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. In the area of agent programming, the challenge is to introduce communication primitives that are useful to a programmer of agent programs as well as semantically well-defined. Moreover, for agents that derive their choice of action from their beliefs and goals it is natural to introduce primitives that support communication related to both of these attitudes. We introduce a communication approach for multi-agent systems based on *mood operators* instead of the usual speech act labels and a semantics based on the idea that a message can be used to *(re)construct a mental model* of the sender. An operational semantics is provided that specifies the precise meaning of the primitives. Finally, to facilitate *coordination* in multi-agent systems, we introduce the concept of a *conversation* to synchronize actions and communication among agents. Conversations provide a limited resource at the multi-agent level, and provide a natural approach for multi-agent systems to coordinate agent activities.

1 Introduction

Communication is key in a multi-agent system for agents to exchange information and coordinate their activities. For this reason, the design of constructs that facilitate communication in agent programming languages also is an important aspect that must be addressed in these programming languages. In particular, for agents that derive their choice of action from their beliefs and goals it is natural to introduce communication primitives that support communication related to both of these attitudes.

We argue that it is not sufficient to provide only communication primitives that facilitate the *exchange of messages*, but it is also necessary to provide explicit support for the *coordination* of agent communication and activities. It often occurs that agents need to decide on which agent will perform a particular task, whether it is the turn of an agent to make a move, or synchronize their activities for other reasons. It is useful to provide programming constructs that facilitate such coordination.

In this paper we address both aspects discussed above, and propose a communication language that facilitates both information exchange as well as the

coordination of agent activities. We take a definite *engineering stance* here. Although we think it is very important to introduce communication primitives that are semantically well-defined and motivated adequately from a theoretical point of view, our main concern is to provide *useful* communication facilities to a programmer of rational agents. An approach is presented that addresses each of these issues: the meaning of the communication constructs are defined using a formal semantics, are theoretically motivated, and - so we believe - address the pragmatic component as the primitives introduced are relatively easy to grasp, which will facilitate programmers that will need to make good use of these constructs.

The paper is organized as follows. In section 2 we briefly discuss related work in agent programming languages with respect to communication. Then in section 3 we present our approach for integrating agent communication into an agent programming language informally first and motivate some of our choices. Section 4 introduces the formal semantics for the various communication facilities. Section 5 illustrates all communication constructs, using the dining philosophers as an example. Finally, section 6 concludes the paper.

2 Communication in Agent Programming Languages

It has been common in agent programming to integrate communication constructs into the programming language using constructs that are based on *speech act theory* [1, 2]. This theory has underpinned much work on agent communication languages such as KQML [3] and FIPA [4], two of the most well-known agent communication languages available. The JADE implementation [5] that is said to be compliant with the FIPA specifications is one of the most used platforms for implementing agents, as it not only facilitates the FIPA list of speech acts but also provides a middleware infrastructure that facilitates message exchange. One of the distinguishes features of any approach based on speech acts is the list of *performative labels* that may be used to label possible speech acts that an agent can perform, such as *inform*ing, *ask*ing, and *request*ing to name but a few of the more well-known speech acts.

One of the approaches to integrating communication into an agent programming language is to explicitly integrate the communication primitives that a platform such as JADE provides. The interpreters of quite a few agent programming languages are built on top of JADE in order to have a middleware layer that facilitates message exchange. It is then relatively easy to provide the communication primitives of JADE as constructs that can be used in an agent program. This is more or less the approach taken in the programming language Jadex [6]. Communication in JACK [7] also seems to be based on a similar approach, but the implementation has not been based on JADE but on a specific implementation for JACK itself.

The integration of communication has been necessarily somewhat pragmatic in order to be able to deal with the specifics of each programming language. In particular, the means to handle received messages and the effects of messages

vary across languages. In 2APL, a successor of 3APL [8], the format of messages as specified in FIPA is basically preserved [9], and the same performative labels available in FIPA may be used as 2APL is built on top of JADE. Messages that are received are stored in a message base and so-called procedure call rules are used to allow an agent to react or respond to messages received. Different from the languages discussed above, the semantics of communication in 2APL is formally specified. The meaning of communication primitives and messages is defined by as a simple "mailbox" semantics: communicating a message means that the message is added to a mailbox and the programmer then needs to write rules to handle these messages.

Interestingly, a somewhat different approach is used for providing communication in the agent language Jason[10]. Although the approach is based on speech acts as well, instead of providing a long list of performative labels the programmer of a Jason agent can choose from a relatively small number of available labels, derived from KQML. We think restricting the set of labels to a limited set in the context of agent programming is a sensible thing to do for two reasons: (i) Offering a broad range of performative labels may confuse a programmer and complicate the design of agents, and (ii) it is difficult to avoid subtle differences in the semantics for two or more labels that are hard to distinguish by programmers.[1] For the set of labels available in Jason a formal semantics is defined that in some respects is similar to that specified in FIPA as far as the effects of communication are concerned. However, Jason does not require any specific preconditions to be true before an agent may send a message. A Jason-specific feature is the use of so-called "annotations" to label a message with the identity of the source of this information. These annotations were motivated specifically by the design of the communication semantics [10]. For example, the *tell* message inserts the content c of the message into the receiver's belief base, labeled with an annotation s to identify the source of this information; that is, $c[s]$ is inserted into the belief base. As another example, the *achieve* message inserts the content of the corresponding message in the event base. Moreover, the *ask* primitive provides a limited means to synchronize agents as the asking agent may wait for an answer from the agent that is being asked to address the message. Finally, abstract functions are introduced to determine whether a message is "socially acceptable", and only socially acceptable messages are processed.

Summarizing, all approaches in agent programming are based on the speech act paradigm, and more or less are based on KQML/FIPA. Some agent programming languages, such as 2APL and Jason, also provide a formal semantics that specifies precisely what happens when a message is sent. Although FIPA also specifies the preconditions and rational effects of the performance of specific speech acts, in practice there is no programming language that actually implements these specifications. This is in part due to the fact that certain features required by FIPA are not supported by agent programming languages, and in

[1] An example to illustrate this issue is the subtle difference in meaning of the `confirm` and `inform` speech acts in FIPA [4].

part due to more fundamental issues that gave rise to a shift from the sender to that of the receiver.

The agent programming language that we take as our starting point in the remainder of this paper is the GOAL language [11]. This language has in common with 2APL and Jadex that agents have mental states that consist of *declarative beliefs and goals*. The presence of declarative beliefs and goals naturally induces the question how a rational agent can be provided with primitives that support communicating information related to both these mental attitudes, an issue that has not been explicitly addressed in the context of agent programming. Mental states also provide the means to define a semantics of communication that makes explicit the idea that a receiver reconstructs a mental model of the sender, in line with the noted shift from sender to receiver. The semantics that we will introduce here explicitly uses messages received to *model the sender*. Another contribution of this paper is the introduction of an explicit mechanism for synchronizing and coordinating agent activities based on the concept of a *conversation*. This concept seems related to that of an artefact or possibly workspace in the simpA language [12], and may be effectively implemented using constructs provided in this language.

3 Design of Communication Approach

Apart from a shift from the sender to the receiver, current approaches to communication in agent programming still use *speech act labels* to tag messages. The reason for this shift can be explained by some of the criticisms that have been leveled against the use of speech act semantics for implementing agent communication, see e.g. [13, 14]. Speech act theory may be adequate as a *descriptive* theory that specifies the conditions that identify the particular type of communicative act that has been performed when a sentence is uttered (or message exchanged). It does not make so much sense anymore, however, when the theory is interpreted as a *recipe* for executing actions specified by means of pre- and postconditions as is traditional in computing science.[2] Part of the problems with speech act semantics, in particular the so-called *sincerity conditions* that should hold with respect to the sender thus have been dropped in order to be able to make practical use of the labels that are used to classify various acts. These conditions have not been incorporated into an operatorional semantics for communication for two reasons. First, it is impossible for a receiving agent to *verify* whether an agent speaks truthfully upon receiving a message of the form $inform(..)$, making the variety of labels not very useful for the purpose of identifying the act that has been performed (the reason for naming these labels as they have been) [13, 14]. Second, a sender would be unnecessarily constrained

[2] To highlight the difference we intend to convey here, an example may be useful: A label such as *misinform* or *lie* makes perfect sense in a *descriptive* theory but not so much as part of a message that is being transmitted, as usually one attempts to hide that one is deceiving.

by imposing such conditions and would no longer be able to "inform" an agent of a statement it believes to be false, i.e. lie.

As discussed above, there is a second, more pragmatic reason to deviate from the speech act paradigm in the context of agent programming, namely the fact that introducing a relatively high number of performative labels with possibly subtle semantical differences complicates agent design. The choice of labels, moreover, is different from theory to theory and it is not clear which set is to be preferred (compare e.g. KQML and FIPA). We argue below that these issues can be resolved by using *mood operators* instead of speech act labels, a choice that also has a strong basis in linguistic theory.

We briefly discuss an alternative approach to the FIPA-style semantics based on so-called *social commitments*, see e.g. [15–17]. There is a huge literature on the topic of *social semantics* which is impossible to survey here. A social semantics for speech acts may be contrasted with a semantics based on mental states. Whereas social commitments, the main entities in a social semantics, are supposed to be *public*, mental states are supposed to be *private*. An advantage of social commitments therefore is that there is no need to reconstruct and attribute them to other agents [15]. The basic idea of social commitments is that speech acts do have public and objective effects with which both sender and receiver can always be confronted again; a receiver may always say, for example, something like: "You told/asked/requested me so". Although the processing of the message by the receiver has moved to the background here, from the perspective of agent programming this is not necessarily an advantage. As agent-oriented programming may also be paraphrased as "programming with mental states", it is important to clarify how social commitments relate to the mental attitudes of agents. We recognize that a social semantics may be complementary to a semantics based on mental attitudes and some work to revolve this issue has been reported in, for example, [18]. The main issue seems to be how to relate commitments which are supposed to be *publicly available* to mental states which are supposed to be *private*. This issue, we believe, is not easily resolved, and in this paper, we focus on a semantics that is based on mental models as they represent the entities that an agent computes with in agent programming.

3.1 Communication Semantics

We take an *engineering stance* towards the design of a communication semantics that fits well with the agent-oriented programming paradigm. Related to this, issues such as how useful communication primitives are to a programmer, whether such primitives facilitate communication about the beliefs and goals of rational agents, and the range of applications that these primitives have need to be considered. This means that communication primitives should support, e.g., the exchange of reasons for acting based on beliefs and goals, and should be provided with a relatively easy to grasp semantics that fits basic intuitions.

The starting point for the semantics introduced here is the idea that a message can be used to *(re)construct a mental model* of the sender. The content of a message is not a speech act per se but a speech act is inferred from a message.

This idea is somewhat related to the *inferential* approach to speech act theory advocated in [19]. In particular, some of our ideas are inspired by the work in theoretical linguistics of Harnish [20] and we aim to provide a framework for agent communication based on some of these ideas. It should however also be noted that we have simplified this work in line with some of the pragmatic issues discussed above. A second idea is to use *mood operators* instead of speech act labels. Here, we take inspiration from natural language to differentiate between various communication modes. Mood, in the sense that we use it here and in line with linguistic theory, refers to "sentential form with a function". We follow [20] and limit the discussion to the three *major moods* in natural language:

- *declarative* mood, e.g. "Snow is white." Typically, the literal and direct use of a declarative sentence is to make statements (about the environment).
- *interrogative* mood, e.g. "Is it snowing?". One of the typical literal and direct uses of an interrogative is, for example, to inquire about a state of affairs.
- *imperative* mood, e.g. "Leave the room!". Typically, the literal and direct use of an imperative is to direct someone to establish a state of affairs.

These moods are recognized as central in their communicative importance and to occur comparatively high in frequency [20], and therefore also seem most important to include in a language for agent communication. Corresponding to each of these moods, mood operators are introduced and $\mathbf{:}\phi$ is used to indicate declarative mood, $\mathbf{?}\phi$ is used to indicate declarative mood, and $\mathbf{!}\phi$ is used to indicate imperative mood.[3]

Returning to one of our main goals, that of defining a semantics for agent communication, we start by discussing some of the ideas presented in [20]. Harnish presents a set of strategies accounting for the *literal and direct* use of declaratives, imperative and interrogative sentences. As a first approximation, Harnish suggests that a hearer upon perceiving that S utters "Leave the room!" is allowed to infer that S is *directing* that someone to leave the room, and the request (etc) is *complied with* just in case someone leaves the room. These strategies thus allow to infer the *force* and the *condition of satisfaction* related to the uterrance. Harnish suggests that this process proceed in two stages: "first, there is an inference from form to expressed attitude; then there is an inference from expressed attitude to force". The expressed attitude of a declarative $\mathbf{:}\phi$ is a believe that ϕ, of an interrogative $\mathbf{?}\phi$ it is a desire that hearer tells whether that ϕ, and of an imperative $\mathbf{!}\phi$ it is a desire that hearer makes it the case that ϕ. Inferences to (illocutionary) force in stage two then are restricted and only support inferences to those speech acts whose conditions require the expressed attitude [20].

We adapt the proposal of Harnish here in two ways, which better meets our pragmatic concerns to provide a relatively easy to understand semantics. First, we do not want to complicate the representational means needed to express these conditions, i.e. at this stage we do not want to complicate things by introducing modal operators in the databases that agents maintain but leave this for future work. The expressed attitudes that we propose therefore are as follows: upon

[3] This notation has also been used by Pendlebury [21].

receiving a message **:**ϕ r concludes that sender s believes ϕ; from **?**ϕ r concludes that s does not know whether ϕ; and, from **!**ϕ r concludes that s has ϕ as a goal, and does not believe ϕ to be the case. Second, we do not incorporate stage two into our semantics. The actual conclusion as to which speech act has been performed is left to the agent; that is, the programmer needs to either supply the agent with explicit inference rules to derive speech act types, or leave these implicit in the design of the agent (which we expect will be easier in practice). A message of thus possibly allows for multiple interpretations as to which speech act is performed [20].

We further motivate and illustrate this semantics briefly using an example (see also [22]). Consider the utterance "The house is white" and let p denote the proposition that is expressed. In line with the strategies discussed above for the literal and direct use of an utterance such as "The house is white", we have chosen to incorporate effect 2 as the default interpretation in our semantics. Obviously, this is not always a safe assumption to make, as the sender may be lying, but it is also not overly presumptuous. Other "effects" such an utterance might have on the mental state of a receiver could be: (i) The receiver comes to believe that the house is white, (ii) The receiver comes to believe that the sender had the intention to make the receiver believe that the house is white,and (iii) The utterance has no effect on the receiver, i.e. its mental state is not changed as a result of the utterance. Even though each of these other interpretations may be waranted given specific circumstances of the speaker and the hearer and the knowledge they have about each other, these interpretations do not correspond with the literal and direct use of an utterance [20]. In general, we consider effect (i) too strong, as it implicitly assumes that the sender always convinces the receiver; effect (ii) too indirect, and not very useful from a programmer's point of view either as rather involved reasoning on the part of the agent seems to be required to make good use of such indirect conclusions about the sender's state of mind; and, finally, effect (iii) too weak, as it is not very useful for programming communication among agents since no effect would occur.

Summarizing, the communication semantics that we propose records the *expressed attitude* of the sender in a *mental model* of that sender maintained by the receiving agent. Declaratives express a belief of the sender, interrogatives a lack of belief, and imperatives desires or goals. Messages thus never directly impact the beliefs or goals of a receiving agent. We do allow, of course, that an agent updates his own beliefs and goals using his model of that of other agents. An agent also may use the mental models of other agents it maintains to decide which action to perform next, which is illustrated in the program of section 5).

3.2 Conversations

A second contribution of this paper is the concept of a *conversation* to facilitate the synchronization of actions and communication in a multi-agent system, which is particularly important to coordinate agent activities.

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.[4] 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 [23]. We introduce a mechanism that fits elegantly into the overall setup of communication primitives introduced above, using the notion of a *conversation*.

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 a key 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.[5] A mental model associated with a GOAL agent needs to satisfy a number of *rationality constraints*.

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:* $\forall \gamma \in \Gamma : \Sigma \not\models \gamma$

[4] 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'* [23]. To resolve this, solutions in concurrency theory contain asymmetries in the form of process identifiers or a kernel maintaining a queue.

[5] 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. [24]) but not considered here.

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 [25].

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.[6]

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

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

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 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 [25] 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*

[6] 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.

conditions is defined by:

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

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, $a \in Act$ 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 $\mathbf{send}(i,msg)$ where i is an agent name and msg is a message of the form $\mathbf{:}\phi$, $\mathbf{?}\phi$ 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:

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

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** $\mathtt{a}$, where $\mathtt{a}$ 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 communication.

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

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

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 $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ and the environment E, which is used to store messages that have been sent and are waiting for delivery.[7] 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 $send(j, msg)$.

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

The following transition rule specifies the semantics of sending messages.

Definition 7. (Send)

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

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

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).[8]

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)

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

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 $\textbf{?}\varphi$ 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)

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

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* $k \neq i$.

[8] The implicit quantifier **allother** may be used to define a broadcasting primitive: $\textbf{broadcast}(msg) \stackrel{df}{=} \textbf{send}(\textbf{allother}, msg)$. 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. As explained above, this would require including modal operators $K_i\phi$ in the goal base, and we leave such complications for future work.

The rule below for imperatives formalizes that if agent i communicates a message $!\phi$ with imperative mood operator, then the receiver j will conclude that i does not believe ϕ, 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)

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

where:

- $m'(i) = \langle \Sigma \ominus \phi, \Gamma \cup \{\phi\}\rangle$ *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 explained, the idea of a conversation 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 number of conversations that an agent can participate in simultaneously thus introduces a limit on access to that resource. For our purposes, it suffices to restrict participation to at most one conversation at any 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 $\mathbf{send}(c : i, g, msg)$, where agents i and g are participants. We write $agent(A, Q)$ to denote the set of agents in A and Q. The reserved **new** label is used to have the system automatically generate a new conversation identifier.

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)

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

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 [23]. In the Table below, a complete GOAL program (for one of the philosopher agents) is listed that implements a solution. 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.

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.

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

    3. 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.
    4. 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.
    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)).
    % Close the conversation if I hold both forks and neighours have noticed this.
    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).

    % Ongoing conversation initiated by a neighbouring philosopher
    % Only in this case a philosopher will put down a fork.
    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))}}
  }
}
```

The solution uses the conversational metaphor for coordinating activities. 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).[9] According to Definition 12, a conversation will be started if no conversation initiated by the agent is part of the queue or ongoing, thus preventing a philosopher from continuously asking for the forks. Each of the philosophers competing for the works will thus be able to initiate at most one conversation to aks for works, which are ordered automatically in the queue and the request to start a conversation thus will be eventually initiated. 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).[10] The initiator of the conversation informs his neighbors that he picked up the fork (6). Upon receiving a message from both neighbors 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).

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. 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)
4. FIPA: Fipa communicative act library specification. Technical Report SC00037J, Foundation for Intelligent Physical Agents, Geneva, Switzerland (2002)

[9] 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.

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

5. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley (2007)
6. Braubach, L., Pokahr, A., Lamersdorf, W. In: Software Agent-Based Applications, Platforms and Development Kits
7. Howden, N., Ronnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents - summary of an agent infrastructure. In Wagner, T., Rana, O.F., eds.: Proceedings of the 5th ACM International Conference on Autonomous Agents, Workshop on Infrastructure for Agents, MAS and Scalable MAS. (2001) 251 257
8. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.J.C.: Agent Programming in 3APL. Autonomous Agents and Multi-Agent Systems **2**(4) (1999) 357–401
9. Dastani, M.: 2APL: a practical agent programming language . Journal Autonomous Agents and Multi-Agent Systems **16**(3) (2008) 214–248
10. 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
11. Hindriks, K.V.: Programming Rational Agents in GOAL. In: Multi-Agent Programming: Languages, Tools and Applications. Springer (2009) 119–157
12. A. Ricci, M.V.V., Piancastelli, G.: simpa: A simple agent-oriented java extension for developing concurrent applications. In: Proc. of the Workshop on Languages, Methodologies and Development Tools for MAS (LADS'007). (2007) 176–191
13. Singh, M.: Agent Communication Languages: Rethinking the Principles. IEEE Computer **31**(12) (1998) 40–47
14. Wooldridge, M.: Semantic Issues in the Verification of Agent Communication Languages. Autonomous Agents and Multi-Agent Systems **3**(1) (2000) 9–31
15. Colombetti, M.: A commitment–based approach to agent speech acts and conversations. In: Proc. Workshop on Agent Languages and Communication Policies, 4th International Conference on Autonomous Agents (Agents 2000). (2000) 21–29
16. Singh, M.: A social semantics for agent communication languages. In: Issues in Agent Communication, Springer-Verlag (2000) 31–45
17. 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
18. Boella, G., Damiano, R., Hulstijn, J., van der Torre, L.: Role-based semantics for agent communication: embedding of the 'mental attitudes' and 'social commitments' semantics. In: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. (2006) 688–690
19. Bach, K., Harnish, R.M.: Linguistic Communication and Speech Acts. The MIT Press (1979)
20. Harnish, R.M.: Mood, Meaning and Speech Acts. In: Foundations of Speech Act Theory: Philosophical and Linguistic Perspectives. Routledge (1994) 407–459
21. Pendlebury, M.: Against the power of force: reflections on the meaning of mood. Mind **95** (1986) 361–372
22. Wooldridge, M.: An introduction to multiagent systems. John Wiley and Sons, LTD, West Sussex (2002)
23. Ben-Ari, M.: Principles of Concurrent and Distributed Programming. Prentice Hall (1990)
24. Gärdenfors, P.: Knowledge in Flux: Modelling the Dynamics of Epistemic States. MIT Press (1988)
25. 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