Modeling Multiagent Systems with CASL—A Feature Interaction Resolution Application[*]

Steven Shapiro[a] Yves Lespérance[b]

[a] Department of Computer Science, University of Toronto
Toronto, ON M5S 3G4, Canada
steven@ai.toronto.edu

[b] Department of Computer Science, York University
Toronto, ON M3J 1P3, Canada
lesperan@cs.yorku.ca

Abstract. In this paper, we describe the Cognitive Agents Specification Language (CASL), and exhibit its characteristics by using it to model the multiagent feature interaction resolution system described by Griffeth and Velthuijsen [7]. We discuss the main features of CASL that make it a useful language for specifying and verifying multiagent systems. CASL has a nice mix of declarative and procedural elements with a formal semantics to facilitate the verification of properties of CASL specifications.

1 Introduction

The use of proper design methods is just as important for multiagent systems as for non-agent-based software. In this paper, we present a formal specification language for multiagent systems called the Cognitive Agents Specification Language (CASL). CASL combines a theory of action [13] and mental states [14, 15] based on the situation calculus [11] with ConGolog [6], a concurrent, nondeterministic programming language that has a formal semantics. The result is a specification language that contains a rich set of operators to facilitate the specification of *complex* multiagent systems. Specifications in this language can exploit the higher level of abstraction that comes from expressing constraints in terms of mental attitudes.

An earlier version of CASL was described in [16], where the use of the formalism was illustrated with a simple meeting scheduling multiagent system example. The motivation for our approach was discussed in further detail there. In this paper, we extend the formalism to support communication with encrypted speech acts and incorporate a simpler account of goals. But foremost, we use the formalism to model a somewhat more complex multiagent system taken from the literature. In [7], Griffeth and Velthuijsen

[*] This research was funded by Communications and Information Technology Ontario and the Natural Science and Engineering Research Council of Canada. We thank Fawzi Daoud for suggesting that we look at feature interaction applications and Griffeth and Velthuijsen's work in particular.

present a feature interaction resolution system for telecommunication applications that involves negotiating, autonomous agents with explicit goals. This is an attractive approach in the context of rapidly expanding telecommunication services, open environments, and the need for client customization. In CASL, agents' goals (and knowledge) can be represented explicitly, making CASL an ideal formalism for specifying systems such as the one described by Griffeth and Velthuijsen. They informally state a property of their system and (also informally) show that the property holds for their system. Since CASL is a formal language, it is possible to *formally* state and prove such properties. We use Griffeth and Velthuijsen's system as an example of the modeling capabilities of CASL, and give a formal statement of the property discussed by Griffeth and Velthuijsen.

2 The Model of Domain Dynamics

In CASL, a dynamic domain is represented using an action theory [13] formulated in the situation calculus [11], a predicate calculus language for representing dynamically changing worlds. A situation represents a snapshot of the domain. There is a set of initial situations corresponding to the ways the agents believe the domain might be initially. The actual initial state of the domain is represented by the initial situation constant, S_0. The term $\mathit{do}(a, s)$ denotes the unique situation that results from the agent performing action a in situation s. Thus, the situations can be structured into a set of trees, where the root of each tree is an initial situation and the arcs are actions. The initial situations are defined as those situations that do not have a predecessor: $\mathit{Init}(s) \overset{\text{def}}{=} \neg\exists a, s'.s = \mathit{do}(a, s')$.

Predicates and functions whose value may change from situation to situation (and whose last argument is a situation) are called *fluents*. For instance, in our model of Griffeth and Velthuijsen's system, we use the fluent $\textsc{TwoWayIP}(x, y, s)$ to represent the property that a two-way phone connection is in progress between agents x and y. The effects of actions on fluents are defined using successor state axioms [13], which provide a solution to the frame problem; see below for an example.

There is a distinguished predicate $\mathit{Poss}(a, s)$, denoting that action a is executable in situation s. $s \preceq s'$ means that s' results from performing a (possibly empty) executable sequence of actions in s. We use a theory containing the following kinds of axioms [13] to specify a dynamic domain:

- Action precondition axioms, one for each action, which characterize *Poss*.
- Successor state axioms, one for each fluent.
- Initial state axioms, which describe the initial state of the domain and the initial mental states of the agents. These are axioms that only describe initial situations.
- Unique names axioms for the actions.
- Domain-independent foundational axioms (similar to the ones given by Lakemeyer and Levesque [9]).

2.1 Modeling the Environment

In the feature interaction resolution system, the agents are negotiating to establish telephone connections on behalf their human users. We consider two types of connections: a regular two-way telephone connection and a recording connection, during which one person leaves a message for another. These two types of connections are established by the actions $\textsc{twoWay}(x, y)$ (x sets up a two-way connection with y) and $\textsc{record}(x, y)$ (x records a message for y), resp. In addition, once a connection is established, the agent initiating the connection may identify its user to the other agent. The initiating agent may send its user's name or telephone number. The corresponding actions are: $\textsc{name}(x, y)$ and $\textsc{number}(x, y)$, resp. There is also the $\textsc{disconnect}(x, y)$ action, which terminates a connection between x and y.

Once a connection of a certain type is established, we say that the connection is in progress until the connection is terminated. For each of the connection and identification actions, we have a corresponding fluent that becomes true when the action is executed. An example of this is the $\textsc{TwoWayIP}(x, y, s)$ fluent discussed above. Similarly, we have the fluent $\textsc{RecordingIP}(x, y, s)$, which means that a recording is in progress between x and y in situation s. For the identification actions, we have the fluents $\textsc{NameIP}(x, y, s)$ and $\textsc{NumberIP}(x, y, s)$ whose intuitive meanings are that x's owner's name (number, resp.) is being displayed for y's owner in situation s. Any of these fluents that are true become false when the $\textsc{disconnect}(x, y)$ action is executed.

We must give precondition axioms and successor state axioms for these actions and fluents. For example, the precondition axiom for $\textsc{twoWay}$ is:

$$\mathit{Poss}(\textsc{twoWay}(x, y), s) \equiv \neg\textsc{TwoWayIP}(x, y, s) \wedge \neg\textsc{RecordingIP}(x, y, s).$$

The successor state axiom for $\textsc{TwoWayIP}$ is:

$$\begin{array}{l}\textsc{TwoWayIP}(x, y, \mathit{do}(a, s)) \equiv \\ \quad a = \textsc{twoWay}(x, y) \vee (\textsc{TwoWayIP}(x, y, s) \wedge a \neq \textsc{disconnect}(x, y)).\end{array}$$

As mentioned earlier, successor state axioms provide a solution to the frame problem. The other precondition and successor state axioms are similar to these, so we omit them.

2.2 Modeling Agents' Mental States

We model two aspects of the mental states of agents: knowledge and goals. These are represented with a possible worlds semantics in the situation calculus using situations as possible worlds, following Scherl and Levesque [14]. The accessibility relation for knowledge, $K(agt, s', s)$[1], holds if situation s' is compatible with what agt knows in situation s. An agent knows a formula ϕ, if ϕ is true in all the K-accessible worlds:

$$\mathbf{Know}(agt, \phi, s) \overset{\text{def}}{=} \forall s'(K(agt, s', s) \supset \phi[s']).$$

[1] Scherl and Levesque only consider a single agent; we [16] generalize the framework to handle multiple agents by adding an agent argument to the accessibility relations.

Here ϕ is a formula that may contain a free variable now. This variable is used as the situation argument of the fluents in ϕ. $\phi[s]$ denotes the formula that results from substituting s for now in ϕ. We also say that an agent knows whether ϕ holds if it either knows ϕ or its negation: **KWhether**$(agt, \phi, s) \overset{\text{def}}{=}$ **Know**$(agt, \phi, s) \vee$ **Know**$(agt, \neg\phi, s)$.

Scherl and Levesque formulated a successor state axiom for K that describes how an agent's knowledge is affected by actions, including expansion due to *sensing* actions. We [16] adapted the axiom to handle the $\text{INFORM}(informer, agt, \phi)$[2] action, i.e., $informer$ informs agt that ϕ holds. A limitation of the formalization is that all agents are aware of all actions, as in a broadcast model of communication. Here, we modify the representation of speech acts to accommodate *encrypted messages*. We model the encryption and decryption of messages using the functional fluents ENCODE and DECODE, resp. The value of $\text{ENCODE}(sender, rec, \phi, s)$ is a code, and the value of $\text{DECODE}(sender, rec, c, s)$—where c is a code— is a formula. These functions have to be fluents in order to be able to model the fact that only rec knows the value of $\text{DECODE}(sender, rec, c, s)$ and only $sender$ knows the value of $\text{ENCODE}(sender, rec, \phi, s)$. That is the only reason these functions are fluents as their values are unchanged by actions as shown in the following successor state axioms:

$$\begin{aligned}&\text{ENCODE}(sender, rec, \phi, do(a, s)) = \text{ENCODE}(sender, rec, \phi, s)\\&\text{DECODE}(sender, rec, c, do(a, s)) = \text{DECODE}(sender, rec, c, s)\end{aligned}$$

The content of messages will be codes instead of formulae, e.g., we will use $\text{INFORM}(informer, agt, c)$ instead of $\text{INFORM}(informer, agt, \phi)$.

We modify the successor state axiom for K to handle encrypted messages:

$$\begin{aligned}&K(agt, s'', do(a, s)) \equiv\\&\exists s'(K(agt, s', s) \wedge s'' = do(a, s') \wedge Poss(a, s') \wedge\\&\quad \forall informer, \phi, c(a = \text{INFORM}(informer, agt, c) \wedge\\&\qquad\qquad\qquad \textbf{Know}(agt, \phi = \text{DECODE}(informer, agt, c), s) \supset \phi(s'))).\end{aligned}$$

This axiom states the conditions under which a situation s'' will be K-accessible from $do(a, s)$. If a is not an INFORM action, then the predecessor of s'' (i.e., s') must be K-accessible from s and the action that takes s' to s'' must be a and executable in s'. If a is the action of $informer$ informing agt that c, where c is a code, and agt knows that ϕ is the decoding of c, then, in addition to the previous conditions, it must be the case that ϕ holds in s'. Thus, this axiom ensures that after any action, the agents know that the action has occurred and that it was executable, and if the action is an INFORM action, and the recipient of the message can decode the message, then the recipient knows that the decrypted content of the message holds. This axiom defines the K relation at non-initial situations. The K relation at initial situations is specified by the axiomatizer of the domain using initial state axioms, subject to the constraint that initial situations can only be K-related to other initial situations. This framework only handles knowledge

[2] Since we have functions and relations that take formulae as arguments, we need to encode formulae as first-order terms. For example, we could use the encoding given by De Giacomo et al. [6]. For notational simplicity, we suppress this encoding and use formulae as terms directly.

expansion. In [17], we give an account of belief revision that is compatible with this framework.

We model the goals of an agent using another accessibility relation on situations, $W(agt, s', s)$. This relation holds if s' is compatible with what the agent *wants* in s. Unlike with the K relation, we allow the W relation to relate situations that have different histories. The reason for this is that an agent may want things that do not currently hold, but that it wants to hold in the future. Therefore, we allow future situations to be among the W-accessible situations.

An agent may want something that it knows to be impossible to obtain, but we want the *goals* of the agent to be consistent with what the agent knows. Therefore, we define the goals of an agent to be those formulae that are true in all W-accessible situations that have a K-accessible situation in their past:

$$\begin{aligned}&\textbf{Goal}(agt, \psi, s) \stackrel{\text{def}}{=} \\ &\quad \forall now, then(K(agt, now, s) \wedge W(agt, then, s) \wedge now \preceq then \supset \psi[now, then]).\end{aligned}$$

Here ψ is a formula that has two free variables, now and $then$. $then$ can be thought of as defining a finite path of situations, namely, the sequence of situations up to situation $then$. now corresponds to the current situation along that path. In the definition of **Goal**, the K relation is used to pick out the current situation (now) along the path defined by the W-related situation ($then$) as well as to filter out the situations that are incompatible with what the agent knows to be the case. $\psi[s', s'']$ denotes the formula that results from substituting s' for now and s'' for $then$ in ψ.

The successor state axiom for W is similar to the one for K:

$$\begin{aligned}&W(agt, then, do(a, s)) \equiv [W(agt, then, s) \wedge \\ &\quad \forall requester, \psi, c, now(a = \text{REQUEST}(requester, agt, c) \wedge \\ &\qquad \textbf{Know}(agt, \psi = \text{DECODE}(requester, agt, c), s) \wedge K(agt, now, s) \wedge \\ &\qquad now \preceq then \wedge \neg\textbf{Goal}(agt, \neg\psi, s) \supset \psi[do(a, now), then)]].\end{aligned}$$

A situation $then$ is W-accessible from $do(a, s)$ iff it is W-accessible from s and if a is the action of $requester$ requesting agt that the decoding of c obtain, and agt knows that ψ is the decoding of c, and now is the current situation along the path defined by $then$, and the agent does not have the goal that $\neg\psi$ in s then ψ holds at $(do(a, now), then)$. If the agent gets a request for ψ and it already has the goal that $\neg\psi$, then it does not adopt the goal that ψ, otherwise its goal state would become inconsistent and it would want everything. This is a simple way of handling goal conflicts. It should be possible to cancel requests. A more sophisticated handling of conflicting requests will be presented in [15].

In order to execute an INFORM action, an agent must know how to encode the message and also know that the content of the message is true. Therefore, after receiving the message, the recipient of the message knows that the sender knew that the content of the message was true. Similarly, in order to execute a REQUEST action, an agent must know how to encode the message and not have any goals that conflict with the request. This is a somewhat simplistic model for these communicative acts, and we plan to refine it in the future. Here are the precondition axioms for INFORM and REQUEST:

$$\begin{array}{l} \mathit{Poss}(\text{INFORM}(informer, agt, c), s) \equiv \\ \quad \exists\phi.\mathbf{Know}(informer, (\text{ENCODE}(informer, agt, \phi) = c \wedge \phi), s). \\ \mathit{Poss}(\text{REQUEST}(reqr, agt, c), s) \equiv \\ \quad \exists\psi.\mathbf{Know}(reqr, \text{ENCODE}(reqr, agt, \psi) = c, s) \wedge \neg\mathbf{Goal}(reqr, \neg\psi, s). \end{array}$$

We also use the following definitions adapted from Lakemeyer and Levesque [9]. A formula ϕ describes *all* that agt knows initially:

$$\mathbf{OKnow}_0(agt, \phi) \overset{\text{def}}{=} \forall s'(K(agt, s', S_0) \equiv \mathit{Init}(s') \wedge \phi(s')).$$

A formula ψ describes *all* the paths that are consistent with agt's initial goals:

$$\begin{array}{r} \mathbf{OGoal}_0(agt, \psi) \overset{\text{def}}{=} \forall now, then.(K(agt, now, S_0) \wedge now \preceq then \supset \\ (W(agt, then, S_0) \equiv \psi[now, then])). \end{array}$$

We need to put constraints on the accessibility relations in order to yield mental attitudes with desirable properties. For example, we want positive and negative introspection of knowledge and of goals. Due to lack of space, we will not discuss the constraints that need to be placed on the K and W in order to yield these properties. Let us simply assume that these properties hold.

In our formalization of goals, goals are evaluated relative to finite paths. Thus, we cannot represent that an agent wants that ψ always be true because the path relative to which the proposition is evaluated ends, so there is no way of knowing whether ψ holds "after" the end of the path. However, we can model maintenance goals that are time bounded: ψ is true until time T. If T is chosen suitably far in the the future, time-bounded maintenance goals can replace unbounded maintenance goals. However, this requires adding a notion of time. We formalize time in the situation calculus as we did in [16]. That is, we add a functional fluent *time*(s) whose value is a natural number that represents the time at situation s. To simplify the formalization of time, here, we assume that all actions have duration of 1 and that the time at all initial situations is 1.

We will express maintenance goals using the predicate **Always**$(\psi, now, then)$, which says that ψ always holds, from now until the end of time, denoted by the constant T: **Always**$(\psi, now, then) \overset{\text{def}}{=} \text{TIME}(then) = \text{T} \wedge \forall s. now \preceq s \preceq then \supset \psi[s, then]$.

3 The Behavior Specification

We specify the behavior of agents with the notation of the process specification language ConGolog [6], the concurrent version of Golog [10]. We take a ConGolog program[3] to be composed of a sequence of procedure declarations, followed by a complex action. Complex actions are composed using the following constructs:

a,	primitive action
$\phi?$,	wait for a condition
$\delta_1 ; \delta_2$,	sequence
$\delta_1 \mid \delta_2$,	nondeterministic choice between actions
δ^*,	nondeterministic iteration

[3] We retain the term *program* even though it is not our intention to execute the programs directly.

if ϕ **then** δ_1 **else** δ_2 **endIf**,	conditional
for $x \in \Sigma$ **do** δ **endFor**,	for loop
while ϕ **do** δ **endWhile**,	while loop
$\delta_1 \parallel \delta_2$,	concurrency with equal priority
$\delta_1 \rangle\!\rangle\, \delta_2$,	concurrency with δ_1 at a higher priority
$\langle\, \boldsymbol{x} : \phi \rightarrow \delta \,\rangle$,	interrupt
$\beta(\boldsymbol{p})$,	procedure call.

a denotes a situation calculus primitive action, as described earlier. The ConGolog specification can be for a single agent or multiple agents, depending on whether the primitive actions contain an argument for the agent of the action. ϕ denotes a situation calculus formula with the situation argument of its fluents suppressed. δ, δ_1, and δ_2 stand for complex actions, Σ is a set, $\boldsymbol{x}$ is a set of variables, β is a procedure name, and $\boldsymbol{p}$ denotes the actual parameters to the procedure. These constructs are mostly self-explanatory. Intuitively, the interrupts work as follows. Whenever $\exists \boldsymbol{x}.\phi$ becomes true, then δ is executed with the bindings of $\boldsymbol{x}$ that satisfied ϕ; once δ terminates, the interrupt can trigger again.

Procedures are defined with the following syntax: **proc** $\beta(\boldsymbol{y})\ \delta$ **endProc**, where β is the procedure name, $\boldsymbol{y}$ denotes the formal parameters to the procedure, and δ is the procedure body, a complex action. The semantics of ConGolog programs are defined using the *Do* predicate (see [6] for details). Informally, *Do*(ρ, s, s') holds if situation s' is a legal terminating situation of program ρ starting in situation s.

4 Modeling the Feature Interaction Resolution System

In the feature interaction resolution system, there are two types of agents: those that represent the interests of humans, which we call *personal* agents and the *negotiator*, which coordinates the negotiation of a solution for the personal agents. The personal agents negotiate to create telephone connections. One personal agent is the initiating agent. The initiating agent has an *aim* for the negotiation. Any solution to the negotiation will be a specialization of this aim. The negotiator receives proposals from the personal agents and forwards them to the other personal agents and waits for their responses. If everyone agrees to a proposal (or if there are no more proposals to try) then the negotiator terminates the negotiation successfully (unsuccessfully, resp.). When a personal agent receives a proposal it answers whether it agrees to it. If it does not agree to the proposal then it can make a counterproposal.

We implement proposals as complex actions in our system. For this example, the possible proposals will be any of the primitive actions listed earlier or any of the following complex actions:

$$\textsc{connect}(x,y) \stackrel{\text{def}}{=} \textsc{twoWay}(x,y) \mid \textsc{record}(x,y)$$
$$\textsc{anonymousCall}(x,y) \stackrel{\text{def}}{=} \textsc{connect}(x,y); \textsc{disconnect}(x,y)$$
$$\textsc{identity}(x,y) \stackrel{\text{def}}{=} \textsc{name}(x,y) \mid \textsc{number}(x,y).$$
$$\textsc{identifiedCall}(x,y) \stackrel{\text{def}}{=} \textsc{connect}(x,y); \textsc{identity}(x,y); \textsc{disconnect}(x,y)$$
$$\textsc{call}(x,y) \stackrel{\text{def}}{=} \textsc{anonymousCall}(x,y) \mid \textsc{identifiedCall}(x,y)$$

Note that these complex actions represent the simplest possible sequence of events that can occur for each proposal. Normally, one would expect that, for example, in an anonymous call other actions would occur between the connect and a disconnect action, namely the agents (or, rather, the humans the agents represent) would speak to each other. However, for the purposes of negotiation, these simpler specifications suffice because they include the actions of each proposal that are relevant to the negotiations.

In order to simplify the presentation of the system, we have a fixed initiating agent with a fixed aim and a fixed initial proposal. The initiator may not accept all specializations of its aim, so it initially suggests a specialization of its aim that it accepts, which is its initial proposal. We have a functional fluent, AIM, to represent the initiator's aim in order to allow other agents to be ignorant of the initial aim. The aim remains fixed over time, therefore, only one negotiation will take place. It would not be difficult to generalize the system to handle multiple negotiations.

We now list some definitions that will be used in the remainder of the paper:

$\text{SENDREC}(x, y, a) \overset{\text{def}}{=} \text{SENDEROF}(a) = x \wedge \text{RECIPIENTOF}(s) = y$

The sender in the action a is x and the recipient in a is y. The definitions of $\text{SENDEROF}(a)$ and $\text{RECIPIENTOF}(a)$ are straightforward and we omit them.

Eventually$(\phi, now, then) \overset{\text{def}}{=} \exists s'.now \preceq s' \preceq then \wedge \phi(s')$

Eventually ϕ holds in the path defined by $(now, then)$.

Next$(seq, now, then) \overset{\text{def}}{=} now \preceq do(seq, now) \preceq then$

The sequence of actions seq occurs next in the path defined by $(now, then)$. For this definition, the *do* function is overloaded to handle sequences of actions, but we leave out the new definition here.

Previously$(\alpha, s) \overset{\text{def}}{=} \exists s', s''.s' \preceq s'' \preceq s \wedge Do(\alpha, s', s'')$

The complex action α occurred in the history of s.

$\text{PROPOSAL}(\tau) \overset{\text{def}}{=}$
$\exists x, y(\tau = \text{TWOWAY}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{RECORD}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{TWOWAY}(x, y); \text{IDENTITY}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{RECORD}(x, y); \text{IDENTITY}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{TWOWAY}(x, y); \text{NAME}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{TWOWAY}(x, y); \text{NUMBER}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{RECORD}(x, y); \text{NAME}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{RECORD}(x, y); \text{NUMBER}(x, y); \text{DISCONNECT}(x, y)) \vee$
$\exists x, y(\tau = \text{ANONYMOUSCALL}(x, y)) \vee \exists x, y(\tau = \text{IDENTIFIEDCALL}(x, y)) \vee$
$\exists x, y(\tau = \text{CALL}(x, y))$

τ is a complex action that can be used as a proposal.

$\text{SPECIALIZATION}(\tau, \rho) \overset{\text{def}}{=}$
$(\forall s, s'.Do(\tau, s, s') \supset Do(\rho, s, s')) \wedge \text{PROPOSAL}(\tau) \wedge \text{PROPOSAL}(\rho) \wedge \tau \neq \rho$

The proposal τ is a (strict) specialization of the proposal ρ.

$\text{COUSIN}(\tau, \rho) \overset{\text{def}}{=} \exists \delta(\text{SPECIALIZATION}(\tau, \delta) \wedge \text{SPECIALIZATION}(\rho, \delta))$

The proposal τ is a cousin of the proposal ρ, if both τ and ρ are specializations of another proposal δ (i.e., δ is a common ancestor of τ and ρ in the specialization tree of proposals). When a non-initiating agent does not accept any of the specializations of a proposal (e.g. τ), it will try to find an acceptable proposal by generalizing τ and finding a specialization of the generalization that it accepts. In other words, it will suggest an acceptable cousin of τ.

$\textsc{Acceptable}(agt, \tau, s) \overset{\text{def}}{=}$
$\forall seq[\textbf{Know}(agt, \textit{Do}(\tau, now, \textit{do}(seq, now)), s) \supset \neg\textbf{Goal}(agt, \neg\textbf{Next}(seq), s)] \wedge$
$\forall\rho[\textbf{Know}(agt, \textsc{aim} = \rho, s) \supset \textbf{Know}(agt, \textsc{Specialization}(\tau, \rho), s)]$

The proposal τ is acceptable to agt in s. This holds if every sequence of actions that agt thinks is a legal execution of τ is compatible with agt's goals, and if agt knows the initial aim (ρ) of the negotiation (i.e., agt is the initiator) then agt also knows that τ is a specialization of ρ.

$\textsc{NiaAccept}(\tau, s) \overset{\text{def}}{=} \forall agt \in \textsc{agents}(\tau) - \{initiator\}.\textsc{Acceptable}(agt, \tau, s)$

All the non-initiating agents accept the proposal τ. $\textsc{agents}(\tau)$ is a function, whose definition we omit, that returns the set of agents involved in proposal τ. $initiator$ is a free variable in this definition and will be bound by an outer construct.

$\textsc{AllAccept}(\tau, s) \overset{\text{def}}{=} \forall agt \in \textsc{agents}(\tau).\textsc{Acceptable}(agt, \tau, s)$

All the agents accept τ.

$\textsc{PossibleSolution}(\tau, agt, s) \overset{\text{def}}{=}$
$\textbf{Know}(\textit{self}, \textsc{Acceptable}(agt, \tau), s) \wedge \neg\textbf{Know}(\textit{self}, \neg\textsc{AllAccept}(\tau), s)$

This definition is used by the negotiator to select possible solutions to propose to the agents. A proposal τ is a possible solution for agt if the negotiator knows that τ is acceptable to agt, i.e., τ has already been suggested to the negotiator by agt, and the negotiator does not know that τ is not acceptable to another agent.

$\textsc{inform}(informer, agt, \phi) \overset{\text{def}}{=}$
$\textsc{inform}(informer, agt, \textsc{encode}(informer, agt, \phi, now))$

$\textsc{request}(requester, agt, \psi) \overset{\text{def}}{=}$
$\textsc{request}(requester, agt, \textsc{encode}(requester, agt, \psi, now))$

All messages are encrypted. To simplify the notation, we will have formulae as arguments to INFORM and REQUEST actions, but the formulae are replaced by their encodings according to these definitions.

4.1 Agent Behaviors and Example Scenario

In Griffeth and Velthuijsen's example scenario, there are two personal agents, UN and CND, and the negotiator, N. UN is the initiator and its AIM is to initiate a call with CND, i.e., $\textsc{Call}(\text{UN}, \text{CND})$. UN's owner has an unlisted number, so UN has the constraint that it never wants to send its number to another agent. CND's owner always wants to know who is calling, so CND never wants to accept an unidentified connection. Here is the axiom that specifies CND's initial goals:

$$\begin{aligned}
&\textbf{OGoal}_0(\text{CND}, \forall x(\textbf{Always}(\forall a_1, a_2, s_2\{\textit{do}(a_1, now) \prec \textit{then} \wedge \\
&\quad \textit{Do}(\textsc{connect}(x, \text{CND}), now, \textit{do}(a_1, now)) \wedge \textsc{SendRec}(x, \text{CND}, a_2) \wedge \\
&\quad \textit{do}(a_1, now) \preceq \textit{do}(a_2, s_2) \preceq \textit{then} \wedge \\
&\quad [\forall a^*, s^*.\textit{do}(a_1, now) \prec \textit{do}(a^*, s^*) \preceq \textit{do}(a_2, s_2) \supset \neg\textsc{SendRec}(x, \text{CND}, a^*)] \supset \\
&\qquad\qquad \textit{Do}(\textsc{identity}(x, \text{CND}), s_2, \textit{do}(a_2, s_2))\}))).
\end{aligned}$$

In all paths consistent with CND's goals, whenever an agent x initiates a connection with CND, the next action performed by x towards CND identifies x to CND.

If an agent has any other initial goals, they will have to be included in the $\textbf{OGoal}_0$ axiom for that agent. In our example, UN is the initiator agent, so we take it to want to know the result of the negotiation, i.e. whether some proposal is acceptable to all, and if so what that proposal is. Since UN's owner has an unlisted number, UN also wants it to be the case that it never divulges its owner's phone number. We state UN's initial goals with the following axiom:

$$\begin{aligned}&\textbf{OGoal}_0(\text{UN}, [\forall y.\textbf{Always}(\neg\text{N}\textsc{umber}\text{IP}(\text{UN}, y)) \wedge \textbf{Eventually}(\\&\quad\exists\tau\,\textbf{Know}(\text{UN}, \text{A}\textsc{ll}\text{A}\textsc{ccept}(\tau)) \vee \textbf{Know}(\text{UN}, \neg\exists\tau\,\text{A}\textsc{ll}\text{A}\textsc{ccept}(\tau)))]).\end{aligned}$$

We also need to specify what the agents know initially. We want to assert that initially, all agents only know how to encode and decode messages addressed to them. In order to do this, we need functions to represent the actual encoding and decoding of messages. Therefore, we introduce $\textsc{enc}(sender, rec, \phi)$, whose value is the encoding of ϕ, and $\textsc{dec}(sender, rec, c)$, whose value is the decoding of c, i.e., a formula. We need an axiom to ensure that $\textsc{dec}$ is the inverse of $\textsc{enc}$:

$$\textsc{dec}(sender, rec, \textsc{enc}(sender, rec, \phi)) = \phi.$$

Here is the axiom that defines the non-initiating agents' initial knowledge:

$$\begin{aligned}&\forall agt.agt \neq \text{UN} \supset\\&\quad\textbf{OKnow}_0(agt, [(\forall rec, \phi.\textsc{encode}(agt, rec, \phi) = \textsc{enc}(agt, rec, \phi)) \wedge\\&\qquad\qquad(\forall sender, c.\textsc{decode}(sender, agt, c) = \textsc{dec}(sender, agt, c))]).\end{aligned}$$

The initiator also needs to know the value of $\textsc{aim}$, which in this example is $\text{C}\textsc{all}(\text{UN}, \text{CND})$, as stated in the following axiom:

$$\begin{aligned}&\textbf{OKnow}_0(\text{UN}, [(\forall rec, \phi.\textsc{encode}(\text{UN}, rec, \phi) = \textsc{enc}(\text{UN}, rec, \phi)) \wedge\\&\qquad(\forall sender, c.\textsc{decode}(sender, \text{UN}, c) = \textsc{dec}(sender, \text{UN}, c)) \wedge\\&\qquad\textsc{aim} = \text{C}\textsc{all}(\text{UN}, \text{CND})]).\end{aligned}$$

The procedures that specify the behavior of the agents are shown in Figures 1 and 2. A system with an initiator whose initial proposal is $initProp$, one other negotiating agent agt, and a negotiator $negot$ can be specified with the following complex action, which we will call $\textsc{fir}(initiator, agt, negot, initProp)$:

$$\begin{aligned}&\textsc{initiate}(initiator, negot, initProp) \;\|\; \textsc{personal}(agt, negot) \;\|\\&\textsc{negotiator}(negot)\end{aligned}$$

For example, we can specify the scenario given in Griffeth and Velthuijsen, where UN initiates a negotiation with CND with N as the negotiator, and with $\textsc{two}\text{W}\textsc{ay}(\text{UN}, \text{CND}); \textsc{disconnect}(\text{UN}, \text{CND})$ as UN's initial proposal, as follows:

$$\textsc{fir}(\text{UN}, \text{CND}, \text{N}, \textsc{two}\text{W}\textsc{ay}(\text{UN}, \text{CND}); \textsc{disconnect}(\text{UN}, \text{CND})).$$

We will now go through a trace of this scenario, in order to explain how the agent procedures work. UN is running the $\textsc{initiate}$ procedure. UN's first action is to request the negotiator to ensure that UN eventually knows a proposal that is acceptable to all:

$$\textsc{request}(\text{UN}, \text{N}, \textbf{Eventually}(\exists\tau(\textbf{Know}(\text{UN}, \text{A}\textsc{ll}\text{A}\textsc{ccept}(\tau))))).$$

Then UN informs the negotiator that it accepts its own initial proposal:

```
**proc** $\textsc{initiate}(self, negotiator, initProp)$
  $\textsc{request}(self, negotiator, \textbf{Eventually}(\exists\tau.\textbf{Know}(self, \textsc{AllAccept}(\tau))))$;
  $\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, initProp))$;
  $[\langle\textbf{Know}(self, \neg\exists\tau.\textsc{AllAccept}(\tau)) \wedge$
    $\neg\textbf{Previously}(\textsc{inform}(self, negotiator, \neg\exists\tau.\textsc{AllAccept}(\tau))) \rightarrow$
        $\textsc{inform}(self, negotiator, \neg\exists\tau.\textsc{AllAccept}(\tau))\rangle$
   $\rangle\!\rangle$
   $\textsc{personal}(self, negotiator)]$
**endProc**

**proc** $\textsc{personal}(self, negotiator)$
  $\langle\tau : \textbf{Goal}(self, \textbf{Eventually}(\textbf{KWhether}(negotiator, \textsc{Acceptable}(self, \tau)))) \wedge$
      $\textbf{Know}(self, \neg\textbf{Previously}(\textsc{informWhether}(self, negotiator,$
                                              $\textsc{Acceptable}(self, \tau)))) \rightarrow$
          **if** $\textbf{Know}(self, \textsc{Acceptable}(self, \tau))$ **then**
            $\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, \tau))$
          **else**
            $\textsc{inform}(self, negotiator, \neg\textsc{Acceptable}(self, \tau))$ ;
            $\textsc{counterPropose}(self, \tau, negotiator)$
          **endIf**$\rangle$
**endProc**

**proc** $\textsc{counterPropose}(self, \tau, negotiator)$
  **if** $\exists\tau'.\textbf{Know}(self, \textsc{Specialization}(\tau', \tau) \wedge \textsc{Acceptable}(self, \tau') \wedge$
        $\neg\textsc{Previously}(\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, \tau'))))$ **then**
    $\pi\tau'.\textbf{Know}(self, \textsc{Specialization}(\tau', \tau) \wedge \textsc{Acceptable}(self, \tau') \wedge$
        $\neg\textsc{Previously}(\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, \tau'))))?$;
      $\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, \tau'))$
  **elsif** $\exists\tau'.\textbf{Know}(self, \textsc{Cousin}(\tau', \tau) \wedge \textsc{Acceptable}(self, \tau') \wedge$
        $\neg\textbf{Previously}(\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, \tau'))))$ **then**
    $\pi\tau'.\textbf{Know}(self, \textsc{Cousin}(\tau', \tau) \wedge \textsc{Acceptable}(self, \tau') \wedge$
        $\neg\textbf{Previously}(\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, \tau'))))?$;
      $\textsc{inform}(self, negotiator, \textsc{Acceptable}(self, \tau'))$
  **endIf**
**endProc**
```

Fig. 1. Procedures run by the initiator and other personal agents.

```
proc NEGOTIATOR(self)
  ⟨initiator : Goal(self, Eventually(∃τ.Know(initiator, ALLACCEPT(τ)))) →
    while (¬∃τ.Know(self, ALLACCEPT(τ)) ∧
            ¬Know(self, ¬∃τ.ALLACCEPT(τ))) do
      if ∃τ, agent.POSSIBLESOLUTION(τ, agent) then
        πτ, agent.POSSIBLESOLUTION(τ, agent)?;
          if agent ≠ initiator ∧
            ¬KWhether(self, ACCEPTABLE(initiator, τ)) then
            REQUEST(self, initiator, Eventually(
                        KWhether(self, ACCEPTABLE(initiator, τ))));
            KWhether(self, ACCEPTABLE(initiator, τ))?;
            if Know(self, ¬ACCEPT(initiator, τ)) then
              INFORMALL(self, AGENTS(τ), ¬ALLACCEPT(τ))
            endIf
          else
            for agt ∈ AGENTS(τ) − {initiator, agent} do
              REQUEST(self, agt, Eventually(
                        KWhether(self, ACCEPTABLE(agt, τ))))
            endFor;
            KWhether(self, NIAACCEPT(τ))?;
            if Know(self, NIAACCEPT(τ)) then
              TERMINATESUCCESSFULLY(self, τ)
            else
              INFORMALL(self, AGENTS(τ), ¬ALLACCEPT(τ))
            endIf
          endIf
      endIf
    endWhile;
    if ¬∃τ.Know(self, ALLACCEPT(τ)) then
      TERMINATEUNSUCCESSFULLY(self)
    endIf⟩
endProc

proc INFORMALL(self, agts, φ)
  for agt ∈ agts do INFORM(self, agt, φ) endFor
endProc

proc TERMINATESUCCESSFULLY(self, τ)
  INFORMALL(self, AGENTS(τ), ALLACCEPT(τ));
  INFORMALL(self, ALLAGENTS − AGENTS(τ), FINISHEDNEGOTIATION)
endProc

proc TERMINATEUNSUCCESSFULLY(self)
  INFORMALL(self, ALLAGENTS, ¬∃τ.ALLACCEPT(τ))
endProc
```

Fig. 2. Procedures run by the negotiator agent.

$$\begin{aligned}\textsc{inform}(\mathrm{UN}, \mathrm{N}, \textsc{Acceptable}(\mathrm{UN}, \textsc{twoWay}(\mathrm{UN}, \mathrm{CND});&\\ \textsc{disconnect}(\mathrm{UN}, \mathrm{CND}))).&\end{aligned}$$

In order to satisfy UN's request, the negotiator—whose behavior is described by the NEGOTIATOR procedure—will start a negotiation process between UN and CND and then inform them of the results of this process. CND is running the PERSONAL procedure, in which it waits for and responds to proposals from the negotiator. After executing the two actions above, UN also runs the PERSONAL procedure, but at a higher priority it executes an interrupt. In the interrupt, if UN comes to know that there are no untried proposals that it accepts, then it informs the negotiator of this and the negotiation terminates unsuccessfully.

UN's request described above causes the negotiator to have the appropriate goal and its interrupt fires. The first action in the interrupt is a while-loop to try possible solutions. Since the negotiator has not yet found a proposal that everyone accepts, and it does not know that there are no acceptable proposals, and UN has suggested a proposal to the negotiator, and the negotiator does not know that anyone rejects this proposal, the negotiator enters its while-loop with UN's proposal as a possible solution. Since UN is the initiator, the negotiator will ask the non-initiating agents whether they accept UN's proposal. In this case, the only non-initiating agent is CND:

$$\begin{aligned}&\textsc{request}(\mathrm{N}, \mathrm{CND}, \mathbf{Eventually}(\mathbf{KWhether}(\mathrm{N},\\ &\quad \textsc{Acceptable}(\mathrm{CND}, \textsc{twoWay}(\mathrm{UN}, \mathrm{CND}); \textsc{disconnect}(\mathrm{UN}, \mathrm{CND}))))).\end{aligned}$$

Then, the negotiator waits to find out if CND accepts the proposal. The negotiator's request causes CND's interrupt to fire. The proposal is not acceptable to CND because $\textsc{twoWay}(\mathrm{UN}, \mathrm{CND})$ is an implementation of $\textsc{connect}(\mathrm{UN}, \mathrm{CND})$ and the next action whose agent is UN and whose recipient is CND (which happens to be the next action in the proposal, i.e., $\textsc{disconnect}(\mathrm{UN}, \mathrm{CND})$) is not a specialization of $\textsc{identity}(\mathrm{UN}, \mathrm{CND})$. Therefore, CND informs the negotiator that UN's proposal is not acceptable:

$$\begin{aligned}\textsc{inform}(\mathrm{CND}, \mathrm{N}, \neg\textsc{Acceptable}(\mathrm{CND}, \textsc{twoWay}(\mathrm{UN}, \mathrm{CND});&\\ \textsc{disconnect}(\mathrm{UN}, \mathrm{CND}))).&\end{aligned}$$

Next, CND enters the COUNTERPROPOSE procedure. There are no specializations of the proposal, so CND seeks an acceptable cousin. For example, suppose CND chooses $\textsc{identifiedCall}(\mathrm{UN}, \mathrm{CND})$. CND then suggests its choice as a proposal to the negotiator:

$$\textsc{inform}(\mathrm{CND}, \mathrm{N}, \textsc{Acceptable}(\mathrm{CND}, \textsc{identifiedCall}(\mathrm{UN}, \mathrm{CND}))).$$

CND exits COUNTERPROPOSE and then exits the body of the interrupt and waits for further proposals from the negotiator.

The negotiator now has its answer from CND. CND did not accept UN's proposal, so it informs UN and CND that not everyone accepted the proposal and goes back to the top of the while-loop. Now, CND's proposal is a possible solution. Since CND is not the initiator and the negotiator does not know whether UN accepts CND's proposal, the negotiator asks UN whether it accepts the proposal and awaits UN's answer:

$$\text{REQUEST}(\text{N}, \text{UN}, \textbf{Eventually}(\textbf{KWhether}(\text{N}, \\ \text{ACCEPTABLE}(\text{UN}, \text{IDENTIFIEDCALL}(\text{UN}, \text{CND}))))).$$

This request causes the interrupt in UN's instantiation of the PERSONAL procedure to fire. CND's proposal is not acceptable to UN because $\text{TWOWAY}(\text{UN}, \text{CND}); \text{NUMBER}(\text{UN}, \text{CND}); \text{DISCONNECT}(\text{UN}, \text{CND})$ is a specialization of the proposal that clashes with UN's goals. So, UN indicates to the negotiator that it does not accept CND's proposal:

$$\text{INFORM}(\text{UN}, \text{N}, \neg\text{ACCEPTABLE}(\text{UN}, \text{IDENTIFIEDCALL}(\text{UN}, \text{CND}))).$$

$\text{TWOWAY}(\text{UN}, \text{CND}); \text{NAME}(\text{UN}, \text{CND}); \text{DISCONNECT}(\text{UN}, \text{CND})$ is a specialization of the proposal that UN accepts, so UN counterproposes it to the negotiator:

$$\text{INFORM}(\text{UN}, \text{N}, \text{ACCEPTABLE}(\text{UN}, \text{TWOWAY}(\text{UN}, \text{CND}); \text{NAME}(\text{UN}, \text{CND}); \\ \text{DISCONNECT}(\text{UN}, \text{CND}))).$$

The execution then continues in a similar way, with the negotiator suggesting UN's latest proposal to CND, who responds by informing the negotiator that it accepts the proposal. The negotiator then knows that all the agents accepted this last proposal, so it enters the TERMINATESUCCESSFULLY procedure where it informs UN and CND of the successful proposal. At this point, no interrupts fire for any agent, so the program terminates.

4.2 Verification

Griffeth and Velthuijsen informally state and show that their system has the property that if there is a proposal to which all the agents involved agree, it will eventually be found. Since our system is defined formally, we can also formally state and verify its properties. Here is a formal statement of the property suggested by Griffeth and Velthuijsen:

$$\begin{aligned}&\mathit{Do}(\text{FIR}(initiator, agt, negot, initProp), S_0, s) \supset \\ &\quad (\exists\tau.\text{ALLACCEPT}(\tau, S_0)) \supset \\ &\qquad \exists\tau.\forall agt \in \text{AGENTS}(\tau).\textbf{Know}(agt, \text{ALLACCEPT}(\tau), s)).\end{aligned}$$

That is, for any legal execution of $\text{FIR}(initiator, agt, negot, initProp)$ that ends in s, if there is a proposal that is acceptable to all agents, then in s, there will be a proposal that is known by all agents concerned to be acceptable to all.

Conversely, we might want to show that if there is no proposal that is acceptable to all, then this fact becomes known to all:

$$\begin{aligned}&\mathit{Do}(\text{FIR}(initiator, agt, negot, initProp), S_0, s) \supset \\ &\quad (\neg\exists\tau.\text{ALLACCEPT}(\tau, S_0)) \supset \\ &\qquad \forall agt \in \text{ALLAGENTS}.\textbf{Know}(agt, \neg\exists\tau.\text{ALLACCEPT}(\tau), s).\end{aligned}$$

We [15] are currently developing an environment to facilitate the verification of properties of CASL specifications using the PVS verification system [12]. The environment has been used to prove properties of simpler multiagent systems. We plan to use it to verify properties of this example as well.

5 Conclusions and Future Work

We feel that using a formal specification language, such as CASL, to model multiagent systems such as this feature interaction resolution system is advantageous for several reasons. Firstly, since the language has a formal semantics, it is possible to formally state and verify properties of the system. The use of a theory of action with complex actions allows us great flexibility in defining agents' preferences. Also, modeling agents' preferences with mental state operators (i.e., knowledge and goals) allows us to abstract over the representation of these preferences. Moreover, it also allows us to model the communication between agents as speech acts (i.e., informs and requests), which abstracts over the messaging mechanism.

There has been a lot of work in the past on formal specification languages for software engineering [1, 18]. But these formalisms did not include any notions of agents and mental attitudes. Only recently have such notions started to be incorporated into requirements engineering frameworks such as KAOS [4], Albert-II [2], and i^* [20]. But the general view has been that goals and other mentalistic notions must be operationalized away by the time requirements are produced.

Within the agents community, there has been some recent work on agent-oriented software design methodologies [3, 8, 19]. In [5], a formal specification language for multiagent systems based on temporal epistemic logic is described, and techniques for specifying and verifying such systems in a compositional manner are proposed. While compositionality is clearly an important issue for verification, the specification language in [5] is less expressive than CASL; there is no goal modality and the specification of complex behaviors appears to be more difficult. In future work, we hope to address the connection between our specification language and methodologies for designing multiagent systems.

We would also like to verify the properties discussed in Sec. 4.2 as well as other properties of the system using the verification environment that we are developing. As mentioned in Sec. 2.2, we want to develop a more sophisticated method for handling conflicting requests. We would like to expand the example presented here and our model of communicative interaction to make them more realistic.

References

1. D. Bjorner and C. B. Jones. *The Vienna Development Method: The Metalanguage*, volume 61 of *LNCS*. Springer-Verlag, 1978.
2. Ph. Du Bois. *The Albert II Language – On the design and the Use of a Formal Specification language for Requirements Analysis*. PhD thesis, Department of Computer Science, University of Namur, 1995.
3. F. Brazier, B. Dunin-Keplicz, N. R. Jennings, and Jan Treur. Formal specifications of multi-agents systems: A real-world case study. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 25–32, San Francisco, CA, June 1995. Springer-Verlag.
4. A. Dardenne, S. Fickas, and A. van Lamsweerde. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

5. Joeri Engelfriet, Catholijn M. Jonker, and Jan Treur. Compositional verification of multi-agent systems in temporal multi-epistemic logic. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V: Proceedings of the Fifth International Workshop on Agent Theories, Architectures and languages (ATAL'98)*, volume 1555 of *LNAI*, pages 177–194. Springer-Verlag, 1999.
6. Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. To appear in *Artificial Intelligence*.
7. Nancy D. Griffeth and Hugo Velthuijsen. Win/win negotiation among autonomous agents. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 187–202, Hidden Valley, PA, May 1993.
8. D. Kinny, M. Georgeff, and A. S. Rao. A methodology and modelling technique for systems of BDI agents. In W. Van der Velde and J. W. Perram, editors, *Agents Breaking Away*, pages 56–71. LNAI 1038, Springer-Verlag, 1996.
9. Gerhard Lakemeyer and Hector J. Levesque. AOL: a logic of acting, sensing, knowing, and only knowing. In *Proceedings of Knowledge Representation and Reasoning (KR-98)*, pages 316–327, 1998.
10. Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.
11. John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 4*. Edinburgh University Press, 1969.
12. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.
13. Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.
14. Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, July 1993. AAAI Press/The MIT Press.
15. Steven Shapiro. PhD thesis. In preparation.
16. Steven Shapiro, Yves Lespérance, and Hector J. Levesque. Specifying communicative multi-agent systems. In Wayne Wobcke, Maurice Pagnucco, and Chengqi Zhang, editors, *Agents and Multi-Agent Systems — Formalisms, Methodologies, and Applications*, volume 1441 of *LNAI*, pages 1–14. Springer-Verlag, Berlin, 1998.
17. Steven Shapiro, Maurice Pagnucco, Yves Lespérance, and Hector J. Levesque. Iterated belief change in the situation calculus. In A. G. Cohn, F. Giunchiglia, and B.Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000)*, pages 527–538, San Francisco, CA, 2000. Morgan Kaufmann Publishers.
18. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1989.
19. M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In O. Etzioni, J. P. Müller, and J. Bradshaw, editors, *Agents '99: Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, May 1999.
20. Eric S. K. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, Dept. of Computer Science, University of Toronto, 1995.