R. Scott Cost, Yannis Labrou, Tim Finin
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, Maryland 21250
cost@acm.org, {jklabrou,finin}@cs.umbc.edu

Coordinating Agents using Agent Communication Languages Conversations

March 16, 2000

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

0.1 Introduction

Internet agents are expected to accomplish their tasks despite heterogeneity; agents of different designs and of varying skills and domain knowledge need to interact successfully through knowledge and information exchange and effective coordination. We identify two distinct and separate problems that Internet agents are faced in an open and dynamic environment: knowledge sharing and coordination.

The knowledge sharing problem between heterogeneous agents has a long thread of work behind it. For a large category of agent systems, the main mechanism for knowledge sharing is the Agent Communication Language (ACL). An ACL relies on a three-layer conceptual breakdown of the knowledge sharing problem. The concepts originated in the work of the Knowledge Sharing Effort (KSE) [37] and have found their way into Knowledge Query Manipulation Languages (KQML) [2] [29], the first ACL, and in FIPA ACL, the ACL proposed by the Foundation for Intelligent Physical Agents (FIPA), a standardization organization in the area of agents. A simple formulation of the *knowledge sharing* problem between heterogeneous intelligent agents is: "expressions in a given agent's native language should be understood by some other agent that uses a different implementation language and domain assumptions".

The three-layered approach distinguishes between propositions and *propositional attitudes*. The first and the second layers are concerned with sharing the meaning of propositions and the third layer is concerned with sharing the meaning of propositional attitudes. So, the first layer is that of (syntactic) translation between languages in the same family (or between families) of languages [1]. The second layer is concerned with guaranteeing that the semantic content of tokens is preserved across applications; in other words, the same concept, object, or entity has a uniform meaning across agents even if different "names" are used to refer to it. Every agent incorporates some view of the domain (and the domain knowledge) to which it applies. The technical term for this body of "background" knowledge is *ontology*. More formally, an ontology is a particular conceptualization of a set of objects, concepts and other entities about which knowledge is expressed and of the relationships that hold among them. An ontology consists of terms, their definitions, and axioms relating them [20]; terms are normally organized in a taxonomy. The final layer addresses the communication between agents. This is not merely about transporting bits and bytes between agents; agents should be able to communicate complex "attitudes" about their information and knowledge content. Agents need to ask other agents, to inform them, to request their services for a task, to find other agents who can assist them, to monitor values and objects, and so on. Such functionality, in an open environment,

[1] The Object Management Group (OMG) standardization effort is an example of work in this direction, within the family of object-oriented languages.

can not be provided by a simple Remote Procedure Call (RPC) mechanism. Agents issue requests by specifying not a procedure but a desired state in a declarative language, *i.e.*, in some ACL.

Assuming effective translations between their respective representation languages, with shared ontologies, an ACL allows them to share their knowledge content. But knowledge sharing alone does not guarantee effective problem solving any more than superb language skills guarantee effective performance for a human agent. Accomplishing tasks requires coordination.

Coordination is the process by which agents reason about their local actions and the (anticipated) actions of others in order to ensure that all agents in a community act in a coherent manner towards a goal or a set of goals. The actions of multiple agents need to be coordinated because of dependencies between agents' actions, there is a need to meet global constraints, and no one agent has sufficient competence, resources or information to achieve such system goals. Examples of coordination include supplying timely information to other agents or ensuring that the actions of agents are synchronized. In an environment of heterogeneous agents coordination is a difficult task because agent must reconcile conflicting or incomplete views of the environment they are acting upon or interacting with.

An ACL offers agents building blocks for coordination. To the extend that coordination is communicative, as is often the case in heterogeneous agent communities, agents can use the propositional attitudes supplied by ACLs in order to sustain the complex interactions with other agents that are necessary for coordination. The ACL alone, though, does not help agents to decide when to "talk" and what exactly to "say" when they do so. Ideally, agents would possess such a rich understanding of themselves, their environment and other agents, that, like human agents, they could decide what to "say" and for what purpose. There is a sizable amount of work in this direction, the problem is extremely difficult in an open and dynamic environment. The direction we explore in this chapter in differs in that it is not concerned with the internal structure of agents. ACL research has been exploring *conversation protocols* as mechanisms for structuring agent interactions. Such protocols serve as prearranged protocols for coordinating specific task-related agent interactions. We are concerned with a suitable formalism for expressing such coordination protocols. Our goal is a formalism that guarantees certain useful properties for such protocols.

We next outline the organization of this chapter. In Section 0.2 we very briefly outline the basic concepts of ACLs and conversation protocols for ACL-speaking agents. In Section 0.3 we discuss conversation protocols, their use for coordination between agents and related work by other researchers. We continue by presenting our formalism for specifying conversation protocols, which is based on Colored Petri Nets CPNs), in Section 0.4, complete with examples of conversation protocols specified in our formalism. In Section

0.5 we discuss the merits of CPN-based descriptions of conversation protocols for agent coordination and in Section 0.6 we present relevant work on CPNs.

0.2 From Agent Communication Languages to Conversation Protocols

An Agent Communication Language provides agents with a means to exchange information and knowledge. ACLs, such as KQML or FIPA ACL, are languages of propositional attitudes. Propositional attitudes are three-part relationships between: (1) an agent, (2) a content-bearing proposition (*e.g.*, "it is raining"), and (3) a finite set of propositional attitudes an agent might have with respect to the proposition (*e.g.*, believing, asserting, fearing, wondering, hoping, *etc.*). For example, $< a, fear, raining(t_{now}) >$ is a propositional attitude.

ACLs are intended to be above the layer of mechanisms such as RPC or RMI because: (1) they handle propositions, rules and actions instead of simple objects (with no semantics associated with them), and (2) the ACL message describes a desired state in a declarative language, rather than a procedure or method. But ACLs by no mean cover the entire spectrum of what agents may want to exchange. More complex *objects* can and should be exchanged between agents, such as shared plans and goals, or even shared experiences and long-term strategies.

At the technical level, when using an ACL, agents transport messages over the network using some lower-level protocol (SMTP, TCP/IP, IIOP, HTTP, etc.). The ACL itself defines the types of messages (and their meaning) that agents may exchange. Agents though, do not just engage in single message exchanges but they have *conversations*, *i.e.* task-oriented, shared sequences of messages that they observe, in order to accomplish specific tasks, such as a negotiation or an auction. At the same time, some higher-level conceptualization of the agent's strategies and behaviors drives the agent's communicative (and non-communicative) behavior.

Traditionally, we understand the message types of ACLs as *speech acts*, which in turn are usually accounted for in terms of beliefs, desires, intentions and similar modalities. This kind of intentional-level description can either be just a useful way to view a system or it can have a concrete computational aspect. The latter case describes a large range of BDI[2] agents which have some (implicit or explicit) representation of the corresponding modalities. This representation is built on top of a substrate that describes the conceptual model of knowledge, goals and commitments of an agent, commonly known as a BDI theory. Despite the criticism that BDI theories and BDI agents have faced, such as the number and choice of modalities and the fact that multi-modal BDI logics have neither complete axiomatizations nor they are

[2] BDI stands for *Belief*, *Desire* (or Goal) and *Intention*.

efficiently computable, they offer an appealing framework to account for agent communications, because agents, when communicating, they communicate their BDI states and/or attempt to alter their interlocutors BDI states.

While an ACL is good for message-passing among agents, on its own it does not offer much in terms of agent coordination. But 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 both the KQML [30] and the FIPA communities [18, 12].

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 pre-arranged coordination protocol, A conversation lends context to the sending and receipt of messages, facilitating interpretation that is more meaningful.

Although, conversations have become part of many infrastructures for ACL-speaking agents, relatively little work has been devoted to the problem of conversation specification and implementation. For conversations to be used for agent coordination, the following three issues require attention:

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 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?

Although we have worked on items (2) and (3) (see [10]), our primary focus, in arguing the usefulness of conversations for agent coordination, is on their specification with emphasis on the ability to validate properties that are important in the context of coordination, such as liveliness and reachability.

0.3 Coordination using Conversation Protocols

Well-defined, sharable conversation protocols, with testable, desirable properties, can be used to coordinate agents that attempt to accomplish specific tasks. Typically, a conversation protocol is associated with a specific task, such as *registration*, or a particular type of a *negotiation*. Agents adhering to the same conversation protocol, can coordinate their communicative actions as they attempt to accomplish the task suggested by the conversation protocol. Such a coordination is akin to a scripted interaction, with specific properties, rather than coordinated action resulting from a deep understanding of the domain and the task at hand. Nevertheless, it can be effective for tasks that can be adequately described in the form of possible sequences of communicative interactions.

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. Next, the set of roles that agents engaging in a conversation may play must be enumerated, and the constraints and dependencies between individual messages need to be captured. 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. For some conversations, the set of participants may change during the course of the interaction.

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.

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.

Implementing and expressing conversations for software agents is not a new idea. As early as 1986, Winograd and Flores [46] used state transition diagrams to describe conversations. The COOL system [3] has perhaps the most detailed current FSM-based 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 have been developed, using various approaches. Extended FSM models, which, like COOL, focus more on expressivity than adherence to a model, include Kuwabara et al. [27], who add inheritance to conversations, Wagner et al. [44], and Elio and Haddadi [13], who defines a multi-level state machine, or Abstract Task Model (ATM). A few others have chosen to stay within the bounds of a DFA, such as Chauhan [6], who uses COOL as the basis for her multi-agent development system [3], Nodine and Unruh [38], and Pitt and Mamdani [41], who uses DFAs to specify protocols for BDI agents. Also using automata, Martin et al. [33] employs Push-Down Transducers (PDT). Lin et al. [32] and Cost et al. [9] demonstrate the use of CPNs, and Moore [36] applies state charts. Parunak [39] introduces Dooley Graphs. Bradshaw [5] introduces the notion of a conversation suite as a collection of commonly-used conversations known by many agents. Labrou [28] uses definite clause grammars to specify conversations.

While each of these works makes contributions to our general understanding of conversations, more work needs to be done to facilitate the sharing and use of conversation policies by agents.

In the next section of this chapter we describe a formalism based on Colored Petri Nets, which can be used to specify conversation protocols.

0.4 Modeling Conversation Protocols with Colored Petri Nets

0.4.1 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 *transition*. *Tokens* occupy places. When there is at least one token in every place connected to a transition, we say that the 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].

CPNs 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 type specification. Places are then sets of tuples, called *multi-sets*. *Arcs* specify the schema they carry, and can also specify basic boolean conditions. Specifically, arcs exiting and entering a place may have an associated

[3] More recent work with this project, JAFMAS, explores conversion of policies to standard Petri Nets for analysis [19].

function which determines what multi-set elements are to be removed or deposited. Simple boolean expressions, called *guards*, are associated with the transitions, and enforce some constraints on tuple elements. 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 a relatively simple formal model with a graphical representation and support for concurrency, which is necessary for many non-trivial interactions. Additionally, CPNs are well researched and understood, and have been applied to many real-world applications. As such, many tools and techniques exist for the design and analysis of CPN-based systems.

0.4.2 Conversation Protocols using Colored Petri Nets

As mentioned earlier, conversations are often specified by simple or modified DFAs. For the purposes of example, we will consider conversation definition in JDFA, a loose Extended Finite State Machine (EFSM) for modeling conversations, introduced in [10], [40]. The base model is a 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, including recursive matching on message content) 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[4] and reformulate it as a CPN. The graphic depiction of this JDFA specification can be seen in Figure 0.1.

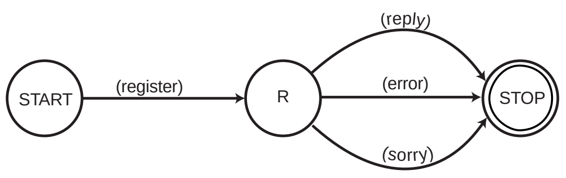

Fig. 0.1. Diagrammatic DFA representation of the simplified KQML Register conversation

There are a number of ways to formulate any conversation, depending on the requirements of use. This conversation has only one final, or accepting, state, but in some situations, it may be desirable to have multiple accepting

[4] A *register* conversation is one the most basic KQML conversations, used for an agent to register its name and availability with an agent service. FIPA ACL uses a similar concept.

states, and have the final state of the conversation denote the *result* of the interaction.

In demonstrating the application of CPNs here, we will first develop an informal model based on the simplified Register conversation (in KQML) presented, and then describe a complete and working CPN-ML model of the full Register conversation.

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 shows the progression from one state to the next. Hence, the point to 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)[5].

We denote the two receiving states as places of the names **Register** and **Done** (Figure 0.2). These place serve as a receipt locations for messages, after processing by the transitions **T1** and **T2**, respectively. As no message is ever received into the initial state, we do not include a corresponding place. Instead, we use a a source place, called **In**. This is implicit in the DFA representation. It must serve as input to every transition, 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 are implemented by transitions preceding each existing place. Note that this one-to-one correspondence is not necessary. Transitions may conditionally place tokens in different places, and several transitions may concurrently deposit tokens in the same place.

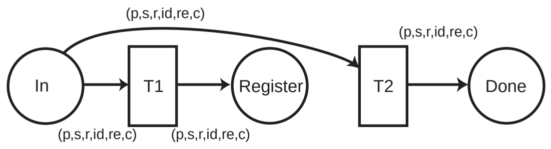

Fig. 0.2. Preliminary CPN model of a simplified KQML register conversation.

Various constants constrain the actions of the net, such as performative (Figure 0.3). These can be represented as color sets in CPN, rather than hard-coded constraints. Other constraints are implemented as guards;

[5] These variable names correspond to attributes of a typical ACL message. Syntactically, a ACL message is a balanced parenthesis list that begins with the *performative* (or message type, or propositional attitude) followed by attribute-value pairs for the sender of the message, the intended receiver and so on

boolean conditions associated with the transitions. Intermediate places **S**, **R** and **I** assure that sender, receiver and ID fields in the response are in the correct correspondence to the initial messages. **I** not only ensures that the message sequence is observed, as prescribed by the message IDs, but that only one response is accepted, since the ID marker is removed following the receipt of one correct reply. 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 to the same ID before responding (as is the case in a typical Subscribe conversation), or allow a response to any one of a set of message IDs. In these cases, we allow IDs to collect in a place, and remove them only when replies to them will no longer be accepted. Places interposed between transitions to implement global constraints, such as alternating sender and receiver, may retain their markings; that is implied by the double arrow, a shorthand notation for two identical arcs in opposite directions.

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.

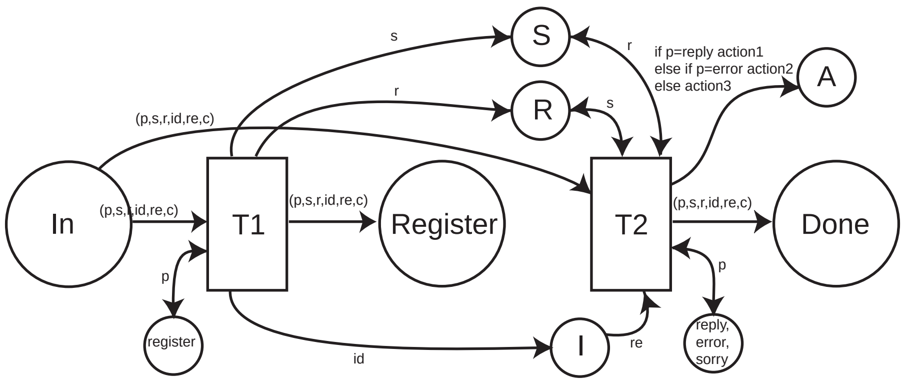

Fig. 0.3. Informal CPN model of a simplified KQML register conversation.

This CPN is generally equivalent to the JDFA depicted in Figure 0.1. 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.

0.4.3 A Conversation protocol for a simple Negotiation between Agents

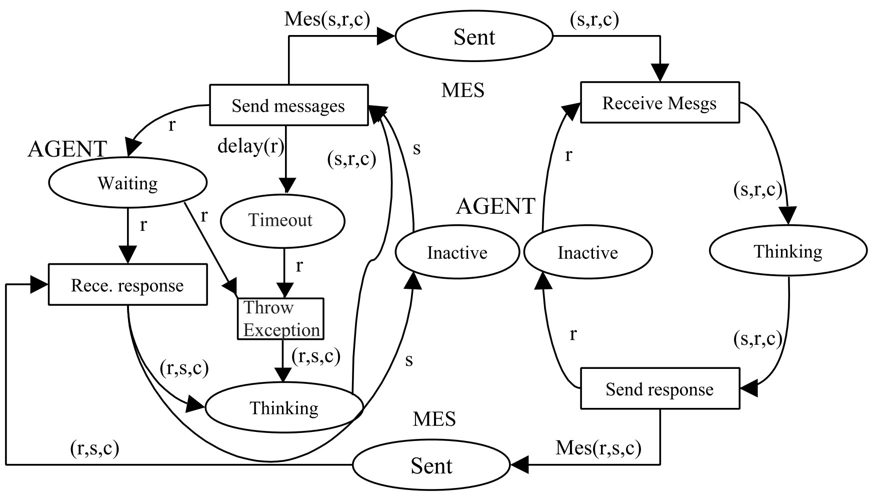

Fig. 0.4. Pair-wise negotiation process for a MAS constituted of two functional agents.

In this section we present a simple negotiation protocol proposed in [7]. The CPN diagram in Figure 0.4 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.

The diagram depicts three places places: **Inactive**, **Waiting**, and **Thinking**, which reflect the states of the agents during a negotiation process[6]; we will use the terms state and place interchangeably. 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 (**Waiting**), and timing machinery not present in the other agent, *seller*. For simplicity, some constraints have been omitted from this diagram; for example, constraints on message types, as depicted in the previous examples.

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

[6] 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.

buyer is waiting 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 (**Throw Exception**) to stop the buyer from waiting, and forward information about this exception to the agent in the **Thinking** state. Timing can be handled in a number of ways in implementation, including delays (as above), the introduction of timer-based interrupt messages, or the use of timestamps. CPN-ML supports the modeling of time-dependent interactions through the later approach.

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

0.5 Advantages for Coordination when using CPN-described Conversations

There are a number of benefits to be derived from modeling agent coordination with a formal system, and with CPNs in particular. Formal models have a more clearly defined semantics, and are more amenable to standardization, exchange and reuse. They also facilitate analysis, both empirical and analytic.

The ability to verify the properties of a specification is one of the important benefits of applying formal methods. These benefits can be derived in two ways:

– Verification of the conversation policies or protocols directly, and
– Verification of agents or Multi Agent Systems (MAS) that are based on such protocols.

We will first consider the range of properties amenable to analysis, and then discuss their value in the two contexts described. The focus will be on the methods provided by Design/CPN and associated tools.

In addition to 'proof by execution', CPNs can be checked for a variety of properties. This is done by way of an Occurrence Graph (OG) [11]. Each node in an OG consists of a possible marking for the net. If another marking (B) can be reached by the firing of a transition, the graph contains a directed arc from the node representing the initial marking to B. All nodes in an OG are therefore derived from some initial marking of the net.

The properties subject to verification are:

1. Reachability Properties: This relates to whether or not the marking denoted by node B is reachable by some sequence of transition firings from node A.

2. Boundedness Properties: The upper or lower bound on the contents of place X in the net, over all possible markings. This can be the cardinality of the multiset at node X, or the greatest or least multiset itself.
3. Home Properties: The marking or set of markings which are reachable from all other markings in the OG define a homespace. One can verify that a marking or set of markings constitutes a homespace, or determine whether or not a home marking exits, and what the minimal such marking is.
4. Liveness Properties: A marking from which no further markings can be derived is 'dead'. Liveness, then, relates to the possible progressions from a given node in the OG. One can verify that a marking is dead, or list dead markings in the OG.
5. Fairness Properties: Relates to the degree to which certain transition instances (TI) will be allowed with respect to other TIs.

Many of these properties have different values depending on whether we are regarding a conversation protocol or a Multi Agent System, and also on the complexity of the net. Conversation protocols describe/operate on a message stream, which in most cases is finite; they are themselves static. One can imagine analyzing a conversation protocol in the context of (1) a single message stream, or (2) in the presence of a generator for all or many representative streams. In that sense, we may be interested in boundedness or home properties, and possibly reachability or fairness, but not liveness. On the other hand, liveness and fairness will often be more important in the analysis of a system as a whole.

It is possible to verify properties even for very large and complex nets. The version of Design/CPN used in this research supports the computation and analysis of OGs of 20,000 - 200,000 nodes and 50,000 to 2,000,000 arcs.

0.6 Related Work

CPNs are not new, and they have been used extensively for a broad range of applications (see [25] for a survey of current uses). Since their target domain is distributed systems, and the line between that domain and MASs is vague at best, there is much work on which to build. We will review here a few of the more directly related research endeavors.

Holvoet and Verbaeten have published extensively on the subject of agents and PNs. In their 1995 paper, "Agents and Petri Nets" [21], they introduced the idea of enhancing AOP by using high-level nets to model agents, and extended this thought in [22] to a variant called 'Generic Nets'. In 1997, Holvoet and Kielmann introduced PNSOL (Petri Net Semantics for Objective Linda) [23, 24], used to model agents which live in and communicate through the Objective Linda [26] tuple space.

Yoo, Merlat and Briot [47] describe a contract-net based system for electronic commerce that uses a modular design. Among the components are BRICS (Block-like Representation for Interacting Components) [17]), which are derived from CPNs.

Fallah-Seghrouchni and Mazouzi have demonstrated the use of CPNs in specifying conversation policies in some detail, using FIPA ACL as a framework [15, 14, 16]. This work suggests an approach for hierarchical construction of conversations.

Moldt and Wienberg have developed an approach called AOCPN (Agent Oriented Colored Petri Nets) [45, 35]. This system employed an object-oriented language, syntactically similar to C++, which maps onto CPN, extended by 'test arcs' [8, 31]. They show how this approach can be used to model societies of agents as described by Shoham [43]. Their model extends down to the level of individual agent theorem provers, facilitating the logical specification of agent behavior.

Other work of note includes Billington et al. [4], Purvis and Cranefield [42], Lin et al. [32] (above), and Merz and Lamersdorf [34].

0.7 Conclusions

An Agent Communication Language is a powerful framework for interacting agents in an open and dynamic environment. Although an ACL provides a framework for knowledge sharing between agents, it is, by itself, inadequate for coordination between agents. Conversations, *i.e.*, well-specified sequences of message exchanges geared towards particular tasks, use ACL messages as building blocks for scripted, task-oriented interactions between agents. Such well-specified conversations, with testable properties, can be used to coordinate the behavior among communicating agents that attempt to accomplish the tasks specified by conversations.

While FSMs have proven their value for describing conversation policies, we feel that inherent limitations necessitate the use of a model supporting concurrency for the more complex interactions. CPNs provide many of the benefits of FSMs, while allowing greater expression and concurrency. We presented a CPN-based formalism for modeling conversation policies and we presented examples of using this formalism to specify conversations. This formalism provides for the testing of properties that are valuable for agent coordination.

1. Tilak Agerwala. Putting Petri Nets to work. *Computer*, pages 85–94, December 1979.
2. ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group, July 1993.
3. 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.
4. J. Billington, M. Farrington, and B. B. Du. Modelling and analysis of multi-agent communication protocols using CP-nets. In *Proceedings of the third Biennial Engineering Mathematics and Applications Conference (EMAC'98)*, pages 119–122, Adelaide, Australia, July 1998.
5. Jeffrey M. Bradshaw. KAoS: An open agent architecture supporting reuse, interoperability, and extensibility. In *Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996.
6. Deepika Chauhan. JAFMAS: A Java-based agent framework for multiagent systems development and implementation. Master's thesis, ECECS Department, University of Cincinnati, 1997.
7. 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 Agents '99 Workshop on Agents for Electronic Commerce and Managing the Internet-Enabled Supply Chain.*, Seattle, WA, April 1999.
8. Søren Christensen and Niels Damgaard Hansen. Coloured petri nets extended with place capacities, test arcs and inhibitor arcs. Technical Report DAIMI PB-398, Computer Science Department, Aarhus University, Aarhus C, Denmark, May 1992.
9. R. Scott Cost, Ye Chen, Tim Finin, Yannis Labrou, and Yun Peng. Modeling agent conversations with colored petri nets. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59–66, Seattle, Washington, May 1999.
10. 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, Minnesota, July 1998. AAAI, AAAI Press.
11. Department of Computer Science, University of Aarhus, Denmark. *Design/CPN Occurrence Graph Manual*, version 3.0 edition, 1996.
12. Ian Dickenson. Agent standards. Technical report, Foundation for Intelligent Physical Agents, October 1997.

13. Renée Elio and Afsaneh Haddadi. On abstract task models and conversation policies. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 89–98, Seattle, Washington, May 1999.
14. A. El Fallah-Seghrouchni, S. Haddad, and H. Mazouzi. A formal study of interactions in multi-agent systems. In *Proceedins of ISCA International Conference in Computer and their Applications (CATA '99)*, April 1999.
15. A. El Fallah-Seghrouchni and S. Haddad H. Mazouzi. Etude des interactions basée sur l'observation reépartie dans un systéme multi-agents. In Hermés, editor, *Proceedings of JFIADSMA '98*, Nancy, France, November 1998.
16. Amal El Fallah-Seghrouchni and Hamza Mazouzi. A hierarchial model for interactions in multi-agent systems. In *Working Notes of the Workshop on Agent Communication Languages, IJCAI '99*, August 1999.
17. Jaques Ferber. *Les Système Multi-Agents*. InterEditions, 1996.
18. FIPA. FIPA 97 specification part 2: Agent communication language. Technical report, FIPA - Foundation for Intelligent Physical Agents, October 1997.
19. Alan Galan and Albert Baker. Multi-agent communications in JAFMAS. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 67–70, Seattle, Washington, May 1999.
20. Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 2:199–220, 1993.
21. T. Holvoet. Agents and petri nets. *The Petri Net Newsletter*, (49):3–8, 1995.
22. T. Holvoet and P. Verbaeten. Synchronization specifications for agents with net-based behavior descriptions. In *Proceedings of CESA '96 IMACS Conference, Symposium on Discrete Events and Manufacturing Systems*, pages 613–618, Lille, France, July 1996.
23. Tom Holvoet and Thilo Keilmann. Behavior specification of active objects in open generative communication environments. In Hesham El-Rewini and Yale N. Patt, editors, *Proceedings of the HICSS-30 Conference, Track on Coordination Models, Languages and Systems*, pages 349–358. IEEE Computer Society Press, January, 7–10 1997.
24. Tom Holvoet and Pierre Verbaeten. Using petri nets for specifying active objects and generative communication. In G. Agha and F. DeCindio, editors, *Advances in Petri Nets on Object-Orientation*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
25. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume Volume 3, Practical Use of *Monographs in Theoretical Computer Science*. Springer-Verlag, 1997.
26. Thilo Kielmann. Designing a coordination model for open systems. In P. Ciancarini and C. Hankin, editors, *Coordination Languages and Models: Proceedings of COORDINATION '96*, number 1061 in Lecture Notes in Computer Science, pages 267–284. Springer, Cesena, Italy, 1996.
27. Kazuhiro Kuwabara, Toru Ishida, and Nobuyasu Osato. AgenTalk: Describing multiagent coordination protocols with inheritance. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '95)*, pages 460–465, 1995.
28. Yannis Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland Baltimore County, 1996.
29. Yannis Labrou and Tim Finin. A proposal for a new kqml specification. Technical Report Technical Report TR-CS-97-03, University of Maryland Baltimore County, 1997.
30. Yannis Labrou and Timothy Finin. Semantics and conversations for an agent communication language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, August 1997.

31. C. Lakos and Søren Christensen. A general systematic approach to arc extensions for coloured petri nets. Technical Report R93-7, Department of Computer Science, University of Tasmania, Hobart, Tasmania, August 1993.
32. Fuhua Lin, Douglas H. Norrie, Weiming Shen, and Rob Kremer. Schema-based approach to specifying conversation policies. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Third International Conference on Autonomous Agents*, pages 71–78, Seattle, Washington, May 1999.
33. Francisco Martin, Enric Plaza, and Juan Rodríguez-Aguilar. Conversation protocols: Modeling and implementing conversations in agent-based systems. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 49–58, Seattle, Washington, May 1999.
34. M. Merz and W. Lamersdorf. Agents, services, and electronic markets: How do they integrate? In *Proceedings of the IFIP/IEEE International Conference on Distributed Platforms*, Dresden, Germany, 1996.
35. Daniel Moldt and Frank Wienberg. Multi-agent-systems based on coloured petri nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets (ICATPN '97)*, number 1248 in Lecture Notes in Computer Science, pages 82–101, Toulouse, France, June 1997.
36. Scott Moore. On conversation policies and the need for exceptions. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 19–28, Seattle, Washington, May 1999.
37. R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, Fall 1991.
38. M. H. Nodine and A. Unruh. Facilitating open communication in agent systems: the InfoSleuth infrastructure. In Michael Wooldridge, Munindar Singh, and Anand Rao, editors, *Intelligent Agents Volume IV – Proceedings of the 1997 Workshop on Agent Theories, Architectures and Languages*, volume 1365 of *Lecture Notes in Artificial Intelligence*, pages 281–295. Springer-Verlag, Berlin, 1997.
39. 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.
40. Y. Peng, T. Finin, Y. Labrou, R. S. Cost, B. Chu, J. Long, W. J. Tolone, and A. Boughannam. An agent-based approach for manufacturing integration - the CIIMPLEX experience. *International Journal of Applied Artificial Intelligence*, 13(1–2):39–64, 1999.
41. Jeremy Pitt and Abe Mamdani. Communication protocols in multi-agent systems. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 39–48, Seattle, Washington, May 1999.
42. M. Purvis and S. Cranefield. Agent modelling with petri nets. In *Proceedings of the CESA '96 (Computational Engineering in Systems Applications) Symposium on Discrete Events and Manufacturing Systems*, pages 602–607, Lille, France, July 1996. IMACS, IEEE-SMC.
43. Yoav Shoham. Agent–oriented programming. *Artificial Intelligence*, 60:51–92, 1993.
44. Thomas Wagner, Brett Benyo, Victor Lesser, and Ping Xuan. Investigating interactions between agent conversations and agent control components. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 79–88, Seattle, Washington, May 1999.
45. Frank Wienberg. *Multiagentensysteme auf def Basis gefärbter Petri-Netze*. PhD thesis, Universität Hamburg Fachbereich Informatik, 1996.

46. Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1986.
47. Min-Jung Yoo, Walter Merlat, and Jean-Pierre Briot. Modeling and validation of mobile agents on the web. In *Proceedings of the International Conference on Web-Based Modeling & Simulation (SCS Western MultiConference on Computer Simulation)*, San Diego, California, January 1998.