Modeling Agent Negotiation

J. Sprinkle, C. P. van Buskirk and G. Karsai
Vanderbilt University
Nashville, TN, 37203

Abstract

A Multi-Agent System (MAS) is a cooperation focused implementation of multiple programs (agents) that coordinate with each other to attempt to converge on the solution to one or more tasks. Agent negotiation is the convergence upon this solution through compromise and communication. Currently, the implementation of agents is highly dependent on the programming language, and any perspective to the negotiation methods agents use to achieve goals and tasks are drawn after the implementation phase of the development. A better solution to the development of agent interaction is to model the negotiation interaction on a high level, and produce from that model the implementation. Model-Integrated Computing (MIC) and Model-Integrated Program Synthesis (MIPS) are two tools which may be used toward the implementation of such a method.

1 Introduction to Agents

Before any discussion of agents, there should exist a somewhat elementary understanding of what an agent is, and what capabilities an agent has. It is almost impossible to find one particular definition of the term "Agent." A quick reference to several documents could yield as many definitions as there are documents, not to mention documents that examine the fact that there is no hard and fast definition [6][7][12]. Fortunately, this paper is concerned with the highest levels of agent interaction, so the set of attributes of an agent for this topic is fairly small.

1.1 Abstract Layers of an Agent

In the substance of an agent, several conceptual layers of information exist [1]. The agent knows about things (its domain, consisting of objects defined in terms of its ontology), can perform tasks, can communicate, etc. For this paper, the most important aspect of agent existence is in its communication with other agents. All other levels are implementation details.

More importantly, the world outside an agent communicates with it through its communication layer. This implies that the agent itself is responsible for interpreting messages it receives, and reacting to them.

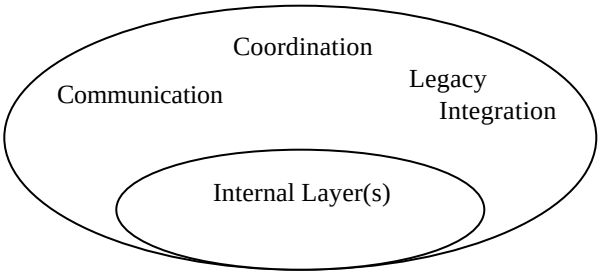

Figure 1 - Conceptual drawing of an agent [1]. The internal workings of the agent are accessed by the outer layer through API calls.

Figure 1 shows a conceptual view of an agent at the highest level of abstraction. The communication portion of this outer layer is the most important with regard to negotiation. Coordination and legacy integration are important for the statically determined behavior of the agent, and thus are not considered or used during negotiation. This view of an agent is consistent with that set forth by the Foundation for Intelligent Physical Agents (FIPA) [5].

1.2 Target Application

Although there exist standards for the format of messages in agent communication [5], much of the implementation in general is nonstandard. Not only are the internal workings of an agent unspecified, but they are implemented in different languages. Even with the acceptance of the FIPA notion that agents must have communication, coordination, and legacy integration aspects, there still exists no standard for their implementation.

However, many agent systems, though they differ in implementation, are conceptually the same. This is the solution point: that by conceptually modeling a process, the output can be an implementation. The MIC and MIPS tools have been proven to provide such solutions [8][9][10][11][14], and are the tools used in the solution of this problem.

In a nutshell, the idea of MIPS is to define concepts of the domain, then model the problem using these concepts. After the problem is modeled, then a custom program

interprets the model, and produces the compilable code that implements the solution to the problem.

The Generic Modeling Environment (GME), developed at Vanderbilt University, was the modeling environment used to define the domain concepts, or paradigm. The negotiation paradigm allowed GME to become a domain-specific modeling environment in the agent negotiation domain. GME was also used to construct the actual models of the negotiation.

The objective of negotiation modeling is to model the agent negotiation domain well enough such that as many target environments as possible can use the same domain concepts, and then to create a custom program for each target environment. In theory, one would be able to generate solutions for any agent environment with the same paradigm. To properly understand the negotiation domain, therefore, let us examine aspects of agent behavior and negotiation.

2 Agent Behavior

Negotiation between agents is captured in the behavior of the agent with regard to stimuli. In some respects, therefore, the concepts of agent behavior are a subset of the concepts of agent negotiation, and thus behavioral concepts must be identified in order to be able to model them. The behavior of an agent may be viewed conceptually in two categories: what it does internally, and how it responds to external stimuli.

2.1 Intra-Agent Behavior

Modeling the behavior inside the agent is outside the scope of this paper. First of all, each target application's definition of an agent has different internal capabilities, so by that argument it would not be possible to produce implementations for all target applications because those individual applications would have to be included in the modeling environment.

However, the need to define internal behaviors during negotiation is recognized and accounted for. One solution is to define a concept within the modeling environment that allows the modeler to input custom implementation code. While this removes the ability of the MIPS environment to produce a solution for any agent implementation from this *instance* of models, the modeling environment *itself* is still implementation independent. This limitation is discussed further in section 5.

2.2 Inter-Agent Behavior

With regard to negotiation, the behavior of an agent while dealing with other agents is the more interesting of the two types of behaviors.

Agents communicate with each other through messages, and the sending and receiving of these messages are the two main concepts of inter-agent behavior. Two generally accepted standards for messaging are KQML [4] and the FIPA-ACL [5]. Unfortunately, these standards provide only the conceptual structure of the message, and not the actual implementation syntax of the text string. However, they do specify certain performatives and parameters. For this particular modeling environment, the FIPA-ACL is implemented as the standard for describing message sending and receiving.

3 Agent Negotiation

Now that the domain concepts of internal and external behavior are defined, the next step in defining the model is to expand the set of behavior concepts with the concepts of the actual process of negotiation. The **bolded** names in parentheses denote the name by which the modeling paradigm refers to these concepts.

3.1 Negotiation State

One way to model negotiation (or for that matter, any type of behavior) is with a state machine. Before continuing on this line of discussion, it is important to differentiate between the state of the negotiation, and the state of the agent. The negotiation state is determined by the last message received and its content, while the agent state is a function of the internal values of the agent's variables.

There are several implementations of agent frameworks that use state machines to model behavior [1][2][15], but none of these actually provide an agent platform independent solution: they either solve the problem for that particular agent environment, or give a general mapping with no ability to provide implementation.

Therefore, a state machine framework was designed that used as states and transitions the concepts from intra-agent and inter-agent behavior. The framework follows the concept of a Mealy state machine [16], with the states being the concept of waiting for a message, and the transitions being the concept of sending a message or performing some action within the state of the agent.

3.2 Possible Negotiation Transitions (Actions)

The agent action which furthers the negotiation is the sending of a message (**send**). The message sent by the negotiating agent is directly related to the last message received (which likely gave some indication as to the next state of the negotiation) and the current internal state of the agent. The attributes of such an action are limited to the attributes of a FIPA-ACL message, which are available in [5].

The concept of modification of the internal state of the agent (**action**) is an elusive concept to model in general. The most generic model of this action is to allow direct input of code that will translate into a subroutine or method of the agent (implemented through its API) that returns a state indication variable which is then used to make a change in the state of the negotiation. Each action must return some value (defined at model building time), which is returned from the subroutine and used to determine the path to take next.

3.3 Possible Negotiation States (Waiting)

The only concept of waiting in agent negotiation is that of waiting on the receipt of a message (**receive**). Arguably, more sophisticated agent implementations may obey interrupts from the underlying agent architecture, but we will assume that this is handled through the API and is transparent to the negotiation state machine.

It is possible to wait on as many types of messages as a send message action could send (in accordance with the FIPA-ACL). The transitions extending from the wait message therefore have the same properties as a send message action. There is no current way to visualize the properties of the connections without a dialog box (due to a GME limitation that will no be present in the new release), so the names under the "Receive" atoms serve as mnemonics for the connection types.

3.4 Roles In Negotiation Protocols

There are two distinct roles in any interaction protocol: that of *initiator*, and that of *responder*. Take for instance, the simple protocol of two persons who meet on the street, say, Jon, and Mary.

> Jon: "Hello, how are you?"
> Mary: "Fine. How are you?"
> Jon: "Just fine."

Then, Jon and Mary resume their paths, or decide to converse further. Jon and Mary both knew when this portion of the conversation was over, because they had a notion of whether they *initiated* or *responded to* the conversation. Consider the following protocol, without this knowledge.

> Jon: "Hello, how are you?"
> Mary: "Fine. How are you?"
> Jon: "Just fine. How are you?"
> Mary: "Fine. How are you?"
> Jon: "Just fine. How are you?"
> Mary: "Fine. How are you?"

There would be no end to the conversation. This example, albeit quite simple, illustrates the need for negotiating parties to understand their roles in the ongoing protocol.

4 Implementation

Now that all the concepts of agent negotiation and behavior as it pertains to negotiation are recognized, it is time to formally define them in terms of a model. A modeling paradigm was developed which embodies these concepts and defines a visual language for the expression of the negotiation state machine. For a legend of the modeling concepts used and their representative icons, see Figure 2.

Figure 2 - Legend of the parts used when building the model of a protocol. The Send, Receive, Action and DefaultAction parts all play states or transitions in the state machine, while the Succeed and Fail parts play roles in the exit value of the protocol.

4.1 Example Protocol: Contract Net

The contract net protocol is a high-level protocol for communication among conceptually distributed objects [13]. In the contract net (CN), the initiator issues calls for proposal (cfp's), and the responder decides whether to issue return bids in the form of proposals. This negotiation continues until the initiator decides that the proposal is acceptable, or that the proposal is unacceptable, and that it will no longer accept new proposals. For a graphical representation of the initiator and responder, please refer to Figure 3 and Figure 5.

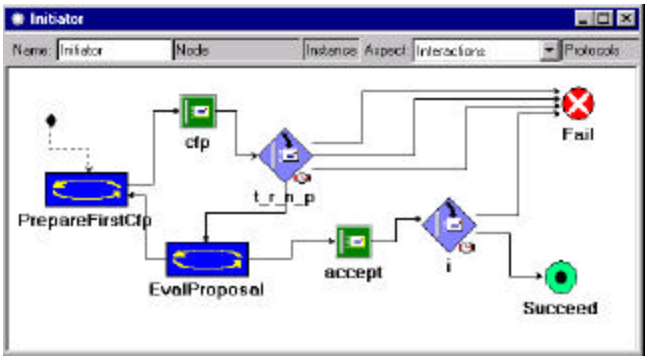

Figure 3 - A model of the initiator role of the contract net protocol. Attributes of the parts determine the types of messages they send or receive.

The default action icon plays the role of determining which of the states is the initial state of the negotiation state machine for the initiator/responder. The default action can point to either an action, or a state.

Note the "EvalProposal" icon, and that connections go to the "accept" and "PrepareFirstCfp" icons. Those connections bear names that directly correspond to the values set in Figure 4.

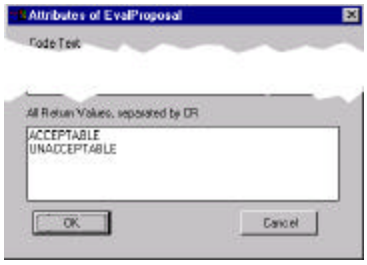

Figure 4 - The exit connections of the "EvalProposal" action contain values that correspond exactly to the possible return values from this segment of code. The code is specified in the API of the agent framework (a portion of the dialog was omitted for brevity).

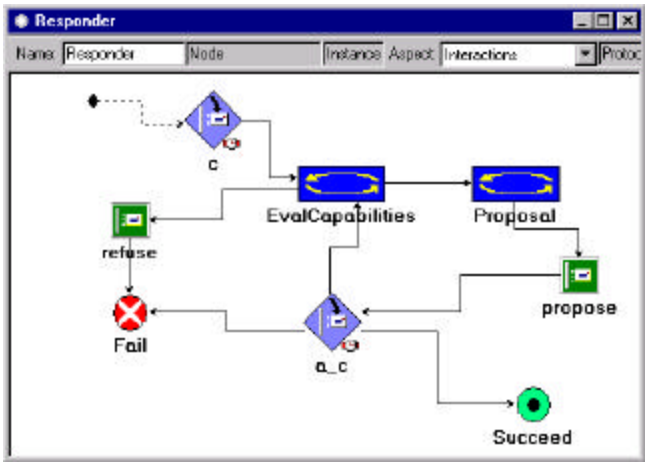

Figure 5 - A model of the responder role of the CN protocol.

4.2 Verification Methods and Constraints

In theory each receive transition in one role (either initiator or responder) should have a corresponding send in the other role. Otherwise, the protocol could end up in a state from which it would have no exit.

One solution to this problem is to allow action matching on a higher level. Figure 6 shows the initiator and responder roles and the high level expectations that one role has from the other. It is the responsibility of the modeler to take advantage of this feature of the modeling environment; it is not strictly enforced.

The reason for this is that it is not an error to have receives that do not match to a send. For example, this particular implementation of CN requires a message from the initiator agent to itself (an inform message), and therefore there is no matching send from the responder. Hence, this ability to match aids in omission errors, but does not restrict the modeler's ability to custom craft the agent negotiation behavior.

However, model constraints are enforced by GME, and are expressed when the paradigm is defined. One of the constraints of this modeling environment is that exactly one default action is allowed per initiator/responder, and it must connect to one and only one atom. Another is that a send atom must have exactly one connection coming out of it (to ensure that the state machine is deterministic).

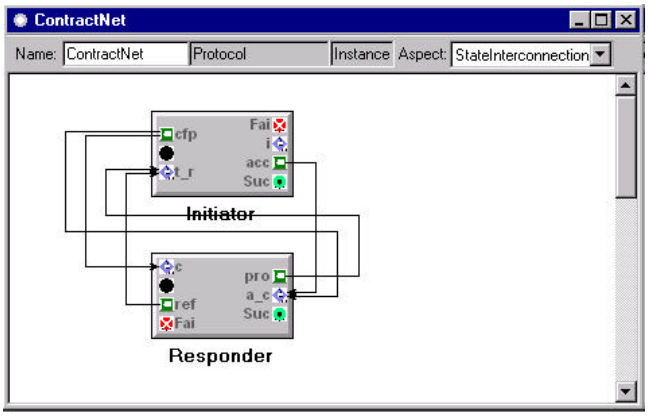

Figure 6 - Matching the sends and receives in the high level views of negotiation roles.

4.3 MIPS Output and Model Interpreter

As discussed in the Introduction, the objective of modeling agent negotiation using MIPS is to visually lay out the negotiation, and produce from that visual language compilable code for an agent implementation. Just as text based programming languages have a compiler that translates the code into executable binary files, the graphical modeling language for this paradigm has a similar compiler, called an interpreter. This interpreter examines the context and syntax of the models and then produces the output desired by the interpreter writer. Interpreter writing is done through a text language by accessing the API of the GME.

For this particular paradigm, the output of the interpreter is compilable code files that describe the agent negotiation process. While many different agent packages may be modeled using the same paradigm, a different interpreter must be written for each agent package. This is because the syntax and semantics of the model are interpreted differently by each agent package. For example, one agent package may format all of its messages with XML, while another may define its own tag based language. By putting the syntax definition in the interpreter, the modeler need only define the semantics.

The particular target application for which this solution was tested was the Zeus Agent Building Tool-Kit [3]. An interpreter was written and tested for the protocol paradigm which outputted Java files that implemented the interfaces necessary for a Zeus negotiation protocol. In the CN example, two Java classes were created, one for the initiator, and one for the responder. In order to incorporate the CN protocol in the agent, the agent must be configured (using the agent platform tools, in this case Zeus) to interface with the classes. Once the classes are compiled and "plugged-in" to the agent definition, then the construction phase is complete.

5 Future Work

Modeling the negotiation when dealing with the inter-agent interaction gives the ability to model the high level interactions of an agent easily. However, some subtle behaviors of a negotiation are still left in textual form. The major one of these is the concept of a negotiation strategy. Currently, negotiation strategy is defined by using the action icon, and typing in the code that implements it. "EvalProposal" is an example of an icon in the initiator role of the CN that represents a strategy.

The negotiation strategy is intra-agent behavior that determines by how much an agent may increase its bid, or whether an agent is willing to sell some of its resources in order to obtain enough money to purchase some other good. In the future, the concepts of the negotiation strategy domain (e.g. increment-bid, linear-increase, exponential-increase) would be a welcome addition to the negotiation modeling paradigm.

Another future item is the ability to produce code for any agent environment from the same instance of models. One solution to allow this is to create a generic API that would execute standard methods in the target agent domain. Then, all actions could be coded using this API instead of the agent platform's API, and each agent platform interpreter could implement the interface defined by the API.

6 Conclusions

Both MIC and MIPS yield benefits when used to create negotiation protocols. When considering the basic contributions of MIC, visualization allows for easy documentation, and also makes it easy for a domain expert (whether a programmer or not) to lay out his concept of negotiation without ambiguity. However, the MIPS offerings are more numerous, and fall into three major contributions.

The first is that the programmer is not forced to think "in the small" through constructing software state machines, and looking up return types, etc. because he may visually lay out the model of the negotiation, and produce from it the necessary code using the interpreter.

Secondly, the modeler is given the tool to see the interaction of the initiator and responder roles, which can prevent the development of a protocol that has an inescapable state.

Thirdly, if several negotiation protocols were developed for a particular agent package, and the agent programmer wanted to also implement the protocols for another agent environment, then the same modeling paradigm could be used to do it. This allows the programmer to write code only once (for the interpreter), which then would produce the state machine in the target environment. After reconfiguring the internal actions of the negotiation in accordance with the API of the target environment, the models would be ready for complete generation.

Acknowledgements

This work was sponsored by the Defense Advanced Research Projects Agency, as part of the Autonomous Negotiating Teams project, under contract #F30602-99-2-0505.

References

[1] M. Barbuceanu, M. S. Fox, "Capturing and Modeling Coordination Knowledge for Multi-Agent Systems," *International Journal of Cooperative Information Systems*, Vol 5, No. 2, pp. 275-314, 1996.

[2] L. Bölöni, D. C. Marinescu, "A Multi-Plane State Machine Agent Model," *Fourth International Conference on Autonomous Agents*, Jun. 1999.

[3] J. Collis, D. Ndumu, H. Nwana, L. Lee, "The Zeus Agent Building Tool-Kit," *BT Technology Journal*, Vol. 16, No. 3, pp. 60-68, Jul. 1998.

[4] T. Finin, et al., "Specification of the KQML Agent Communication Language," The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1992.

[5] Foundation for Intelligent Physical Agents, "FIPA 97 Specification," Part 1, Ver. 2.0, Oct. 1998.

[6] S. Franklin, A. Graesser, "Is It an Agent, Or Just a Program? A Taxonomy for Autonomous Agents," *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.

[7] N. R. Jennings, M. Wooldridge, "Software Agents", *IEE Review*, pp. 17-20, Jan. 1996.

[8] G. Karsai, F. DeCaria, "Model-Integrated On-line Problem-Solving Environment for Chemical

Engineering," *IFAC Control Engineering Practice*, Vol. 5, No. 5, pp. 1-9, 1997.

[9] G. Karsai, J. Sztipanovits, S. Padalkar, C. Biegl, "Model Based Intelligent Process Control for Cogenerator Plants," *Journal of Parallel and Distributed Systems*, pp. 90-103, 1992.

[10] E. Long, A. Misra, J. Sztipanovits, "Increasing Productivity at Saturn," *IEEE Computer Magazine*, August, 1998.

[11] A. Misra, G. Karsai, J. Sztipanovits, "Model-Integrated Development of Complex Applications," Proceedings of the Fifth International Symposium on Assessment of Software Tools, pp. 14-23, Pittsburgh, PA, June, 1997.

[12] H. Nwana, "Software Agents: An Overview," *Knowledge Engineering Review Journal*, Vol. 11, No. 3, pp. 205-234, Nov. 1996.

[13] R. G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Transactions on Computers*, Vol. C-29, No. 12, Dec. 1980.

[14] J. Sztipanovits, G. Karsai, "Model-Integrated Computing," *IEEE Computer*, pp. 110-112, April, 1997.

[15] Y. Tahara, A. Ohsuga, S. Honiden, "Agent System Development Method Based on Agent Patterns", *Proceedings of the 21st International Conference on Software Engineering*, ACM Press, pp.356-367, 1999.

[16] J. Wakerly, *Digital Design Principles and Practices*, 2nd edition, p. 468, Prentice Hall, 1994.