

Multi-Agent Dialogue Protocols

Christopher D. Walton (cdw@inf.ed.ac.uk) *

Centre for Intelligent Systems and their Applications (CISA), Edinburgh, UK

November 28, 2003

Abstract

In this paper we propose a new agent communication language which separates agent dialogue from any specific agent reasoning technology. This language is intended to address a number of perceived shortcomings with the mentalistic model of agent communication on which the FIPA-ACL standard is founded. Our language expresses inter-agent dialogue through the use of agent protocols, and is intended to be independent of the technology used for message delivery. In this paper we specify the syntax of our communication language, together with an operation semantics which defines an implementation of the language. Our language specification is derived from process calculus and thus forms a sound basis for the verification of our agent protocols.

1 Introduction

A Multi-Agent-System (MAS) may be defined as a collection of *agents*, which are autonomous and rational components, that interact within an environment [Jen00]. An individual agent of a MAS exhibits intelligent behaviour based on interactions with other agents, the environment, and internal reasoning processes. It is this intelligent behaviour that distinguishes a MAS from a conventional distributed or parallel software system. From this definition it is clear that an essential pragmatic consideration in the construction of a MAS must be a specification for the interactions between individual agents, as agents must interact in order to exhibit intelligent behaviour. A popular basis for this interaction is the theory of theory of *rational action* by Cohen and Levesque [CL90]. The FIPA-ACL specification [FIP99] recognises this theory by providing a formal semantics for the performatives expressed in BDI logic [RG98]. However, there is a growing dissatisfaction with the mentalistic model of agency as a basis for defining *inter-operable* agents between different agent platforms [Sin98].

Inter-operability requires that agents built by different organisations, and using different software systems, are able to safely communicate with one another in a common language with an agreed semantics. The problem with the BDI model as a basis for inter-operable agents is that although agents can be defined according to a commonly agreed semantics, it is not generally possible to verify that an agent is acting according to these semantics.

*This work is sponsored by the UK Engineering and Physical Sciences Research Council (Grant GR/N15764/01) Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration.

This stems from the fact that it is not known how to assign mental states systematically to arbitrary programs. For example, we have no way of knowing whether an agent actually believes a particular fact. For the semantics to be verifiable it would be necessary to have access to an agents' internal mental states which is not typically possible. This problem is known as the *semantic verification* problem and is detailed in [Woo00]. In order to avoid the semantic verification problem a number of alternative semantics for expressing rational agency have been proposed. Two of these approaches are a semantics based on social commitments, and a semantics based on dialogue games. A summary of these approaches, and other semantic models is presented in [MC02].

In this paper we do not adopt a specific semantics of rational agency, or define a fixed model of interaction between agents. Our belief is that in a truly heterogeneous agent system we cannot constrain the agents to any particular model. Instead, we define a model of dialogue which separates the rational process and interactions from the actual dialogue itself. This is accomplished through the adoption of a *dialogue protocol* which exists at a layer between these processes. This approach has been adopted in the Conversation Policy [GHB99] and Electronic Institutions [ERS⁺01] formalisms. The definition presented in this paper differs in that dialogue protocol specifications can be directly executed.

A dialogue protocol allows the semantics of a dialogue to be independently expressed. This approach has some compelling advantages, for example, we can succinctly express the rules of an auction as a dialogue protocol, while the agents participating in this dialogue are free to select their own auction strategies, i.e. dialogue protocols do not compromise the self-interest of the individual agents. Agents can be specified in different languages, using different rational processes, and still participate in the dialogue expressed in the protocol. The only restriction on the autonomy of the agents is that they follow the dialogue protocol which encodes all the necessary information to participate in the dialogue. It should be noted that dialogue protocols also greatly assist in the design of large MAS as they impose structure on the agents, and co-ordinate tasks between agents. They also simplify the design of individual agents as they separate the task of defining the co-ordination of the agents from the definition of agent behaviours. This separation also permits the refinement and verification of the agent protocol independently from the design of the individual agents.

In this paper we present our approach to defining dialogue protocols suitable for constructing large MAS of inter-operable agents. Our method draws from our earlier experiences with defining Electronic Institutions and borrows its theory from the process calculus domain. In section 2 we present a small language which we use to express dialogue protocols. In section 3 we present a relational operational-semantics for evaluating our language which can be implemented in an agent system. Lastly, in section 4 we describe our implementation and discuss future work concerning the verification of our dialogue protocols.

2 The MAP Language

The MAP language is a lightweight dialogue protocol language which provides a replacement for the state-chart representation of protocol found in Electronic Institutions [ERS⁺01]. Our formalism allows the definition of infinite-state dialogues and the mechanical processing of the resulting dialogue protocols. The semantics of our language are derived from the field of

process calculus, and in particular the Calculus of Communicating Systems (CCS) [Mil89]. We have redefined the core of the Electronic Institutions framework to provide an executable specification, while retaining the concepts of *institutions*, *scenes*, and *roles*.

The division of agent dialogues into *scenes* is a key concept in our protocol language. A scene can be thought of as a bounded space in which a group agents interact on a single task. The use of scenes divides a large protocol into manageable chunks. For example, a negotiation scene may be part of a larger marketplace institution. Scenes also add a measure of security to a protocol, in that agents which are not relevant to the task are excluded from the scene. This can prevent interference with the protocol and limits the number of exceptions and special cases that must be considered in the design of the protocol. Additional security measures can also be introduced into a scene, such as placing entry and exit conditions on the agents, though we do not deal with these here. However, we assume that a scene places barrier conditions on the agents, such that a scene cannot begin until all the agents are present, and the agents cannot leave the scene until the dialogue is complete.

The concept of an agent *role* is also central to our definition of a dialogue protocol. Agents entering a scene assume a fixed role which persists until the end of the scene. For example, a negotiation scene may involve agents with the roles of *buyer* and *seller*. The protocol which the agent follows in a dialogue will typically depend on the role of the agent. For example, an agent acting as a seller will typically attempt to maximise profit and will act accordingly in the negotiation. A role also identifies capabilities which the agent must provide. For example, the buyer must have the capability to make buying decisions and to purchase items. Capabilities are related to the rational processes of the agent and are encapsulated by *decision procedures* in our definition.

$S \in \text{Scene}$	$::= n[\mathcal{R}, \mathcal{A}, P^{(k)}]$	(Scene Definition)
$P \in \text{Protocol}$	$::= \mathbf{agent}(a, r, \phi^{(k)}) = op$	(Agent Protocol)
$op \in \text{Operation}$	$::= \alpha$	(Action)
	$op_1 \mathbf{then} op_2$	(Sequence)
	$op_1 \mathbf{or} op_2$	(Choice)
	$op_1 \mathbf{par} op_2$	(Parallel Composition)
	$\mathbf{waitfor} op_1 \mathbf{timeout} op_2$	(Iteration)
	$\mathbf{agent}(\phi^{(k)})$	(Recursion)
$\alpha \in \text{Action}$	$::= \epsilon$	(No Action)
	$v = p(\phi^{(k)})$	(Decision Procedure)
	$M \Rightarrow \mathbf{agent}(\phi^{(2)})$	(Send)
	$M \Leftarrow \mathbf{agent}(\phi^{(2)})$	(Receive)
$M \in \text{Message}$	$::= \rho(\phi^{(k)})$	(Performative)
$\phi \in \text{Term}$	$::= v \mid a \mid r \mid c \mid _$	

Figure 1: MAP Abstract Syntax.

The abstract syntax of MAP is presented in Figure 1. Agents are uniquely identified by a name a , and have a fixed role r for the duration of the scene. A scene comprises a fixed set

of roles \mathcal{R} , a set of participating agents \mathcal{A} , and a sequence of protocols $P^{(k)}$. A protocol P can be considered a procedure where a , r , and $\phi^{(k)}$ are the arguments. The initial protocol for an agent is specified by setting $\phi^{(k)}$ to be empty (i.e. $k = 0$). Protocols are constructed from operations op which control the flow of the protocol, and actions α which have side-effects and can fail. The interface between the protocol and the rational process of the agent is achieved through the invocation of decision procedures p . Interaction between agents is performed by the exchange of messages M which contain performatives ρ . Procedures and performatives are parameterised by terms ϕ , which are either variables v , agents a , roles r , constants c , or wild-cards $_$. Variables are bound to terms by unification which occurs in the invocation of procedures, the receipt of messages, or through recursive calls.

```

1 GeneralPractitionerScene[!%patient, %doctor], {!Patient1, !Doctor1},
2
3 agent(!Patient1, %patient) =
4   request(appointment) => agent(_, %doctor) then
5   waitfor
6     (accept(appointment, $appointment) <= agent($doctor, %doctor) then
7     ($symptoms = getSymptoms() then
8     inform(symptoms, $symptoms) => agent($doctor, %doctor) then
9     waitfor
10      (inform(refer) <= agent($doctor, %doctor) or
11      inform(norefer) <= agent($doctor, %doctor))
12      timeout (e)) or
13      reject(appointment) <= agent($doctor, %doctor))
14   timeout (e)
15
16 agent(!Doctor1, %doctor) =
17   waitfor (request(appointment) <= agent($patient, %patient)) timeout (e) then
18   ($appointment = makeAppointment($patient) then
19   accept(appointment, $appointment) => agent($patient, %patient) then
20   waitfor
21     (inform(symptoms, $symptoms) <= agent($patient, %patient) then
22     ($ref = doReferral($patient, $symptoms) then
23     inform(refer) => agent($patient, %patient)) or
24     inform(norefer) => agent($patient, %patient))
25     timeout (e)) or
26   reject(appointment) => agent($patient, %patient)]

```

Figure 2: General Practitioner Protocol.

It is helpful to consider an example of a MAP protocol in order to illustrate these concepts. In Figure 2 we present an example scene which would form part of a larger institution. This scene defines an interaction protocol between doctor and patient agents. This scene is intended to represent a patient visiting a General Practitioner (GP) to obtain a diagnosis of some symptoms. We distinguish between the different types of terms by prefixing variables names with \$, role names with %, and agent names with !. We define two agents !Patient1 and !Doctor1 which have roles %patient and %doctor respectively. The protocol for the

patient is specified separately from the doctor, though the two will interact closely. The agents are synchronised purely through the exchange of messages.

When exchanging messages, through send and receive actions, a unification of terms in the definition `agent(ϕ^1 , ϕ^2)` is performed, where ϕ^1 is matched against the agent name, and ϕ^2 is matched against the agent role. For example, the request for an appointment in line 4 of the protocol will match any agent whose role is a `%doctor`. Similarly, the receipt of the request in line 17 of the protocol will match any agent whose role is `%patient`, and the name of this agent will be bound to the variable `$patient`. We can therefore define broadcast and multi-cast communications. Furthermore, our example will scale when more than two agents are present in the scene.

The semantics of message passing in our language corresponds to reliable, buffered, non-blocking communication. Sending a message will succeed immediately if an agent matches the definition, and the message M will be stored in a buffer on the recipient. Receiving a message involves an additional unification step. The message M supplied in the definition is treated as a template to be matched against any message in the buffer. For example, in line 6 of the protocol, a message must match `accept(appointment, $appointment)`, and the variable `$appointment` will be bound to the second term in the message if the match is successful. Sending a message will fail if no agent matches the terms, and receiving a message will fail if no message matches the message template.

Communication is non-blocking in that the send and receive actions do not delay the agent. For this reason, all of the receive actions are wrapped by `waitfor` loops to avoid race conditions. For example, in line 17 the agent will loop until a message is received. If this loop were not present the agent may fail to find an appointment request and the protocol would terminate prematurely. The advantage of non-blocking communication is that we can check for a number of different messages. For example, in lines 9 through 12 of the protocol the agent waits for either a refer or norefer decision. The `waitfor` loop includes a `timeout` condition which is triggered after a certain interval has elapsed. This can be useful in handling certain kinds of failures, though we do not make use of timeouts in our example.

At various points in the protocol, an agent is required to perform tasks, such as making a decision, or retrieving some information. This is done through the use of decision procedures. As stated earlier, decision procedures provide an interface between the dialogue protocol and the rational processes of the agent. In our language, a decision procedure p takes a number of terms as arguments and returns a single result variable v . The actual implementation of the decision procedure is external to the dialogue protocol. In effect, the decision procedure acts as a hook between the dialogue and the rational processes. For example, the `makeAppointment` decision procedure in line 18 of the dialogue refers to an external appointment procedure, which can be arbitrarily complex (e.g. a timetabling application).

The operations in the protocol are sequenced by the `then` operator which evaluates op_1 followed by op_2 , unless op_1 involved an action which failed. The failure of actions is generally handled by the `or` operator. This operator is defined such that if op_1 fails, then op_2 is evaluated, otherwise op_2 is ignored. For example, if the `doReferral` procedure in line 22 fails, then a `norefer` message will be sent in line 24. Our language also includes a `par` operator which evaluates op_1 and op_2 in parallel. This is useful when an agent is involved in more than one action simultaneously, though we do not use this in our example.

External data is represented by constants c in our language. We do not attempt to assign types to this data, rather we leave the interpretation of this data to the decision procedures. For example, in line 7 the symptoms are returned by the `getSymptoms` procedure, and interpreted by the `doReferral` procedure in line 22. Constants can therefore refer to complex data-types, e.g. flat-file data, XML documents, images.

It should be clear that MAP is a powerful language for expressing multi-agent dialogue. We have used this language to specify a wide range of protocols, including a range of popular negotiation and auction protocols. It is important to note that MAP is not intended to be a general-purpose language, and therefore the relative paucity of features (e.g. no user-defined data-types) is entirely appropriate. Nonetheless, dialogue protocols are executable, given a suitable communication platform and an appropriate definition of the decision procedures. In the following section we define the semantics of the language formally as the basis for an implementation.

3 Semantics of MAP

The provision of a clean and unambiguous semantics for our MAP language was a primary consideration in the design of the language. The purpose of the semantics is to formally describe the meaning of the different language constructs, such that dialogue protocols expressed in the language can be interpreted in a consistent manner. We consider this to be a failing of the formal semantics of FIPA [FIP99], which is expressed in BDI logic. The FIPA semantics is an abstract description, which neglects practical aspects such as a definition of the communication primitives. Furthermore, the BDI modalities can be interpreted in a number of different ways, e.g. [RG95, Jen93], meaning that implementations of BDI agents have typically been *ad-hoc* in nature.

$\Delta \in \text{Agent Environment}$	$::=$	$a \xrightarrow{\text{map}} (r, AE, VE, PE, ME^{(k)})$
$AE \in \text{Agent Protocols}$	$::=$	$\phi^{(k)} \xrightarrow{\text{map}} op$
$VE \in \text{Variables}$	$::=$	$v \xrightarrow{\text{map}} \phi$
$PE \in \text{Decision Procedures}$	$::=$	$p \xrightarrow{\text{map}} \phi^{(k)}$
$ME \in \text{Messages}$	$::=$	(a, r, M)

Figure 3: MAP Evaluation Environment.

We have chosen to present the MAP semantics in a relational operational semantics formalism called *natural semantics* [Kah87], so called because the evaluation of the relations is reminiscent of natural deduction. The natural semantics style is convenient because the entire evaluation of an agent dialogue can be captured within a (semi-)compositional derivation that can be reasoned about inductively. The rules of the semantics can be implemented directly (e.g. as Prolog Horn clauses) and a derivation can be performed incrementally (depth-first) from the root to the leaves. In natural semantics, we define relations between the initial and final *states of program fragments*. A program fragment in MAP is either an operation op , or an action α . The state is captured by an agent environment Δ which is defined in

$$\boxed{\Delta, a \vdash op \Rightarrow \Delta'}$$

$$(1) \quad \frac{\Delta, a \vdash \alpha \Rightarrow \Delta'}{\Delta, a \vdash \alpha \Rightarrow \Delta'}$$

$$(4) \quad \frac{\Delta, a \vdash op_2 \Rightarrow \Delta'}{\Delta, a \vdash op_1 \text{ or } op_2 \Rightarrow \Delta'}$$

$$(2) \quad \frac{\Delta, a \vdash op_1 \Rightarrow \Delta' \quad \Delta', a \vdash op_2 \Rightarrow \Delta''}{\Delta, a \vdash op_1 \text{ then } op_2 \Rightarrow \Delta''}$$

$$(5) \quad \frac{\Delta, a \vdash op_1 \Rightarrow \Delta' \quad \Delta, a \vdash op_2 \Rightarrow \Delta''}{\Delta, a \vdash op_1 \text{ par } op_2 \Rightarrow \Delta' \cup \Delta''}$$

$$(3) \quad \frac{\Delta, a \vdash op_1 \Rightarrow \Delta'}{\Delta, a \vdash op_1 \text{ or } op_2 \Rightarrow \Delta'}$$

$$(6) \quad \frac{\Delta(a) = (r, AE, VE, -, -) \quad VE \vdash subst(\phi_1^{(k)}) \Rightarrow \phi_2^{(k)} \quad \exists \phi_3^{(k)} \in AE \mid \{\emptyset \vdash unify(\phi_3^{(k)}, \phi_2^{(k)}) \Rightarrow VE'\} \quad \Delta \cup VE', a \vdash op \Rightarrow \Delta'}{\Delta, a \vdash agent(\phi_1^{(k)}) \Rightarrow \Delta'}$$

$$\boxed{\Delta, a \vdash \alpha \Rightarrow \Delta'}$$

$$(7) \quad \frac{}{\Delta, a \vdash \epsilon \Rightarrow \Delta}$$

$$(8) \quad \frac{\Delta(a) = (r, -, VE, PE, -) \quad VE \vdash subst(\phi_1^{(k)}) \Rightarrow \phi_2^{(k)} \quad VE \vdash unify(PE(p), \phi_2^{(k)}) \Rightarrow VE' \quad VE' \vdash eval(p, v) \Rightarrow VE''}{\Delta, a \vdash v = p(\phi_1^{(k)}) \Rightarrow \Delta \cup VE''}$$

$$(9) \quad \frac{\Delta(a) = (r, -, VE, -, -) \quad VE \vdash subst(\phi_1^{(k)}) \Rightarrow \phi_3^{(k)} \quad VE \vdash subst(\phi_2^{(2)}) \Rightarrow \phi_4^{(2)} \quad \forall a' \in \Delta \mid \left\{ \begin{array}{l} \Delta(a') = (r', -, VE', -, ME'^{(k)}) \\ \emptyset \vdash unify(\phi_4^{(2)}, (a', r')) \Rightarrow \emptyset \end{array} \right\}}{\Delta, a \vdash \rho_1(\phi_1^{(k)}) \Rightarrow agent(\phi_2^{(2)}) \Rightarrow \Delta(a') \cup (a, r, \rho_1(\phi_3^{(k)}))}$$

$$(10) \quad \frac{\exists ME \in \Delta(a) \mid \left\{ \begin{array}{l} ME = (a', r', \rho_2(\phi_3^{(k)})) \\ \emptyset \vdash unify((a', r'), \phi_2^{(2)}) \Rightarrow VE \\ VE \vdash unify(\phi_1^{(k)}, \phi_3^{(k)}) \Rightarrow VE' \end{array} \right\}}{\Delta, a \vdash \rho_1(\phi_1^{(k)}) \Leftarrow agent(\phi_2^{(2)}) \Rightarrow (\Delta - ME) \cup VE'}$$

$$VE \vdash subst(v) \Rightarrow VE(v)$$

$$VE \vdash unify(-, \phi) \Rightarrow VE$$

$$VE \vdash subst(\phi) \Rightarrow \phi$$

$$VE \vdash unify(\phi, \phi) \Rightarrow VE$$

$$VE \vdash subst(\phi^{(k)}) \Rightarrow$$

$$VE \vdash unify(v, \phi) \Rightarrow VE[v \mapsto \phi]$$

$$(subst(\phi^1), \dots, subst(\phi^k))$$

$$VE \vdash unify(\phi_1^{(k)}, \phi_2^{(k)}) \Rightarrow$$

$$unify(\phi_1^1, \phi_2^1) \cup \dots \cup unify(\phi_1^k, \phi_2^k)$$

Figure 4: MAP Operational Semantics.

Figure 3. The environment contains an n-tuple for each agent comprising the agent role r , the agent protocols AE , the bound variables VE , the decision procedures PE , and a message queue ME . The agent protocols AE map from arguments $\phi^{(k)}$ to operations op , where an empty sequence of arguments is the initial agent protocol. The decision procedures PE are represented as a map from the procedure name p to the argument terms $\phi^{(k)}$. The message queue $ME^{(k)}$ is a sequence of n-tuples (a, r, M) , where a and r are the name and role of the sender, and M is the actual message. For brevity we omit the rules for constructing the initial environment, and for checking well-formedness of the environment from our definition.

We define the evaluation rules for the program fragments of MAP in Figure 4. To capture the exchange of messages between agents we assume that the environment Δ is shared between agents. Thus, sending a message to an agent is captured by placing the message into the message queue ME of the recipient. Rules 1 through 6 define the evaluation of the different types of operations op . The form of these rules is $\Delta, a \vdash op \Rightarrow \Delta'$, where Δ is the state at the start of evaluation, a is the name of the agent performing the evaluation, op is the operation, and Δ' is the state on completion. Similarly, rules 7 through 10 capture the evaluation of the actions α . The form of these rules is $\Delta, a \vdash \alpha \Rightarrow \Delta'$, which is as before where α is the action. We also define a substitution function, $VE \vdash subst(\phi) \Rightarrow \phi'$ which substitutes variables for their values, and a unification function $VE \vdash unify(\phi_1, \phi_2) \Rightarrow VE'$ which matches terms and binds variables to values. The $VE \vdash eval(p, v) \Rightarrow VE'$ function evaluates the external decision procedure p , binding the result to v in VE' .

4 Conclusions and Further Work

In this paper we have defined a novel language for representing dialogue protocols in Multi-Agent Systems. Our language of multi-agent dialogue protocols (MAP) fills an essential gap between the low-level communication and high-level reasoning processes found in such systems. The language is founded on process calculus and is expressive enough to describe a large range of agent protocols.

Dialogue protocols specified in the MAP language are designed to be directly executable by the agents participating in the dialogue. To this end we have presented an operational semantics for the language, which precisely defines the evaluation behaviour of the language. Our presentation in the natural semantics style enables a direct implementation of the evaluation rules of the language. We have implemented these rules directly as Prolog Horn clauses using LINDA for inter-agent communication. We have also implemented the MAP language in Java using concurrent threads for the individual agents, and an interpreter which provides the necessary back-tracking and unification behaviour.

Dialogue protocols specify complex asynchronous and concurrent interactions, and therefore it is difficult to design correct protocols. Our experience with defining protocols in MAP has shown that predicting undesirable behaviour is a non-trivial task. To address this issue we are currently investigating the use of model-checking techniques [CGP99] to perform automated verification. The appeal of this approach over simulation is that an exhaustive exploration of the dialogue space is performed. Our early experiments with this technique have shown a high success rate in the detection of failures (e.g. non-termination) in our dialogues.

References

- [CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
- [CL90] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. *Intentions in Communication*, pages 221–256, 1990.
- [ERS⁺01] M. Esteva, J. A. Rodríguez, C. Sierra, P. Garcia, and J. L. Arcos. On the Formal Specification of Electronic Institutions. In *Agent-mediated Electronic Commerce (The European AgentLink Perspective)*, number 1991 in Lecture Notes in Artificial Intelligence, pages 126–147, 2001.
- [FIP99] FIPA Foundation for Intelligent Physical Agents. *FIPA Specification Part 2 - Agent Communication Language*, April 1999. Available at: www.fipa.org.
- [GHB99] M. Greaves, H. Holmback, and J. Bradshaw. What is a Conversation Policy? In *Proceedings of the Workshop on Specifying and Implementing Conversation Policies, Autonomous Agents '99*, Seattle, Washington, May 1999.
- [Jen93] N. R. Jennings. Specification and Implementation of a Belief-Desire-Joint-Intention Architecture for Collaborative Problem Solving. *Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.
- [Jen00] N. R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 117(2):277–296, 2000.
- [Kah87] G. Kahn. Natural Semantics. In *Proceedings of the Fourth Annual Symposium on Theoretical Aspects of Computer Science (STACS'87)*, number 247 in Lecture Notes in Computer Science, pages 22–39, Passau, Germany, February 1987. Springer-Verlag.
- [MC02] N. Maudet and B. Chaib-draa. Commitment-based and Dialogue-game based Protocols – New Trends in Agent Communication Language. *The Knowledge Engineering Review*, 17(2):157–179, 2002.
- [Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989.
- [RG95] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pages 312–319, San Francisco, USA, June 1995. AAAI Press.
- [RG98] A. S. Rao and M. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–344, 1998.
- [Sin98] M. P. Singh. Agent Communication Languages: Rethinking the Principles. *IEEE Computer*, pages 40–47, December 1998.
- [Woo00] M. Wooldridge. Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 3(1):9–31, 2000.