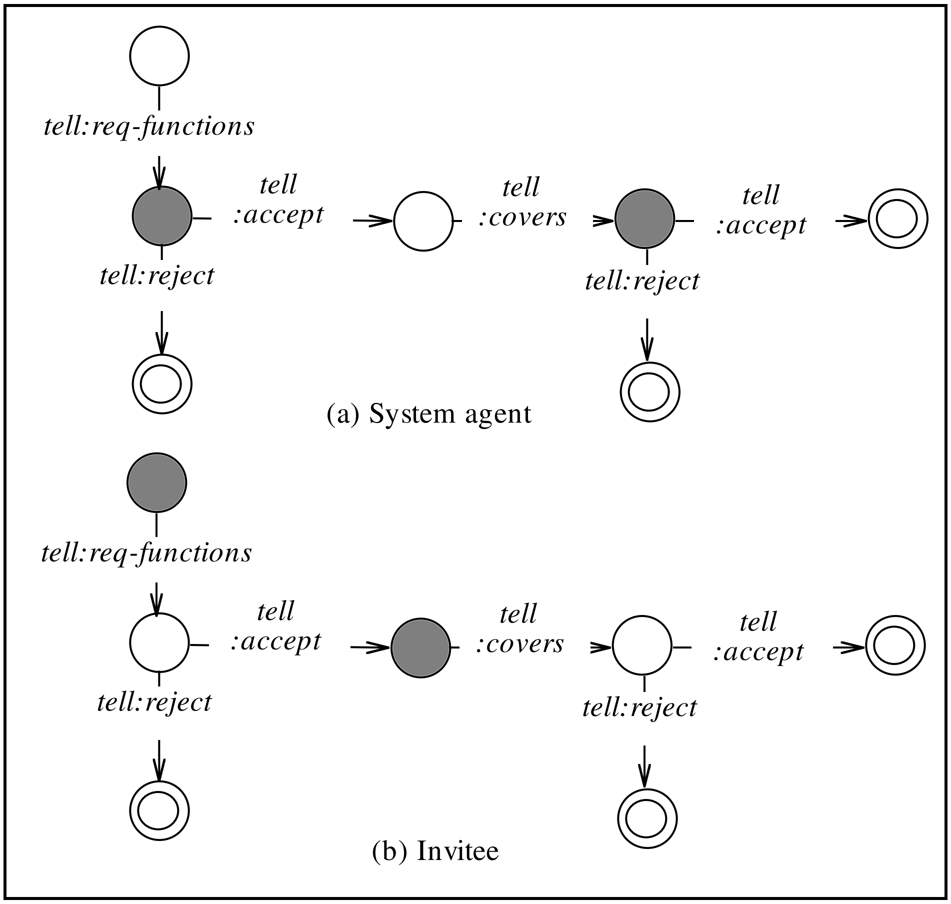

Figure 5: Invitation Protocol

If the invitee accepts the invitation, the system agent sends the function-cover specification to the target agent who removes parts whose part-function multiplicity does not match any of the covers in the function-cover specification. If parts remain after this step, the invitee accepts the invitation, otherwise it rejects the invitation.

1.4.2 Design Consistency

The design-consistency protocol (Figure 6) defines the sequence of messages, among a constraint and a set of catalog agents with attributes in the constraint's arguments, to create a consistent space. Catalog agents send their state to a constraint agent using the evaluate-state:consistency message (Figure 6(a)). The agents then wait until they receive the assert-state message. The catalog agents use the state described in the assert-state message to remove inconsistent parts from their catalogs. Within this protocol, catalog agents are required to remove the parts necessary to make their state consistent with the asserted state.

The constraint agent (Figure 6 (b)) waits until it receives at least one evaluate-state:consistency message from all the catalog agents. This ensures that the space is consistent before evaluating it for decomposability. Thus, this protocol captures part of the decision-making necessary to choose among goals; in this case, whether to achieve a consistent, or decomposable space. It uses the attribute-space elements in these messages to form a consistent state, which is then sent to the catalog agent using the assert-state message.

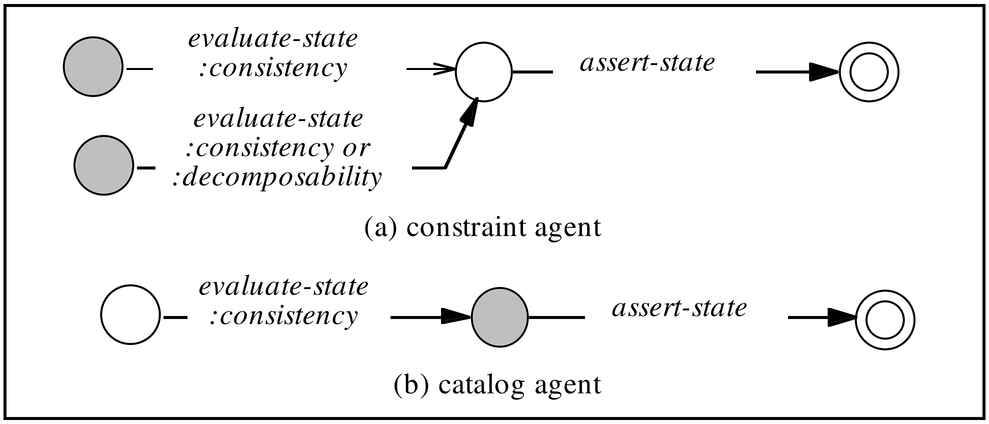

Figure 6: Design Consistency Protocol

1.4.3 Design Decomposability

The design-decomposability protocol (Figure 7) defines the sequence of messages, among a constraint and a set of catalog agents with attributes in the constraint's arguments, to achieve a decomposable state. Catalog agents send their state to a constraint agent using the evaluate-state:decomposability message (Figure 7(a)). The agent then waits until it receives the suggest-state or tell:decomposable message. The tell:decomposable message indicates that the most recent attribute-space element is decomposable with respect to the constraint that sent the message, and the protocol terminates. The suggest-state message indicates that the most recent attribute-space element is not decomposable, and includes suggestions of how the catalog agents can modify its state to move toward a decomposable state.

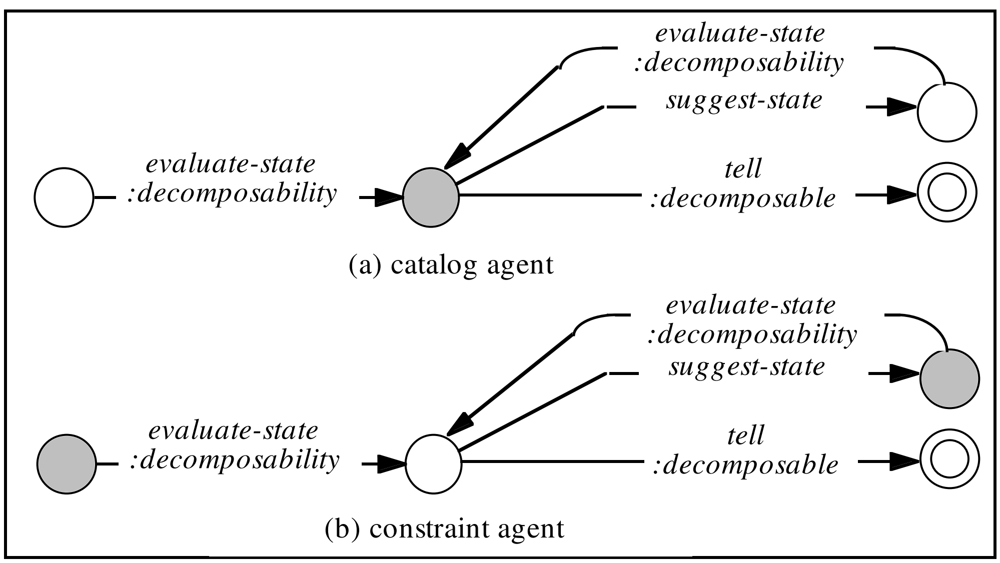

Figure 7: Design Decomposability Protocol

The constraint agent waits until the evaluate-state:decomposability message is received from each catalog agent (Figure 7(b)). It uses the attribute-space element information in these messages to determine if the attribute-space element is decomposable. If it is, the constraint agent sends the tell:decomposable message to the catalog agents and the protocol terminates. If the constraint is not decomposable, the agent sends the suggest-state message, including a description of how to modify the state to move toward decomposability, to the catalog agents.

1.4.4 Solution

The solution protocol (Figure 8) defines the sequence of messages to collect the parts in the final solution. Once in a decomposable state, each catalog agent selects a part and sends the part to the system agent using the tell:solution message, who adds the part to its knowledge base. If all functions are covered, the system agent sends the tell:design-done message to all catalog agents.

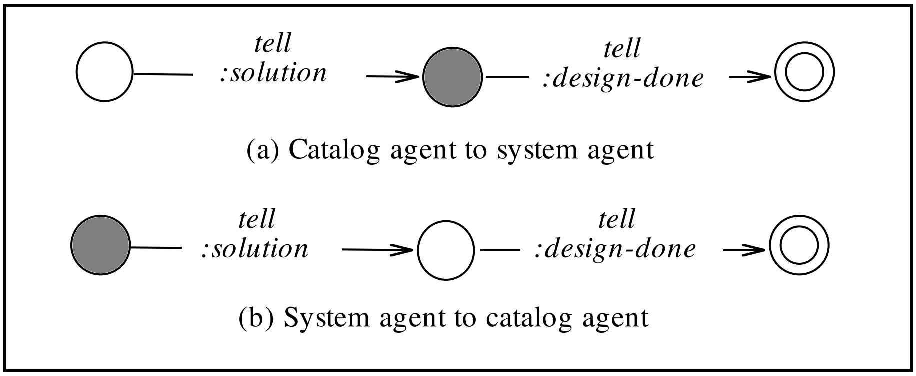

Figure 8: Solution Protocol

1.5 Bibliography

Darr, T. P. (1996). An Agent-Based Approach for Solving Distributed Part Selection Problems, Ph.D. Thesis, The University of Michigan.

Davis, R. and R. G. Smith (1983). "Negotiation as a metaphor for distributed problem solving." Artificial Intelligence **20**(1): 63-109.

Finin, T., R. Fritzson, et al. (1992). A Language and Protocol to Support Intelligent Agent Interoperability. Proceedings First Workshop on Enabling Technologies for Concurrent Engineering.

Neches, R., R. Fikes, et al. (1991). "Enabling Technology for Knowledge Sharing." AI Magazine **12**(3): 36-55.

Shoham, Y. (1993). "Agent-oriented programming." Artificial Intelligence **51**: 51-92.