Modeling Agent Conversations with Colored Petri Nets

R. Scott Cost, Ye Chen, Tim Finin, Yannis Labrou, Yun Peng
Laboratory for Advanced Information Technology
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, Maryland
cost@acm.org, yechen,finin,jklabrou,ypeng@cs.umbc.edu

Abstract

Conversations are a useful means of structuring communicative interactions among agents. The value of a conversation-based approach is largely determined by the conversational model it uses. Finite State Machines, used heavily to date for this purpose, are not sufficient for complex agent interactions requiring a notion of concurrency. We propose the use of Colored Petri Nets as a model underlying a language for conversation specification. This carries the relative simplicity and graphical representation of the former approach, along with greater expressive power and support for concurrency. The construction of such a language, Protolingua, is currently being investigated within the framework of the Jackal agent development environment.

1 Introduction

Conversations are a useful means of structuring communicative interactions among agents, by organizing messages into relevant contexts and providing a common guide to all parties. The value of a conversation-based approach is largely determined by the conversational model it uses. The presence of an underlying formal model supports the use of structured design techniques and formal analysis, greatly facilitating development, composition and reuse. Most conversation-modeling projects to date have used or extended finite state machines (FSM) in various ways, and for good reason. FSMs are simple, depict the flow of action/communication in an intuitive way, and are sufficient for many sequential interactions. However, they are not adequately expressive to model more complex interactions, especially those with some degree of concurrency. Colored Petri Nets (CPN) [9, 10, 11] are a well known and established model of concurrency, and can support the expression of a greater range of interaction. In addition, CPNs, like DFAs, have an intuitive graphical representation, are relatively simple to implement, and are accompanied by a variety of techniques and tools for formal analysis and design.

We have explored the use of model-based conversation specification in the context of multi agent system (MAS) support for manufacturing integration [17]. Agents in our system are constructed using the Jackal agent development platform [6], and communicate using the KQML agent communication language. Jackal, primarily a tool for communication, supports conversation-based message management through the use of abstract conversation specifications, which are interpreted relative to some appropriate model. Conversation specifications, or protocols, can describe anything from simple message/acknowledgment interactions to complex negotiations.

In the next section, we present a motivation for using conversations to organize agent interaction. Next, we present CPNs, the model we propose to use, in more detail. Following this, we discuss the implementation of these ideas in a real MAS framework. Finally, we present two examples of CPN use: the first, specification of a simple KQML register conversation, and the next, a more sophisticated negotiation interaction.

2 Conversation-Based Interaction Protocols

The study of agent communication languages (ACLs) is one of the pillars of current agent research. KQML and the FIPA ACL are the leading candidates as standards for specifying the encoding and transfer of messages among agents. While KQML is good for message-passing among agents, the message-passing level is not actually a very good one to exploit directly in building a system of cooperating agents. After all, when an agent sends a message, it has expectations about how the recipient will respond to the message. Those expectations are not encoded in the message itself; a higher-level structure must be used to encode them. The need for such conversation policies is increasingly recognized by the KQML community, and has been formally recognized in the latest FIPA draft standard [8, 7].

It is common in KQML-based systems to provide a message handler that examines the message performative to determine what action to take in response to the message. Such a method for handling incoming messages is adequate for very simple agents, but breaks down as the range of interactions in which an agent might participate increases. Missing from the traditional message-level processing is a notion of message context.

A notion growing in popularity is that the unit of communication between agents should be the conversation. A conversation is a pattern of message exchange that two (or more) agents agree to follow in communicating with one another. In effect, a conversation is a communications protocol, albeit one that may be initiated through negotiation, and may be short-lived relative to the way we are accustomed to thinking about protocols. A conversation lends context to the sending and receipt of messages, facilitating interpretation that is more meaningful. The adoption of conversation-based communication carries with it numerous advantages to the developer. There is a better fit with intuitive models of how agents will interact than is found in message-based communication. There is also a closer match to the way that network research approaches protocols, which allows both theoretical and practical results from that field to be applied to agent systems. Since conversation structure is separated from the actions to be taken by an agent engaged in the conversation, this facilitates the reuse of conversations in multiple contexts.

To date, little work has been devoted to the problem of conversation specification and implementation for mediated architectures. Strides must be taken in the toward facilitating the construction and reuse of conversations. An ontology of conversations and conversation libraries would advance this goal, as would solutions to the following questions:

1. Conversation specification: How can conversations best be described so that they are accessible both to people and to machines?

2. Conversation sharing: How can an agent use a conversation specification standard to describe the conversations in which it is willing to engage, and to learn what conversations are supported by other agents?

3. Conversation aggregation: How can sets of conversations be used as agent 'APIs' to describe classes of capabilities that define a particular service?

2.1 Conversation Specification

A specification of a conversation that could be shared among agents must contain several kinds of information about the conversation and about the agents that will use it. First, the sequence of messages must be specified. Traditionally, deterministic finite-state automata (DFAs) have been used for this purpose; DFAs can express a variety of behaviors while remaining conceptually simple. For more sophisticated interactions, however, it is desirable to use a formalism with more support for concurrency and verification. This is the motivation behind our investigation of CPNs as an alternative mechanism for more sophisticated conversation specification. Next, the set of roles that agents engaging in a conversation may play must be enumerated. Many conversations will be dialogues, and will specify just two roles; however conversations with more than two roles are equally important, representing the coordination of communication among several agents in pursuit of a single common goal.

DFAs and roles dictate the syntax of a conversation, but say nothing about the conversation's semantics. The ability of an agent to read a description of a conversation, then engage in such a conversation, demands that the description specify the conversation's semantics. To be useful though, such a specification must not rely on a full-blown, highly expressive knowledge representation language. We believe that a simple ontology of common goals and actions, together with a way to relate entries in the ontology to the roles, states, and transitions of the conversation specification, will be adequate for most purposes. This approach sacrifices expressiveness for simplicity and ease of implementation. It is nonetheless perfectly compatible with attempts to relate conversation policy to the semantics of underlying performatives, as proposed for example by [3].

These capabilities will allow the easy specification of individual conversations. To develop systems of conversations though, developers must have the ability to extend existing conversations through specialization and composition. Specialization is the ability to create new versions of a conversation that are more detailed than the original version; it is akin to the idea of subclassing in an object-oriented language. Composition is the ability to combine two conversations into a new, compound conversation. Development of these two capabilities will entail the creation of syntax for expressing a new conversation in terms of existing conversations, and for linking the appropriate pieces of the component conversations. It will also demand solution of a variety of technical problems, such as naming conflicts, and the merger of semantic descriptions of the conversations.

2.2 Conversation Sharing

A standardized conversation language, as proposed above, dictates how conversations will be represented; however, it does not say how such representations are shared among agents. While the details of how conversation sharing is accomplished are more mundane than those of conversation representation, they are nevertheless crucial to the viability of dynamic conversation-based systems. Three questions present themselves:

- How can an agent map from the name of a conversation to the specification of that conversation?
- How can one agent communicate to another the identity of the conversation it is using?
- How can an agent determine what conversations are handled by a service provider that does not yet know of the agent's interest?

2.3 Conversations Sets as APIs

The set of conversations in which an agent will participate defines an interface to that agent. Thus, standardized sets of conversations can serve as abstract agent interfaces (AAIs), in much the same way that standardized sets of function calls or method invocations serve as APIs in the traditional approach to system-building. That is, an interface to a particular class of service can be specified by identifying a collection of one or more conversations in which the provider of such a service agrees to participate. Any agent that wishes to provide this class of service need only implement the appropriate set of conversations. To be practical, a naming scheme will need to be developed for referring to such sets of conversations, and one or more agents will be needed to track the development and dissolution of particular AAIs. In addition to a mechanism for establishing and maintaining AAIs, standard roles and ontologies, applicable to a variety of applications, will need to be created.

There has been little work on communication languages from a practitioner's point of view. If we set aside work on network transport protocols or protocols in distributed computing (e.g., CORBA) as being too low-level for the purposes of intelligent agents, the remainder of the relevant research may be divided into two categories. The first deals with theoretical constructs and formalisms that address the issue of agency in general and communication in particular, as a dimension of agent behavior (e.g., AOP [19]). The second addresses agent languages and associated communication languages that have evolved somewhat to applications (e.g., TELESCRIPT [20]). In both cases, the bulk of the work on communication languages has been part of a broader project that commits to specific architectures.

Agent communication languages like KQML provide a much richer set of interaction primitives (e.g., KQML's performatives), support a richer set of communication protocols (e.g., point-to-point, brokering, recommending, broadcasting, multicasting, etc.), work with richer content languages (e.g., KIF), and are more readily extensible than any of the systems described above. However, as discussed above, KQML lacks organization at the conversation level that lends context to the messages it expresses and transmits. Limited work has been done on implementing conversations for software agents, and almost none has been done on expressing those conversations. As early as 1986, Winograd and Flores [21] used state transition diagrams to describe conversations. The COOL system [2] has perhaps the most detailed current finite state automata model to describe agent conversations. Each arc in a COOL state transition diagram represents a message transmission, a message receipt, or both. One consequence of this policy is that two different agents must use different automata to engage in the same conversation. COOL also uses an :intent slot to allow the recipient to decide which conversation structure to use in understanding the message. This is a simple way to express the semantics of the conversation, though it is not sufficient for sophisticated reasoning about and sharing of conversations.

Other conversation models that have been developed include those of Parunak [16], Chauhan [4], who uses COOL as the basis for his multi-agent development system, Kuwabara et al. [12], who add inheritance to conversations, Nodine and Unruh [15], who use conversation specifications to enforce correct conversational behavior by agents, Bradshaw [3], who introduces the notion of a conversation suite as a collection of commonly-used conversations known by many agents, and Labrou [14], who uses definite clause grammars to specify conversations. While each of these makes contributions to our general understanding of conversations, none show how descriptions of conversations might be shared by agents and used directly by them in implementing conversations.

2.4 Defining Common Agent Services via Conversations

A significant impediment to the development of agent systems is the lack of basic standard agent services that can be easily built on top of the conversation architecture. Examples of such services are: name and address resolution; authentication and security services; brokerage services; registration and group formation; message tracking and logging; communication and interaction; visualization; proxy services; auction services; workflow services; coordination services; and performance monitoring services. Services such as these have typically been implemented as needed in individual agent development environments. Two such examples are an agent name server and an intelligent broker.

3 Colored Petri Nets

Petri Nets (PN), or Place Transition Nets, are a well known formalism for modeling concurrency. A PN is a directed, connected, bipartite graph in which each node is either a *place* or a *transitions*. *Tokens* occupy *places*. When there is at least one *token* in every place connected to a *transition*, we say that transition is *enabled*. Any enabled *transition* may *fire*, removing one *token* from every input *place*, and depositing one *token* in each output *place*. Petri nets have been used extensively in the analysis of networks and concurrent systems. For a more complete introduction, see [1].

Colored Petri Nets (CPN) differ from PNs in one significant respect; *tokens* are not simply blank markers, but have data associated with them. A *token's color* is a schema, or tuple definition. *Places* are then sets of tuples, called *color sets*. Arcs specify the schema they carry, and can also specify basic boolean conditions. Specifically, arcs exiting a place may have an associated boolean function, called a guard, which enforces some constraints on tuple elements. Likewise, arcs entering a place may have an associated transformation function, which maps an actions output to some other tuple. This notation is demonstrated in examples below. CPNs are formally equivalent to traditional PNs; however, the richer notation makes it possible to model interactions in CPNs where it would be impractical to do so with PNs.

CPNs have great value for conversational modeling, in that:

- They are relatively simple formal model.
- They have a graphical representation.
- They support concurrency, which is necessary for many non-trivial interactions.

- They are well researched and understood, and have been applied to many real-world applications.
- Many tools and techniques exist for the design and analysis of CPN-based systems.

4 Putting Colored Petri Nets to Work

Currently, we are investigating the value of CPNs in a general framework for agent interaction specification. Within this scheme, agents use a common language, Protolingua, for manipulating CPN-based conversations. Protolingua itself is very sparse, and relies on the use of a basic interface definition language (IDL) for the association of well known functions and data types with a CPN framework. Agents use Protolingua interpreters to execute various protocols. Protolingua has been kept simple in order to facilitate the porting of interpreters to most or all platforms.

One advantage to this approach is that a variety of interpreter implementations may be used, and the agent may trade resources for conversational 'power'. A very simple CPN interpreter may be able to efficiently execute very small or simple protocols; an agent may chose to use this in most interactions, while employing more expensive and efficient interpreters for more complex negotiations. In addition to using direct CPN simulators, CPN specifications have a very natural embedding in a general rule-based framework.

To clarify the relationship between agents, interpreters, and protocols, let us assume that a Java-based agent would like to converse with another agent, and that it has determined, through assumption, negotiation, or other means, that it needs to use protocol xyz. It can obtain the declarative specification for xyz, if it does not already have it, from the other agent or from some third party; let's say a protocol server identified through a broker. Xyz contains the wire-frame specification of the protocol (arcs, places, transition), plus schema and functions given in the IDL. The agent can then obtain (as it did the specification) the executable attachments and type specifications appropriate for its interpreter (in this case, Java classes and associated methods), and then use the protocol to engage the other agent.

This very CORBA-like approach allows the use of very lightweight, universal interpreters without restricting the expressiveness of the protocols used. If types and actions are appropriately specified, they should be suitable for analysis, or translation into some analyzable form. For example, we are using DesignCPN, a tool from Aarhus University, Denmark, for high level design and analysis of protocols. This system uses an extension of ML, CPN-ML, as its modeling language. We plan to translate developed protocols into Protolingua and Java extensions, and restrict modification in such a way that the protocols can be translated back into CPN-ML for additional analysis.

5 Example: Conversation Protocol

From its inception, Jackal has used JDFA, a loose Extended Finite State Machine (EFSM), to model conversations [6, 18]. The base model is a Deterministic Finite State Automaton (DFA), but the tokens of the system are messages and message templates, rather than simply characters from an alphabet. Messages match template messages (with arbitrary match complexity) to determine arc selection. A local read/write store is available to the machine.

CPNs make it possible to formalize much of the extra-model extensions of DFAs. To make this concrete, we take the example of a standard JDFA representation of a KQML register conversation, as specified in [13], and reformulate it as a CPN. The graphic JDFA representation is depicted in Figure 1.

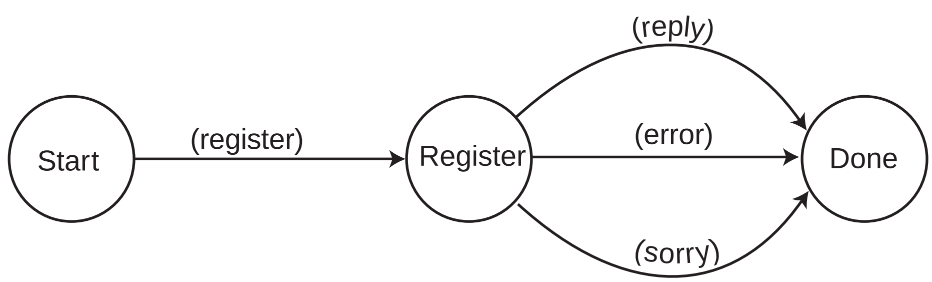

Figure 1: Diagrammatic Representation of KQML Register Conversation in JDFA

There are a number of ways to formulate any conversation, depending on the requirements of the user. This conversation has only one final state, but in some situations, it may be desirable to have the final state of the conversation denote the *result* of the interaction.

Some aspects of the model which are implicit under the DFA model must be made explicit under CPNs. The DFA allows a system to be in one state at a time, and to progress from one state to the next. Hence, the point at which an input is applied is clear, and that aspect is omitted from the diagrammatic representation. Since a CPN can always accept input at any location, we must make that explicit in the model.

We will use an abbreviated message which contains the following components, listed with their associated variable names: performative(p), sender(s), receiver(r), reply-with(id), in-reply-to(re), and content(c).

We denote the two receiving states as *places* of the same names (Figure 2). Places here represent receipt locations for messages. As no message is ever received in the initial state, we do not include a corresponding place. Instead, we use a a source *place*. This is implicit in the DFA representation. It must serve as input to every *action*, and could represent the input pool for the entire collection of conversations, or just this one. Note that the source has links to every *place*, but there is no path corresponding to the flow of state transitions, as in the DFA-based model.

The match conditions on the various arcs of the DFA we represent by *actions* preceding each existing *place*. Note that this one-to-one correspondence is not necessary. *Actions* may conditionally place tokens in different places, and several *actions* may concurrently deposit tokens in the same place.

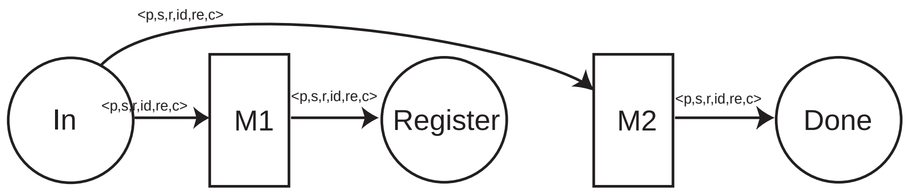

Figure 2: Source feeds all states in the model.

Various constants constrain the actions of the net, such as performative. These are typical represented as color sets in CPN, rather than hard-coded constraints. Other constraints are implemented as guards. The first, and most significant, ensures that the message sequence is observed, as prescribed by the message IDs. This not only assures that messages are delivered to the correct location, but in order, and only once. Not all conversations follow a simple, linear thread, however. We might, for example, want to send a message and allow an arbitrary number of asynchronous replies before responding (as is the case in a typical *subscribe*). In this case, we allow IDs to collect in a *place*, and remove them only when replies to them will no longer be accept. Finally, places interposed between actions implement global constraints, such as alternating sender and receiver.

We add a *place* after the final message transaction to denote some arbitrary action not implemented by the conversation protocol (that is, not by an arc-association action). This may be some event internal to the interpreter, or a signal to the executing agent itself. A procedural attachment at this location would not violate the conversational semantics as long as it did not in turn influence the course of the conversation.

This CPN is generally equivalent to the JDFA depiction above. In addition to modeling what is present in the JDFA, it also models mechanisms implicit in the machinery, such as message ordering. Also, the JDFA incorporates much which is beyond the underlying formal DFA model, and thus cannot be subjected to verification. The CPN captures all of the same mechanisms within the formal model.

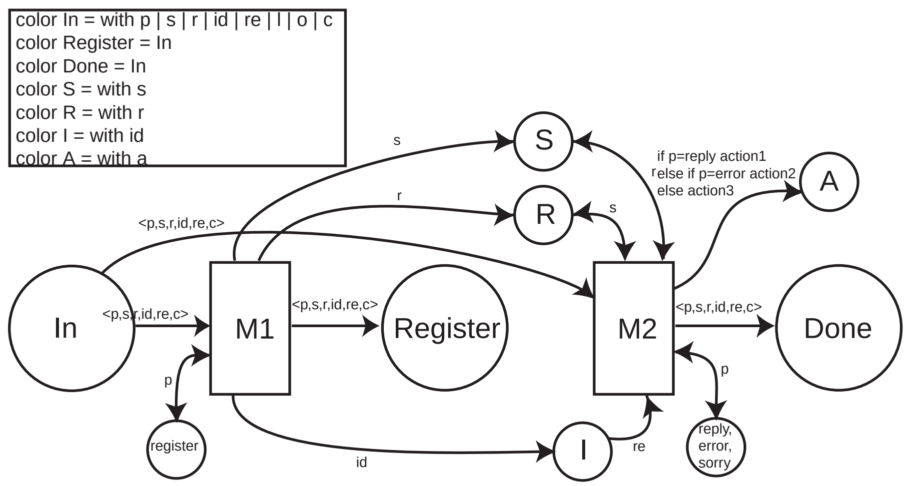

Figure 3: The final CPN specification, with complete notation.

6 Example: Negotiation Protocol

In this section we present a simple negotiation protocol proposed in [5]. The CPN diagram in Figure 4 describes the pair-wise negotiation process in a simple MAS, which consists of two functional agents bargaining for goods. The messages used are based on the FIPA ACL negotiation performative set, extended with five new performatives: accept-proposal, CFP, proposal, reject-proposal and terminate.

The diagram depicts three places places: Inactive, Waiting, and Thinking, which reflect the states of the agents during a negotiation process[1]. Both agents in this simple MAS have similar architecture, differing primarily in the number of places/states. This difference arises from the roles they play in the negotiation process. The agent that begins the negotiation, called the buyer agent, which is shown on the left side of the diagram, has the responsibility of handling message failures. For this, it has an extra 'wait' state, and timing machinery not present in the other agent. For simplicity, some constraints have been omitted from this diagram; for example, constraints on message types, as depicted in the previous example.

In this system, both agents are initially waiting in the Inactive places. The buyer initiates the negotiation process by sending a call for proposals (CFP) to some seller, and its state changes from Inactive to Waiting. The buyer is waiting

[1] It is not always the case with such a model that specific nodes correspond to states of the system or particular agents. More often the state of the system is described by the combined state of all places.

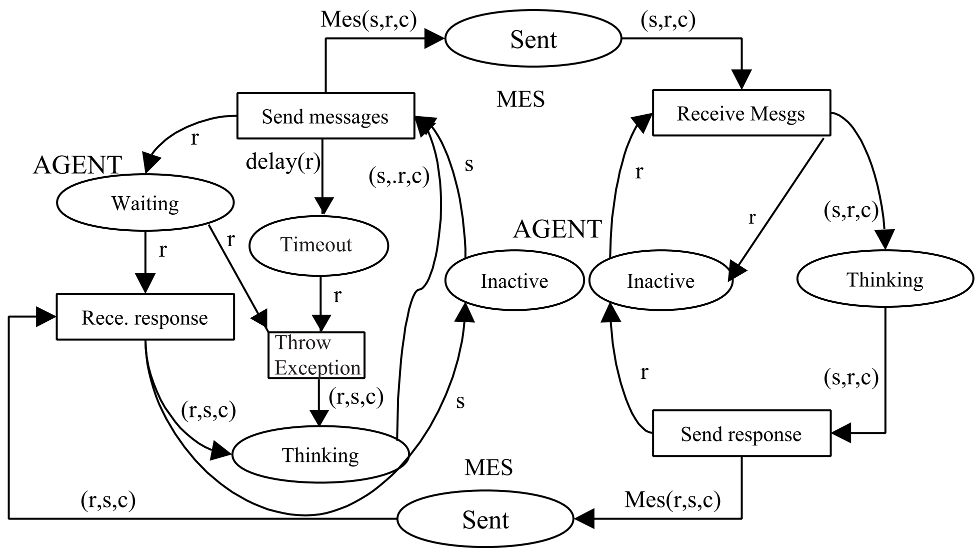

Figure 4: Pair-wise negotiation process for a MAS constituted of two functional agents.

for a response (proposal, accept-proposal, reject-proposal or terminate). On receipt, its state changes from Inactive to Thinking, at which point it must determine how it should reply. Once it replies, completing the cycle, it returns to the Inactive state. We have inserted a rudimentary timeout mechanism which uses a delay function to name messages which have likely failed in the Timeout place. This enables the exception action to stop the buyer from waiting, and forward information about this exception to the agent in the Thinking state.

Note that this protocol models concurrent pairwise interactions between a buyer and any number of sellers.

7 Summary

The use of conversation policies greatly facilitates the development of systems of interacting agents. While FSMs have proven their value over time in this endeavor, we feel that inherent limitations necessitate the use of a model supporting concurrency for the more complex interactions now arising. CPNs provide many of the benefits of FSMs, while allowing greater expression and concurrency. Using the Jackal agent development platform, we hope to demonstrate the value of CPNs as the underlying model for a protocol specification language, Protolingua.

References

[1] Tilak Agerwala. Putting petri nets to work. *Computer*, pages 85–94, December 1979.

[2] Mihai Barbuceanu and Mark S. Fox. COOL: A language for describing coordination in multiagent systems. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 17–25, San Francisco, CA, 1995. MIT Press.

[3] Jeffrey M. Bradshaw, Stuart Dutfield, Pete Benoit, and John D. Woolley. KAoS: Toward an industrial-strength open agent architecture. In Jeffrey M. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, 1998.

[4] Deepika Chauhan. JAFMAS: A java-based agent framework for multiagent systems development and implementation. Master's thesis, ECECS Department, University of Cincinnati, 1997.

[5] Ye Chen, Yun Peng, Tim Finin, Yannis Labrou, and Scott Cost. A negotiation-based multi-agent system for supply chain management. In *Working Notes of the First IAC Workshop on Internet Based Negotiation Technologies*, Yorktown Heights, NY, March 1999. IBM Institute for Advanced Commerce. Submitted.

[6] R. Scott Cost, Tim Finin, Yannis Labrou, Xiaocheng Luan, Yun Peng, Ian Soboroff, James Mayfield, and Akram Boughannam. Jackal: A java-based tool for agent development. In Jeremy Baxter and Chairs Brian Logan, editors, *Working Notes of the Workshop on Tools for Developing Agents, AAAI '98*, number WS-98-10 in AAAI Technical Reports, pages 73–82, Minneapolis, Minnesotta, July 1998. AAAI, AAAI Press.

[7] Ian Dickenson. Agent standards. Technical report, Foundation for Intelligent Physical Agents, october 1997.

[8] FIPA. FIPA 97 specification part 2: Agent communication language. Technical report, FIPA - Foundation for Intelligent Physical Agents, october 1997.

[9] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume Volume 1, Basic Concepts of *Monographs in Theoretical Computer Science*. SV, 1992.

[10] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume Volume 2, Analysis Methods of *Monographs in Theoretical Computer Science*. SV, 1994.

[11] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume Volume 3, Practical Use of *Monographs in Theoretical Computer Science*. SV, 1997.

[12] K. Kuwabara. AgenTalk: Coordination protocol description for multi-agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS '95)*. AAAI/MIT Press, 1995.

[13] Yannis Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland Baltimore County, 1996.

[14] Yannis Labrou and Tim Finin. Comments on the specification for FIPA '97 AGENT COMMUNICATION LANGUAGE. Internet document, 1997.

[15] M. H. Nodine and A. Unruh. Facilitating open communication in agent systems: the InfoSleuth infrastructure. In M. Singh, A. Rao, and M. Wooldridge, editors, *Proceedings of the 14th Annual Workshop on Agent Theories, Architectures and Languages (ATAL '97)*, Providence, RI, 1997.

[16] H. Van Dyke Parunak. Visualizing agent conversations: Using enhanced dooley graphs for agent design and analysis. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS '96)*, 1996.

[17] Y. Peng, T. Finin, Y. Labrou, B. Chu, J. Long, W. J. Tolone, and A. Boughannam. A multi-agent system for enterprise integration. In *Proceedings of the Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology*, London, April 1998. Submitted.

[18] Y. Peng, T. Finin, Y. Labrou, B. Chu, J. Long, W. J. Tolone, and A. Boughannam. A multi-agent system for enterprise integration. *International Journal of Agile Manufacturing*, 1998. To appear.

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

[20] James White. Mobile agents. In Jeffery M. Bradshaw, editor, *Software Agents*. MIT Press, 1995.

[21] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1986.