A Computational Semantics for Communicating Rational Agents Based on Mental Models

Koen V. Hindriks and M. Birna van Riemsdijk

EEMCS, Delft University of Technology, Delft, The Netherlands
{k.v.hindriks,m.b.vanriemsdijk}@tudelft.nl

Abstract. Communication is key in a multi-agent system for agents to exchange information and coordinate their activities. For agents that derive their choice of action from their beliefs and goals it is natural to facilitate communication about both these attitudes in an agent programming language. The traditional approach based on speech act theory, however, does not provide the right tools to do so because of its emphasis on mental conditions on the speaker. Here, we introduce an alternative semantics based on the idea that a received message can be used to *(re)construct a mental model* of the sender. As coordination is particularly important, we introduce the concept of a *conversation* to synchronize actions and communication in a multi-agent system. Conversations are resources at the multi-agent level with restricted access, which provide a natural counterpart in multi-agent systems for classic constructs from distributed programming such as semaphores.

1 Introduction

Communication is key in a multi-agent system for agents to exchange information and coordinate their activities, and therefore needs to be addressed in agent programming languages. Two main lines of research can be identified when it comes to agent communication languages: research based on speech act theory [1, 2] and research based on a social semantics for agent communication (social commitments) [3, 4].

Speech act theory is a philosophical theory that is based on the idea that uttering a sentence is an act which can be used to change the world like any other act. The focus of speech act theory has been on specifying the conditions that identify the particular act that is performed, thereby focussing most of the theory on the sender instead of the receiver. Two of the most well-known agent communication languages that are based on speech act theory are KQML [5] and the FIPA Agent Communication Language (ACL) [6]. Both languages specify the semantics of messages by means of their pre-condition, expressing conditions on the mental states of the sender and receiver of the message that should hold if the message is sent, and their effect, expressing the effect of the message on the mental state of the sending and/or receiving agent.[1] Both languages use *performative labels* to specify message types. For example, the precondition for the

[1] FIPA does not specify that there should be any effect on the sender [6].

(simplified) FIPA *inform*(r, ϕ) message, where *inform* denotes the performative label and ϕ the content of the message, includes that the sending agent believes ϕ and the effect is that the receiver r believes ϕ.

These ACLs have been extensively criticized [7–9]. Here, we mention in particular their *complexity* due to the relatively high number of performative labels and subtle semantical differences between them, and the lack of *verifiability*. The latter means that, for example, an agent receiving a message that is supposed to be an informative speech act with preconditions that require truth of the content, cannot verify these conditions. An important alternative approach that addresses these and other issues, is the semantics for agent communication based on social commitments [3, 4]. This approach does not define communication by referring to the mental states of the involved agents, but focuses on the social consequences of communication. The basic idea here is that a receiver can always confront the sender again with a previous message, i.e., by saying something like: "You told/asked/requested me so". The processing of the message by the receiver, however, has moved to the background here.

We can thus see that communication based on speech act theory is problematic due to its complexity and verifiability. Moreover, the approach based on social commitments does not say anything about how communication affects the involved agents. This issue *does* need to be addressed when developing techniques for communication in agent programming languages, which is what we are interested in in this paper. The contribution of this paper is the introduction of an alternative semantics based on the idea that a received message can be used to *(re)construct a mental model* of the sender. Our semantics does not specify any preconditions on the mental states of the agents for sending a message. Moreover, the effect on the receiver is only that it updates its mental model of the sender, i.e., the mental state of the receiver itself is not directly updated. A second contribution is the introduction of the concept of a *conversation* to facilitate the synchronization of actions and communication in a multi-agent system, which is particularly important to organize agent coordination. Our proposal is made concrete in the agent programming language GOAL [10].

2 Communication in Agent Programming Languages

Communication within other agent programming languages than GOAL has taken a quite pragmatic turn to address the issue discussed above. For example, in 2APL the semantics of communication is reduced to a simple "mailbox" semantics: communicating a message only means that the message is added to a mailbox and the programmer then needs to write rules to handle received messages [11]. This reduces the meaning of communication to the bare minimum. The developers of Jason have chosen a small set of primitives, based on KQML, for which simple default semantics have been defined [8]. For example, the *tell* message inserts the content of the message into the receiver's belief base, with a tag to identify the source of this information, and the *achieve* message inserts the content of the message as a goal in the event base. A function is introduced to determine whether a message is "socially acceptable", and only socially acceptable messages are processed.

Although we take a definite engineering stance in this paper regarding the design of a semantics for communication, our approach is intended to satisfy a number of basic criteria. First, the communication primitives should have a well-defined *semantics*. Preferably, this semantics provides a basis for verifying multi-agent systems as well, where agents use these primitives to communicate. Second, *the distinction between beliefs and goals* needs to be taken into account when defining a communication semantics. As GOAL agents derive their choice of action from their beliefs and goals, it is important to be able to communicate about these different reasons for acting as well as to be able to distinguish between beliefs and goals communicated by other agents. Third, the communication primitives introduced should be *useful for programming* GOAL *agents*. This is a pragmatic criterion that requires minimization of complexity, in particular by limiting the number of communication primitives and endowing these with easy to grasp semantics. Fourth, and finally, *various speech acts should be definable* using the communication primitives introduced. Ideally, it would be possible to define or "program" various well-known speech acts such as promises, requests, etc. in terms of the primitives introduced. It should be kept in mind, however, that characterizing a particular communication event of a multi-agent system may be possible only from a designer or observer's point of view, e.g. by using a *logic* to *reason about* the communication primitives. Initial work to meet this last criterium, which sets our approach apart from others, is reported in [12].

3 Redesigning Agent Communication

Our proposal for redesigning agent communication in the context of the programming language GOAL focuses on the effects of communication on the receiver, a perspective also taken in [13, 8] but which may be contrasted with social commitment semantics. Doing so is not straightforward and requires that some reasonable decisions are made with respect to defining a communication semantics. We illustrate this using an example (see also [14]). Consider the utterance "The house is white". Its effect on the receiver may be one or more of the following, ranging from a very strong to a very weak effect:

1. The receiver comes to believe that the house is white.
2. The receiver comes to believe that the sender believes that the house is white.
3. The receiver comes to believe that the sender had the intention to make the receiver believe that the house is white.
4. The utterance has no effect on the receiver, i.e. its mental state is not changed as a result of the utterance.

In choosing which of these effects should form the basis for our semantics, we take a pragmatic engineering stance, and in addition want to avoid making too strong assumptions. In particular, we consider effect 1 to be too strong in general, as it makes the assumption that the sender always convinces the receiver.

Effect 4 is too weak, and not very useful for programming communication among agents since it would have no effect. Effect 3 would be rather indirect, as it is no longer very clear what *use* the communication has, other than to conclude such indirect statements about the sender's mind. In practice, using this type of semantics would be rather similar to effect 4, doing nothing with a message received from another agent, as making good use of information about the intentions of another agent would require rather involved reasoning patterns. Here, we choose our semantics according to effect 2, which means that the receiver makes the assumption that the speaker believes what it says. Obviously, this is not always a safe assumption to make, as the sender may be lying. However, it is also not overly presumptuous, as the receiver just takes the utterance of the sender at face value.

In contrast with effect 1, effect 2 does not affect the mental state of the receiver directly, but only the *mental model* that the receiver has of the sender: as a result of the utterance, the receiver comes to believe something about the sender, but the utterance does not directly influence the beliefs of the agent about the environment. This is one of the main ideas of our approach: the direct effect of communication is that the receiver updates its mental model of the sender. Additional reasoning may then result in the receiver making updates also to its own beliefs and goals. However, this is not part of our semantics. For example, if the receiver knows by experience that the sender is quite reliable, the receiver may also update its own beliefs accordingly, or use the beliefs of the mental model of the sender to decide on action (as in the example program of Section 5). This is in contrast with Jason, in which reasoning about the acceptability of the message is done upon receipt of the message (using the function to determine the social acceptability), and if the message is acceptable, the mental state of the agent itself is updated.

The second main idea behind our approach has to do with the types of messages that we distinguish. For this, we take inspiration from natural language, in which one uses grammatical structure to differentiate between various types of communication modes. In the communication framework we propose, we distinguish between *three message types*, derived from three *grammatical* distinctions in natural language:

(i) *declaratives*, typically used to make factual statements about the environment (e.g., "The house is white."). Syntactically, a declarative is represented by $\bullet\phi$. Informally, a declarative message with content ϕ may be paraphrased as: "It is the case that ϕ." Semantically, the idea is that the receiver r takes this at face value, and r concludes that sender s believes ϕ.

(ii) *interrogatives*, typically used to pose questions about a state of affairs (e.g., "Is the house white?"). Syntactically, an interrogative is represented by $\mathbf{?}\phi$. Informally, an interrogative with content ϕ may be paraphrased as: "Is it the case that ϕ?". Taking this at face value, r concludes that s does not know whether ϕ.

(iii) *imperatives*, typically used to express a desirable state of affairs (e.g., "See to it that the house is white!"). Syntactically, an imperative is represented by $\mathbf{!}\phi$. Informally, an imperative may be paraphrased as: "Someone, see to it that ϕ." Taking this at face value, r concludes that s has ϕ as a goal, and does not believe ϕ.[2]

The semantics of interrogatives and imperatives have in common with that of declaratives that they do not prescribe what the receiver should do in terms of updating its own beliefs and goals. The semantics of interrogatives does not define that the receiver should, e.g., adopt the goal to tell the sender about ϕ. Similarly, the semantics of imperatives does not define that the receiver should update its own goals with ϕ. Moreover, the semantics of imperatives does not even define whether the uttered imperative should be interpreted as a request, or simply as information about the goals of the sender. Adding these kinds of interpretations and additional reasoning on whether to update the receiver's own beliefs and goals is *left to the agent programmer*. For example, in certain applications imperatives might always be interpreted as requests, while in others one might want to make a distinction between these and imperatives that express the goals of the sender. However, we argue that in all of these cases, it makes sense to update the mental model that the receiver has of the sender as informally described above.

4 A Communication Semantics Based on Mental Models

In this section, we make the informal semantics discussed above precise in the context of GOAL.

4.1 Mental Models and Mental States

Mental models play an essential role in this semantics and are introduced first. GOAL agents maintain mental models that consists of *declarative* beliefs and goals. An agent's beliefs represent its environment whereas the goals represent a state of the environment the agent wants. Beliefs and goals are specified using some knowledge representation technology. In the specification of the operational semantics we use a propositional logic $\mathcal{L}_0$ built from a set of propositional atoms *Atom* and the usual boolean connectives. We use $\models$ to denote the usual consequence relation associated with $\mathcal{L}_0$, and assume a special symbol $\bot \in \mathcal{L}_0$ which denotes the false proposition. In addition, the presence of an operator $\oplus$ for adding ϕ to a belief base and an operator $\ominus$ for removing ϕ from a belief base are assumed to be available.[3] A mental model associated with a GOAL agent needs to satisfy a number of *rationality constraints*.

[2] The latter part, that s does not believe ϕ is derived from a rationality constraint. An agent should not have a goal to achieve something if it believes it has already been achieved.

[3] We assume that $\Sigma \oplus \phi \models \phi$ whenever ϕ is consistent, and that otherwise nothing changes, and that $\Sigma \ominus \phi \not\models \phi$ whenever ϕ is not a tautology, and that otherwise nothing changes. Additional properties such as minimal change, etc. are usually associated with these operators (see e.g. [15]) but not relevant in this context.

Definition 1. *(Mental Model)*
A mental model is a pair $\langle \Sigma, \Gamma \rangle$ *with* $\Sigma, \Gamma \subseteq \mathcal{L}_0$ *such that:*

- *The beliefs are consistent:* $\Sigma \not\models \bot$
- *Individual goals are consistent:* $\forall \gamma \in \Gamma : \gamma \not\models \bot$
- *Goals are not yet (believed to be) achieved:* $\forall \gamma \in \Gamma : \Sigma \not\models \gamma$

In a multi-agent system it is useful for an agent to maintain mental models of other agents. This allows an agent to keep track of the perspectives of other agents on the environment and the goals they have adopted to change it. A mental model maintained by an agent i about another agent j represents what i thinks that j believes and which goals it has. Mental models of other agents can also be used to take the beliefs and goals of these agents into account in its own decision-making. An agent may construct a mental model of another agent from the messages it receives from that agent or from observations of the actions that that agent performs (e.g., using intention recognition techniques). Here we focus on the former option.

We assume a multi-agent system that consists of a fixed number of agents. To simplify the presentation further, we use $\{1, \ldots, n\}$ as names for these agents. A *mental state* of an agent is then defined as a mapping from all agent names to mental models.

Definition 2. *(Mental State)*
A mental state m *is a total mapping from agent names to mental models, i.e.* $m(i) = \langle \Sigma_i, \Gamma_i \rangle$ *for* $i \in \{1, \ldots, n\}$.

For an agent i, $m(i)$ are its own beliefs and goals, which was called the agent's mental state in [10].

A Goal agent is able to inspect its mental state by means of *mental state conditions*. The mental state conditions of Goal consist of atoms of the form $\mathbf{bel}(i, \phi)$ and $\mathbf{goal}(i, \phi)$ and Boolean combinations of such atoms. $\mathbf{bel}(i, \phi)$ where i refers to the agent itself means that the agent itself believes ϕ, whereas $\mathbf{bel}(i, \phi)$ where i refers to another agent means that the agent believes that agent i believes ϕ. Similarly, $\mathbf{goal}(i, \phi)$ is used to check whether agent i has a goal ϕ.[4]

Definition 3. *(Syntax of Mental State Conditions)*
A mental state condition*, denoted by* ψ*, is defined by the following rules:*

$$\begin{aligned} i &::= \textit{any element from } \{1, \ldots, n\} \mid \textbf{\textit{me}} \mid \textbf{\textit{allother}} \\ \phi &::= \textit{any element from } \mathcal{L}_0 \\ \psi &::= \mathbf{bel}(i, \phi) \mid \mathbf{goal}(i, \phi) \mid \psi \wedge \psi \mid \neg \psi \end{aligned}$$

The meaning of a mental state condition is defined by means of the mental state of an agent. An atom $\mathbf{bel}(i, \phi)$ is true whenever ϕ follows from the belief

[4] In a multi-agent setting it is useful to introduce additional labels instead of agent names i, e.g. **me** to refer to the agent itself and **allother** to refer to all other agents, but we will not discuss these here in any detail.

base of the mental model for agent i. An atom $\mathbf{goal}(i, \phi)$ is true whenever ϕ follows from *one* of the goals of the mental model for agent i. This is in line with the usual semantics for goals in GOAL, which allows the goal base to be inconsistent (see [10] for details). Note that we overload $\models$.

Definition 4. *(Semantics of Mental State Conditions)*
Let m be a mental state and $m(i) = \langle \Sigma_i, \Gamma_i \rangle$. Then the semantics of mental state conditions is defined by:

$$\begin{array}{lll} m \models \mathbf{bel}(i, \phi) & \textit{iff} & \Sigma_i \models \phi \\ m \models \mathbf{goal}(i, \phi) & \textit{iff} & \exists \gamma \in \Gamma_i \textit{ such that } \gamma \models \phi \\ m \models \neg\psi & \textit{iff} & m \not\models \psi \\ m \models \psi \wedge \psi' & \textit{iff} & m \models \psi \textit{ and } m \models \psi' \end{array}$$

4.2 Actions

GOAL has a number of built-in actions and also allows programmers to introduce user-specified actions by means of STRIPS-style action specifications. The program discussed in Section 5 provides examples of various user-specified actions. In the definition of the semantics we will abstract from action specifications specified by programmers and assume that a fixed set of actions *Act* and a (partial) transition function T is given. T specifies how actions from *Act*, performed by agent i, update i's mental state, i.e., $T(i, a, m) = m'$ for i an agent name, $a \in \mathit{Act}$ and m, m' mental states. All actions except for communicative actions are assumed to only affect the mental state of the agent performing the action.

The built-in actions available in GOAL (adapted to distinguish between mental models) that we need here include $\mathbf{ins}(i, \phi)$, $\mathbf{del}(i, \phi)$, $\mathbf{adopt}(i, \phi)$, $\mathbf{drop}(i, \phi)$ and communicative actions of the form $\mathbf{send}(i, \mathit{msg})$ where i is an agent name and msg is a message of the form $\bullet\phi$, $\mathbf{?}\phi$ or $\mathbf{!}\phi$. The semantics of actions from *Act* and built-in actions performed by agent i is formally captured by a mental state transformer function M defined as follows:

$$\begin{array}{lll} M(i, a, m) & = & \begin{cases} T(i, a, m) & \text{if } a \in \mathit{Act} \text{ and } T(i, a, m) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases} \\ M(i, \mathbf{ins}(j, \phi), m) & = & m \oplus_j \phi \\ M(i, \mathbf{del}(j, \phi), m) & = & m \ominus_j \phi \\ M(i, \mathbf{adopt}(j, \phi), m) & = & \begin{cases} m \cup_j \phi & \text{if } \phi \text{ is consistent and } m \not\models \mathbf{bel}(i, \phi) \\ \text{undefined} & \text{otherwise} \end{cases} \\ M(i, \mathbf{drop}(j, \phi), m) & = & m -_j \phi \\ M(i, \mathbf{send}(j, \mathit{msg}), m) & = & m \end{array}$$

where $m \times_j \phi$ means that operator $\times \in \{\oplus, \ominus, \cup, -\}$ is applied to mental model $m(j)$, i.e. $m \times_j \phi(i) = m(j) \times \phi$ and $m \times_j \phi(k) = m(k)$ for $k \neq j$. To define the application of operators to mental models, we use $Th(T)$ to denote the logical theory induced by T, i.e. the set of all logical consequences that can be derived from T. Assuming that $m(i) = \langle \Sigma, \Gamma \rangle$, we then define: $m(i) \oplus \phi = \langle \Sigma \oplus \phi, \Gamma \setminus (Th(\Sigma \oplus \phi)) \rangle$, $m(i) \ominus \phi = \langle \Sigma \ominus \phi, \Gamma \rangle$, $m(i) \cup \phi = \langle \Sigma, \Gamma \cup \{\phi\} \rangle$, and

$m(i) - \phi = \langle \Sigma, \Gamma \setminus \{\gamma \in \Gamma \mid \gamma \models \phi\}\rangle$. Note that sending a message does not have any effect on the sender. There is no need to incorporate any such effects in the semantics of **send** since such effects may be *programmed* by using the other built-in operators.

It is useful to be able to perform multiple actions simultaneously and we introduce the + operator to do so. The idea here is that multiple mental actions may be performed simultaneously, possibly in combination with the execution of a *single* user-specified action (as such actions may have effects on the external environment it is not allowed to combine multiple user-specified actions by the + operator). The meaning of $a + a'$ where a, a' are actions, is defined as follows: if $M(i, a, m)$ and $M(i, a', m)$ are defined and $M(i, a', M(i, a, m)) = M(i, a, M(i, a', m))$ is a mental state, then $M(i, a{+}a', m) = M(i, a', M(i, a, m))$; otherwise, $a + a'$ is undefined.

In order to select actions for execution, an agent uses action rules of the form **if** ψ **then** a, where a is a user-specified action, a built-in action, or a combination using the +-operator. An agent $\mathcal{A}$ is then a triple $\langle i, m, \Pi\rangle$ where i is the agent's name, m is the agent's mental state, and Π is the agent's program (a set of action rules).

4.3 Operational Semantics: Basic Communication

We first introduce a single transition rule for an agent performing an action. Transitions "at the agent level" are labelled with the performed action, since this information is required "at the multi-agent level" in the case of communicative actions.

Definition 5. (Actions)
Let $\mathcal{A} = \langle i, m, \Pi\rangle$ *be an agent, and* **if** ψ **then** $\mathsf{a} \in \Pi$ *be an action rule.*

$$\frac{m \models \psi \quad M(i, \mathsf{a}, m) \textit{ is defined}}{m \xrightarrow{\mathsf{a}} M(i, \mathsf{a}, m)}$$

Using Plotkin-style operational semantics, the semantics at the multi-agent level is provided by the rules below. A configuration of a multi-agent system consists of the agents of the multi-agent system $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ and the environment E, which is used to store messages that have been sent and are waiting for delivery.[5] The environment is used to model *asynchronous* communication, i.e., no handshake is required between sender and receiver of a message. Transitions at the multi-agent level are not labelled. Actions other than the **send** action only change the agent that executes them, as specified below.

Definition 6. (Action Execution)
Let "a" be an action other than send(j, msg).

$$\frac{\mathcal{A}_i \xrightarrow{a} \mathcal{A}_i'}{\mathcal{A}_1, \dots, \mathcal{A}_i, \dots, \mathcal{A}_n, E \longrightarrow \mathcal{A}_1, \dots, \mathcal{A}_i', \dots, \mathcal{A}_n, E}$$

[5] Other aspects of the environment might also be modeled, but that is beyond the scope of this paper.

The following transition rule specifies the semantics of sending messages.

Definition 7. (Send)

$$\frac{\mathcal{A}_i \overset{send(j,msg)}{\longrightarrow} \mathcal{A}_i}{\mathcal{A}_1,\ldots,\mathcal{A}_i,\ldots,\mathcal{A}_n,E \longrightarrow \mathcal{A}_1,\ldots,\mathcal{A}_i,\ldots,\mathcal{A}_n,E \cup \{send(i,j,msg)\}}$$

The premise of the rule indicates that agent $\mathcal{A}_i$ sends a message to agent $\mathcal{A}_j$. To record this, $send(i,j,msg)$ is added to the environment, including both the sender i and the intended receiver j. Also note that a message that is sent more than once has no effect as the environment is modeled as a set here (this is the case until the message is received).[6]

Three rules for receiving a message are introduced below, corresponding to each of the three message types. In each of these rules, the conclusion of the rule indicates that the mental state of the receiving agent is changed. If agent j receives a message from agent i that consists of a declarative sentence, it has the effect that the mental model $m(i)$ of the mental state of the receiver j is modified by updating the belief base of $m(i)$ with ϕ. In addition, any goals in the goal base of $m(i)$ that are implied by the updated belief base are removed from the goal base to ensure that the rationality constraints associated with mental models are satisfied.

Definition 8. (Receive: Declaratives)

$$\frac{send(i,j,\bullet\phi) \in E}{\mathcal{A}_1,\ldots,\langle j,m,\Pi\rangle,\ldots,\mathcal{A}_n,E \longrightarrow \mathcal{A}_1,\ldots,\langle j,m',\Pi\rangle,\ldots,\mathcal{A}_n,E \setminus \{send(i,j,\bullet\phi)\}}$$

where:

- $m'(i) = \langle \Sigma \oplus \phi, \Gamma \setminus Th(\Sigma \oplus \phi)\rangle$ *if* $m(i) = \langle \Sigma, \Gamma\rangle$, *and*
- $m'(k) = m(k)$ *for* $k \neq i$.

The condition $m'(k) = m(k)$ for $k \neq i$ ensures that only the mental model associated with the sender i is changed.

The rule below for interrogatives formalizes that if agent i communicates a message $?\varphi$ of the interrogative type, then the receiver j will assume that i does not know the truth value of ϕ. Accordingly, it removes ϕ using the $\ominus$ operator from the belief base in its mental model of agent i to reflect this.

Definition 9. (Receive: Interrogatives)

$$\frac{send(i,j,?\phi) \in E}{\mathcal{A}_1,\ldots,\langle j,m,\Pi\rangle,\ldots,\mathcal{A}_n,E \longrightarrow \mathcal{A}_1,\ldots,\langle j,m',\Pi\rangle,\ldots,\mathcal{A}_n,E \setminus \{send(i,j,?\phi)\}}$$

where:

- $m'(i) = \langle (\Sigma \ominus \phi)) \ominus \neg\phi, \Gamma\rangle$ *if* $m(i) = \langle \Sigma, \Gamma\rangle$, *and*
- $m'(k) = m(k)$ *for* $k \neq i$.

[6] The implicit quantifier **allother** may be used to define a broadcasting primitive: $\mathbf{broadcast}(msg) \overset{df}{=} \mathbf{send}(\mathbf{allother}, msg)$. In the rule above, in that case, for all $i \neq j$ $send(i,j,msg)$ should be added to E, but we do not provide the details here.

Remark An alternative, more complex semantics would not just conclude that agent i does not know ϕ but also that i wants to know the truth value of ϕ, introducing a complex proposition $K_i\phi$ into the model of the goal base of that agent. This would involve nesting of operators, or including $K_i\phi$ in the goal base.

The rule below for imperatives formalizes that if agent i communicates a message $\boldsymbol{!}\phi$ of the imperative type, then the receiver j will assume that i does not believe that ϕ is the case, and also that ϕ is a goal of i. Accordingly, it removes ϕ using the $\ominus$ operator and adds ϕ to its model of the goal base of agent i.

Definition 10. (Receive: Imperatives)

$$\frac{send(i, j, \boldsymbol{?}\phi) \in E}{\mathcal{A}_1, \ldots, \langle j, m, \Pi \rangle, \ldots, \mathcal{A}_n, E \longrightarrow \mathcal{A}_1, \ldots, \langle j, m', \Pi \rangle, \ldots, \mathcal{A}_n, E \setminus \{send(i, j, \boldsymbol{?}\phi)\}}$$

where:

- $m'(i) = \langle \Sigma \ominus \phi, \Gamma \cup \{\phi\} \rangle$ *if* $\phi \not\models \bot$ *and* $m(i) = \langle \Sigma, \Gamma \rangle$;
  otherwise, $m'(i) = m(i)$.
- $m'(k) = m(k)$ *for* $k \neq i$.

Note that this semantics does not refer to the *actual* mental state of the sender, nor does it define when a sender should send a message or what a receiver should do with the contents of a received message (other than simply record it in its mental model of the sending agent).

4.4 Operational Semantics: Conversations

As is well-known, in concurrent systems one needs mechanisms to ensure that processes cannot access a particular resource simultaneously. A similar need arises in multi-agent systems, but this has received little attention in the agent programming community so far. Emphasis has been put on the fact that agent communication is *asynchronous*. However, in order to ensure that only one agent has access to a particular resource at any time, agents need to be able to coordinate their activities and *synchronize* their actions.[7] Of course, asynchronous communication allows to implement synchronization between agents. We argue, however, that it is useful to have predefined primitives available in an agent programming language that facilitate coordination and synchronization, as is usual in concurrent programming [16]. We introduce a mechanism that fits elegantly into the overall setup of communication primitives introduced above, using the notion of a *conversation*.

The basic idea is that an agent can engage only in a limited number of conversations at the same time. By viewing a conversation as a resource, the

[7] Note that *perfectly symmetrical solutions to problems in concurrent programming are impossible because if every process executes exactly the same program, they can never 'break ties'* [16]. To resolve this, solutions in concurrency theory contain asymmetries in the form of process identifiers or a kernel maintaining a queue.

limit on the number of conversations an agent can participate in simultaneously thus introduces a limit on access to that resource. For our purposes, it will suffice to assume that an agent can participate in at most one conversation at the same time.

More specifically, a parameter representing a unique conversation identifier can be added when sending a message, i.e., $\mathbf{send}(c : j, msg)$ specifies that the message msg should be sent to agent j as part of the ongoing conversation c. We also allow conversations with groups of more than two agents which is facilitated by allowing groups of agent names $\{\ldots\}$ to be inserted into $\mathbf{send}(c : \{\ldots\}, msg)$. A message that is sent as part of an ongoing conversation c is handled similarly to a message that is not part of a specific conversation. Whenever a conversation c has been closed (see below), sent messages that are intended to be part of that conversation are "lost", i.e. nothing happens. To initiate a conversation, the term **new** can be used instead of the conversation identifier. That is, whenever an agent i performs a $\mathbf{send}(\mathbf{new} : g, msg)$ action where g is an agent or a group of agents, agent i initiates a new conversation. Because agents can only engage in a limited number of conversations at the time, it may be that an initiated conversation is *put on hold initially* because one of the agents that should participate already participates in another conversation.

Semantically, to be able to model that a conversation is ongoing, we split the environment into a set A of *active conversations*, a queue Q of *pending conversations*, and a set M of other pending messages. A message to initiate a new conversation is added to the queue *if* at least one agent that should participate is already present in the set A or the queue Q. The check on Q guarantees that a conversation is not started when another conversation requiring the participation of one of the same agents is still on hold in the queue ("no overtaking takes place"). Otherwise, the message is directly added to the set of active conversations.

Whenever a message $send(c : i, g, msg)$ that initiated a conversation is part of the set A, written $c \in A$, we will say that *conversation c is ongoing*, and when such a message is part of the queue Q, written $c \in Q$, we will say that *conversation c is put on hold.* Since the rules for receiving messages remain essentially the same, we only provide the rules for sending a message at the multi-agent level. The following rule specifies the semantics of sending a message that is part of an ongoing conversation.

Definition 11. (Send: Ongoing Conversation)

$$\frac{\mathcal{A}_i \xrightarrow{send(c:j,msg)} \mathcal{A}_i' \quad c \in A}{\mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots, \mathcal{A}_n, \langle A, Q, M\rangle \longrightarrow \mathcal{A}_1, \ldots, \mathcal{A}_i', \ldots, \mathcal{A}_n, \langle A, Q, M'\rangle}$$

where $M' = M \cup \{send(c : i, j, msg)\}$.

The following transition rule specifies the semantics of messages that are used to initiate conversations. We use $+$ (e.g., $Q{+}send(c : i, g, msg)$) to add a message to the tail of a queue. The set of active conversations A and the queue Q store information about participants in conversations, as this may be derived from **send**$(c : i, g, msg)$, where agents i and g are participants. We write $agent(A, Q)$ to denote the set of agents in A and Q.

Definition 12. (Send: Initiating a Conversation)
Let g be a set of agent names, and c a new conversation identifier not yet present in A or Q.

$$\frac{\mathcal{A}_i \xrightarrow{send(\mathbf{new}:g,msg)} \mathcal{A}_i'}{\mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots, \mathcal{A}_n, \langle A, Q, M\rangle \longrightarrow \mathcal{A}_1, \ldots, \mathcal{A}_i', \ldots, \mathcal{A}_n, \langle A', Q', M'\rangle}$$

where if $(\{i\} \cup g) \cap agents(A, Q) = \emptyset$ then $A' = A \cup \{send(c : i, g, msg)\}$, $Q' = Q$ and $M' = \bigcup_{k \in g} send(c : i, k, msg)$, and otherwise $A' = A$, $Q' = Q + send(c : i, g, msg)$, and $M' = M$.

This semantics specifies that we cannot simply allow a conversation between two agents to start when these agents are not part of an ongoing conversation, as this may prevent a conversation between another group of agents involving the same agents from ever taking place. The point is that it should be prevented that "smaller" conversations always "overtake" a conversation between a larger group of agents that is waiting in the queue.

As conversations are a resource shared at the multi-agent level, it must be possible to free this resource again. To this end, we introduce a special action **close**(c) which has the effect of removing an ongoing conversation from the set A and potentially adding conversations on hold from the queue Q to A. This is the only essentially new primitive needed to implement the conversation synchronization mechanism.

We need an additional definition: we say that F is a *maximal fifo-set of messages* derived from a queue Q relative to a set of agent names Agt if F consists of *all* messages $send(c : i, g, msg)$ from Q that satisfy the following constraints: (i) $(\{i\} \cup g) \cap Agt = \emptyset$, and (ii) there is no earlier message $send(c' : i', g', msg')$ in the queue Q such that $(\{i\} \cup g) \cap g' \neq \emptyset$.

Definition 13. (Send: Closing a Conversation)

$$\frac{\mathcal{A}_i \xrightarrow{close(c)} \mathcal{A}_i'}{\mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots, \mathcal{A}_n, \langle A, Q, M\rangle \longrightarrow \mathcal{A}_1, \ldots, \mathcal{A}_i', \ldots, \mathcal{A}_n, \langle A', Q', M\rangle}$$

where, assuming that F is the maximal fifo-set derived from Q relative to $agents(A)$, if $send(c : i, g, msg) \in A$ then $A' = (A \setminus \{send(c : i, g, msg)\}) \cup F$ and $Q' = Q \setminus F$, and otherwise $A' = A$ and $Q' = Q$.

Note that the transition rule for closing a conversation only allows the initiator of a conversation, i.e. agent $\mathcal{A}_i$, to close the conversation again. (Otherwise agents that want to start their own conversation immediately might try to get it going by closing other conversations.) Finally, as it is important that the initiating agent as well as other participating agents are aware that a conversation has started or is ongoing, we assume a special predicate $conversation(c, i)$ is available, where c denotes a unique conversation identifier and i the initiating agent, which can be used in the belief base of an agent to verify whether a conversation is ongoing or not. We do not provide the formal details here due to space restrictions (see the next section for an example).

5 The Dining Philosophers

The dining philosophers is a classic problem in concurrency theory [16]. Below, we show parts of a GOAL program that implements a solution. The complete program (for one philosopher agent) is listed in Appendix A. The currently implemented version of GOAL uses Prolog as a knowledge representation language, which we also use here. We use numbers to refer to the action rules of the GOAL program. For convenience, when referring to the agent's own mental model in mental state conditions, we drop this parameter.

A number of philosophers are sitting at a round table where they each engage in two activities: thinking and eating (1,2). Our philosophers only think when they are not hungry and get hungry after thinking a while (see the action specifications). At the table an unlimited supply of spaghetti is available for eating. A philosopher needs two forks, however, to be able to eat (3). Forks are available as well, but the number of forks equals the number of philosophers sitting at the table (one fork is between each two philosophers). It is thus is never possible for all of the philosophers to eat at the same time and they have to coordinate. The problem is how to ensure that each philosopher will eventually be able to eat.

Using our conversational metaphor for coordinating activities, the problem of the dining philosophers can be solved elegantly at the knowledge level. In the solution we present, the dining philosophers are assumed to be decent agents that are always willing to listen to the needs of their fellow philosophers at the table, and provide them with the forks when they indicate they require the forks to eat. If a philosopher needs the forks to eat but they are not available, he will initiate a conversation with his neighbors and indicate that he needs the forks (4).[8] If a philosopher i is eating and receives a request for forks from a fellow philosopher X as part of a new conversation, i will finish eating and put down the fork in between X and himself and notify X of this fact (8). A philosopher i will put down a fork only upon being requested. As long as the conversation is ongoing, i will not pick up the fork again. The philosopher that initiated the conversation will pick up the fork after being informed by his neighbor that the fork is on the table (6).[9] The initiator of the conversation informs his neighbors that he picked up the fork (6). Upon receiving a message from both neighbors

[8] In the sent messages the direction of the forks (left, right) has been dropped as this is just a matter of perspective, useful for keeping track of which fork has been picked up or put down from a single philosopher's perspective. From the point of view of two philosophers, a fork is just "in between" them.

[9] In fact, only when having initiated a conversation to require the forks, will a philosopher pick up a fork in our solution.

that they do not know whether the fork is on the table or not (reflected in the mental models of the neighbors), the initiator closes the conversation (7), and another conversation involving one of the philosophers may be started. Rules 5 and 9 are used to update the philosopher's own beliefs on the basis of its mental models of other philosophers (which are changed due to the sending of messages).

```
  % i is the name of this philosopher agent
  beliefs{ hold(fork,left). }  goals{ hold(fork,left), hold(fork,right). }
  program{
   1. if true then think.
   2. if true then eat.
   3. if bel(hungry) then adopt(hold(fork,left), hold(fork,right)).
   4. if goal(hold(fork,_)), bel(not(forksAvailable), neighbours(X,Y))
     then send(new:{X,Y},!hold(fork)).
   5. if bel(neighbour(X,D), not(hold(fork,D))), bel(X, on(fork,table))
     then ins(on(fork,table,D)).
   6. if bel(conversation(Id,i)) then pickUp(fork,D) + send(Id:X, .hold(fork)).
   7. if bel(conversation(Id,i), hold(fork,left), hold(fork,right), neighbours(X,Y))
        bel(X,not(on(fork,table))), bel(Y,not(on(fork,table)))
     then close(Id).
   8. if bel(conversation(Id,X)), goal(X, hold(fork))
     then putDown(fork,D) + send(Id:X, .on(fork,table), not(hold(fork))).
   9. if bel(conversation(Id,X), neighbour(X,D)), bel(X, hold(fork))
     then del(on(fork,table,D)) + send(Id:X, ?on(fork,table)).
  }
  action-spec{
     think{pre{not(hungry)}post{hungry}}
     pickUp(fork,D){pre{on(fork,table,D)}post{hold(fork,D),not(on(fork,table,D))}}
     eat{pre{hungry,hold(fork,left), hold(fork,right)}post{not(hungry)}}
     putDown(fork, D){pre{hold(fork,D)}post{on(fork,table,D),not(hold(fork,D))}}
  }
}
```

6 Conclusion

In this paper, we have introduced an alternative semantics for communication in agent programming languages, based on the idea that a received message can be used to (re)construct a mental model of the sender. We have made this idea precise for the GOAL agent programming language. Also, we have introduced the concept of a conversation to synchronize actions and communication in a multi-agent system. We have shown how these new constructs can be used to program a solution for a classic problem in concurrency theory. We are currently implementing these ideas to allow further experimentation and testing.

References

1. Austin, J.: How to Do Things with Words. Oxford University Press, London (1962)
2. Searle, J.: Speech acts. Cambridge University Press (1969)
3. Singh, M.: A social semantics for agent communication languages. In: Issues in Agent Communication, Springer-Verlag (2000) 31–45
4. Chopra, A., Singh, M.: Constitutive interoperability. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'08). (2008) 797–804

5. Labrou, Y., Finin, T.: A semantics approach for KQML - a general purpose communication language for software agents. In: Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), ACM (1994)
6. FIPA: Fipa communicative act library specification. Technical Report SC00037J, Foundation for Intelligent Physical Agents, Geneva, Switzerland (2002)
7. Singh, M.: Agent Communication Languages: Rethinking the Principles. IEEE Computer **31**(12) (1998) 40–47
8. Vieira, R., Moreira, A., Wooldridge, M., Bordini, R.: Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language. Artificial Intelligence Research **29** (2007) 221–267
9. Wooldridge, M.: Semantic Issues in the Verification of Agent Communication Languages. Autonomous Agents and Multi-Agent Systems **3**(1) (2000) 9–31
10. de Boer, F., Hindriks, K., van der Hoek, W., Meyer, J.J.: A Verification Framework for Agent Programming with Declarative Goals. Journal of Applied Logic **5**(2) (2007) 277–302
11. Dastani, M.: 2APL: a practical agent programming language . Journal Autonomous Agents and Multi-Agent Systems **16**(3) (2008) 214–248
12. Bulling, N., Hindriks, K.V.: Communicating Rational Agents: Semantics and Verification. (2009) submitted.
13. Hindriks, K.V., Boer, F.S.D., Hoek, W.V.D., Meyer, J.J.C.: Semantics of communicating agents based on deduction and abduction. In: Issues in Agent Communication, Springer Verlag (2000) 63 – 79
14. Wooldridge, M.: An introduction to multiagent systems. John Wiley and Sons, LTD, West Sussex (2002)
15. Gärdenfors, P.: Knowledge in Flux: Modelling the Dynamics of Epistemic States. MIT Press (1988)
16. Ben-Ari, M.: Principles of Concurrent and Distributed Programming. Prentice Hall (1990)

A Goal program for the dining philosophers

```
main i { % i, a number between 1 and N, is the name of the philosopher agent
  knowledge{
    neighbour(X,left) :- i>1, X is i-1.
    neighbour(X,left) :- i=1, X is N.     % N is the number of philosophers.
    neighbour(X,right) :- i<N, X is i+1.
    neighbour(X,right) :- i=N, X is 1.
    neighbours(X,Y) :- neighbour(X,left), neighbour(Y,right).
    forkAvailable(D) :- hold(fork,D) ; on(fork,table,D).
    forksAvailable :- forkAvailable(left), forkAvailable(right).
  beliefs{ hold(fork,left). }
  goals{ hold(fork,left), hold(fork,right). }

  program{
    if true then think.         % can only think when not hungry (see action spec)
    if true then eat.           % can only eat when hungry and holding forks

    if bel(hungry) then adopt(hold(fork,left), hold(fork,right)).

    % Initiate conversation with neighbors if you want to eat but forks are not
    % available by sending an imperative: See to it that I hold the fork.
    if goal(hold(fork,_)), bel(not(forksAvailable), neighbours(X,Y))
      then send(new:{X,Y},!hold(fork)).

    % Ongoing conversation initiated by philosopher itself.
    % Only in this case the philosopher will pick up forks.
    if bel(neighbour(X,D), not(hold(fork,D))), bel(X, on(fork,table))
      then ins(on(fork,table,D)).
    if bel(conversation(Id,i)) then pickUp(fork,D) + send(Id:X, .hold(fork)).
    % Close the conversation if I hold both forks and neighours have noticed this.
    if bel(conversation(Id,i), hold(fork,left), hold(fork,right), neighbours(X,Y))
        bel(X,not(on(fork,table))), bel(Y,not(on(fork,table)))
      then close(Id).

    % Ongoing conversation initiated by a neighbouring philosopher
    % Only in this case a philosopher will put down a fork.
    if bel(conversation(Id,X)), goal(X, hold(fork))
      then putDown(fork,D) + send(Id:X, .on(fork,table), not(hold(fork))).
    if bel(conversation(Id,X), neighbour(X,D)), bel(X, hold(fork))
      then del(on(fork,table,D)) + send(Id:X, ?on(fork,table)).
  }
  action-spec{
     think{pre{not(hungry)}post{hungry}}
     pickUp(fork,D){pre{on(fork,table,D)}post{hold(fork,D),not(on(fork,table,D))}}
     eat{pre{hungry,hold(fork,left), hold(fork,right)}post{not(hungry)}}
     putDown(fork, D){pre{hold(fork,D)}post{on(fork,table,D),not(hold(fork,D))}}
  }
}
```