Operational Semantics for Agent Communication Languages

Rogier M. van Eijk, Frank S. de Boer,
Wiebe van der Hoek, and John-Jules Ch. Meyer

Utrecht University, Department of Computer Science
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{rogier, frankb, wiebe, jj}@cs.uu.nl

Abstract. In this paper, we study the operational semantics of agent communication languages. We develop a basic multi-agent programming language for systems of concurrently operating agents, into which agent communication languages can be integrated. In this language, each agent has a mental state comprised of an informational component and a motivational component; interaction between the agents proceeds via a rendezvous communication mechanism. The language builds upon well-understood concepts from the object-oriented programming paradigm as object classes, method invocations and object creation. The formal semantics of the language are de ned by means of transition rules that describe its operational behaviour. Moreover, the operational semantics closely follow the syntactic structure of the language, and hence give rise to an abstract machine to interpret it.

1 Introduction

The research on agent-oriented programming has yielded a variety of programming languages for multi-agent systems, each of which incorporates a particular mechanism of agent communication. Many of these communication mechanisms are based on *speech act theory,* originally developed as a model for human communication. A speech act is an action performed by a speaker to convey information about its mental state with the objective to in uence the mental state of the hearer. This notion has been fruitfully adopted in agent communication languages as KQML [10] and FIPA-ACL [11], which prescribe the syntax and semantics of a collection of speech act-like messages. Such a message is comprised of a performative, a content and some additional parameters as the sender and receiver of the message; e.g., there is a message to inform a receiving agent that the sender believes a particular formula to be true, and a message to request a receiving agent to perform some particular action. Both agent communication languages assume an underlying communication mechanism that proceeds via asynchronous message-passing.

As was indicated by Cohen and Levesque (cf. [4]) communicative acts should be considered as *attempts* of the sending agent to get something done from the receiving agent. An important consequence of this view is that there is no guarantee that the receiving agent will actually act in accordance with the purposes the sending agent has attributed to the message. That is, it is possible that a receiving agent will simply ignore

F. Dignum and M. Greaves (Eds.): Agent Communication, LNAI 1916, pp. 80–95, 2000.

the message or will do the opposite of what it is requested for. Hence, in giving semantics to messages one should not confuse the *effects* that a message has on the receiving agent with the subsequent *reactions* taken by the receiver that it brings about. For instance, one could easily be tempted to describe the meaning of an $ask(\varphi)$ message as that upon its reception, the receiving agent should respond with a $confirm(\varphi)$ message in the situation it believes the formula φ to be true, and with a $disconfirm(\varphi)$ message if it this is not the case. In our opinion however, the reactions towards a message are not part of the semantics of the message, but rather are consequences of the characteristics of the receiving agent. The point we advocate in this paper is to describe the semantics of messages solely in terms of the effects they have on the mental state of the receiving agent, without giving any references to the possible subsequent reactions. For instance, the meaning of an $ask(\varphi)$ message would be that upon its reception the receiving agent registers that it is asked whether it believes φ to hold. It will be on the basis of its altered mental state that the agent will subsequently decide what actions to perform. This behaviour is however not considered to be part of the semantics of the message. Moreover, we give an *operational* semantics of agent communication (in contrast for instance with the approach of Colombetti [5], which is based on modal logic).

Types Of Communication. In the research on intelligent agents, it is common practice to think of an agent as an entity having several different mental attitudes. These attitudes are divided into a category of *information attitudes*, which are related to the information an agent has about its environment such as belief and knowledge, and secondly, a class of *pro-attitudes* which govern the actions that the agent performs in the world and hence, concern the agent's motivations. Examples of the latter category of attitudes are goals, intentions and commitments (cf. [16]). Our previous work on communication in multi-agent systems has been centred around the agents' information attitudes (cf. [6,8,7,9]). A central topic in this study is the development of a communication mechanism that allows for the *exchange of information*. In this report, we consider communication that concerns the agents' pro-attitudes; that is, we study the communication of *actions and programs to be executed*. In particular, we will focus on the motivations that directly stem from communication, and refer the reader to [13] for details on non-communicative motivations.

We distinguish between two different types of interaction. First of all, there is communication that concerns *properties* of the agent system, which are not considered as to give information about the current agent system but rather as *speci cations* of a pursued con guration of the agent system. An example of a communicative act that falls in this category is the KQML message $\mathbf{achieve}(i, j, \varphi)$, which denotes a request of the agent i to the agent j to accomplish a state in which φ is true. This type of communication typically involves planning of actions, which is an issue that we will not address here.

Secondly, there is communication that involves executable programming code to change the agent system; i.e., the communication concerns implementation rather than speci cation, or in other words, is procedural of nature instead of declarative. An example of a communicative act in this category is the FIPA-ACL message $\langle i, \mathsf{request}(j, a)\rangle$, which denotes a request of the agent i directed to the agent j to execute the action a. This type of interaction is similar to that of *higher order communication,* studied in the eld of concurrency theory. In this paradigm, programs themselves constitute rst class

communication data that can be passed among the processes in the system (cf. [15]). Upon reception of a program, the receiving process integrates it in its own program and subsequently resumes its computation. Although communication of mobile code gives rise to a very powerful computation mechanism, it also gives rise to a wide range of problems especially with respect to safety and security issues in computer systems, like the unauthorised access to and manipulation of sensitive data and services.

In this paper, we will not follow the road of higher order communication, but adopt a more traditional communication mechanism that has been fruitfully employed in various distributed programming paradigms. It amounts to the idea that rather than accepting and executing arbitrary programs, a process specifies a collection of programs that other processes can request it to execute.

Rendezvous and Remote Procedure Call. In the field of concurrency, there is the classical notion of a *rendezvous*, which constitutes a communication mechanism in which one process j executes one of its procedures on behalf of another process i (cf. [2]). In particular, a rendezvous can be viewed upon as to consist of three distinct steps. First, there is the call of a procedure of j by i. This is followed by the execution of this procedure by j, during which the execution of the calling process i is suspended. Finally, there is the communication of the result of the execution back to i, which thereupon resumes its execution. It follows that a rendezvous comprises two points of synchronisation. First, there is the call with the exchange of the actual procedure parameters from the caller to the callee and secondly, there is the communication of the results back from the callee to the caller

The notion of a rendezvous is almost equal to that of a *remote procedure call* (RPC) (cf. [3]). The difference between the two notions lies in the fact that for an RPC an entirely new process is created that takes care of the call, whereas in the case of a rendezvous the call is handled by the called process itself (cf. [2]). This implies that in the former case different procedure calls can be handled simultaneously, whereas in the latter case the calls are taken one at a time.

The rendezvous communication mechanism has for instance been adopted in the concurrent programming language POOL (cf. [1]), which constitutes a semantically well-understood programming language for systems of concurrently operating *objects*. In this paper, we outline a framework for agent communication that builds upon the object-oriented features of languages like POOL. The most important aspect we have to take into account is that in agent-oriented programming computations are not performed relative to a *local state* that maps variables to their associated values but by to *mental state* that consists of attitudes as beliefs and goals.

The remainder of this paper is organised as follows. In Section 2, we define a general multi-agent programming language for systems of agents that interact with each other by means of a rendezvous communication scheme. In Section 3, develop an operational model of the programming language by means of a transition system. We study the relation of the framework with the existing agent communication languages KQML and FIPA-ACL in Section 4. Finally, we round off in Section 5 by suggesting several issues for future research.

2 Syntax

In this section, we de ne a general programming language for multi-agent systems that covers the basic ingredients to describe the operational semantics of agent communication languages. In particular, we assume that a multi-agent system consists of a dynamic collection of agents in which new agents can be integrated, and where the interaction between the agents proceeds via a rendezvous communication scheme. Each agent is assigned a mental state comprised of a belief state and a goal state that can be inspected and updated. Additionally, an agent is assumed to be an instance of an agent class that de nes the methods the agent can invoke itself as well as can be called by other agents in the system. Finally, the behaviour of an agent is governed by a program consisting of the standard programming constructs of sequential composition, non-deterministic choice, parallel composition and recursion.

De nition 1 *(First-order language)*

- A *signature* $\mathcal{L}$ is a tuple $\langle \mathcal{R}, \mathcal{F} \rangle$, where $\mathcal{R}$ is a collection of predicate symbols and $\mathcal{F}$ a collection of function symbols. The 0-ary function symbols are called *constants*. We assume a set $Ident$ of constants to represent agent identi ers with typical elements i, j and k.
- We assume a set Var of logical variables with typical elements x, y and z.
- The set Ter (typical element t) of terms over $\mathcal{L}$ is the smallest set that satis es $Var \subseteq Ter$ and secondly, if $t_1, \ldots, t_k \in Ter$ and $F \in \mathcal{F}$ of arity k then $F(t_1, \ldots, t_k) \in Ter$. A term is *closed* if it contains no variables from Var.
- The set For (typical element φ, ψ) is the smallest set that contains a set $ForVar$ of formula variables (typical element X) and additionally satis es: if $t_1, t_2, \ldots, t_k \in Ter$ and $R \in \mathcal{R}$ of arity k then $(t_1 = t_2),\ R(t_1, \ldots, t_k) \in For$ and if $\varphi, \psi \in For$ and $x \in Var$ then $\neg\varphi,\ \varphi \wedge \psi,\ \exists x\varphi \in For$.

We de ne a mechanism of inferring conclusions from rst-order information stores, which accounts for conclusions that contain free variables (cf. [9]). The idea of the derivation of a formula $?\mathbf{x}\psi$ from an information store φ is that the free occurrences of the variables $\mathbf{x}$ (we use boldface notation to denote *sequences*) are substituted for in ψ by a closed term such that the resulting formula becomes a consequence of φ.

De nition 2 *(Ground substitutions)* A ground substitution Δ is modeled as a nite conjunction of equalities of the form $x = t$, where $x \in Var$ and t is a closed term in Ter. We require that Δ binds each variable to at most one term. Moreover, we will use $dom(\Delta)$ to denote the variables for which Δ is de ned.

For all formulae φ and substitutions Δ, we de ne $\varphi \oplus \Delta$ to be the formula φ in which all free occurrences of the variables of $dom(\Delta)$ have been substituted for by their corresponding value in Δ.

De nition 3 *(Ground uni er s)* Given the formulae $\varphi, \psi \in For$ and the sequence $\mathbf{x}$ of variables, a substitution Δ is called a *ground uni er* for the pair $(\varphi, ?\mathbf{x}\psi)$, if $dom(\Delta) = \mathbf{x}$ and $\varphi \vdash (\psi \oplus \Delta)$, where $\vdash$ denotes the standard rst-order entailment relation.

We will use the notation $\varphi \vdash_{\Delta} \psi$ to denote that Δ is a ground uni er for (φ, ψ). For instance, the substitution $x = a$ is a ground uni er for (φ, ψ), where φ is given by $\forall x(P(x) \rightarrow Q(x)) \wedge P(a) \wedge P(b)$ and ψ equals $?xQ(x)$, and so is the substitution $x = b$. This shows that ground uni ers are not necessarily unique.

We assume a set $Func$ of functions that map formula variables to formulae, with typical element θ. We assume a function $\cdot$ such that $\varphi \cdot \theta$ yields the formula φ in which all formula variables have been replaced by the formula given by the function θ. Additionally, we use the notation $\theta\{\varphi/X\}$ to denote a function that yields φ on the input X and the value yielded by the function θ on all other inputs. For instance, we have that $\exists x(x = a \wedge X_1 \wedge X_2) \cdot \theta$ where $\theta = \theta'\{P(a)/X_2\}\{\neg Q(a)/X_1\}$, is de ned to be the formula $\exists x(x = a \wedge \neg Q(a) \wedge P(a))$.

We assume a set $Class$ of agent classes with typical element C, a set $Meth$ of methods with typical element m and an operator $\circ$ to update belief states. Additionally, the set $Goal$ is comprised of goal states with typical element G. A goal state is a set of elements of the form $m(\varphi) \Rightarrow i$, denoting a goal to execute the method m with actual parameter φ (we consider methods with only one parameter, the generalisation to methods with more than one parameter is obvious). The constant i denotes the agent to which the result of the execution is to be sent back.

De nition 4 *(Syntax of programming language)*

$$a ::= X \leftarrow \varphi \mid X \leftarrow a \mid \mathsf{query}(?\mathbf{x}\varphi) \mid \mathsf{update}(\varphi) \mid m(\varphi) \mid \mathsf{integrate}(C, S) \mid$$
$$\mathsf{request}(\iota x.\psi, m(\varphi)) \mid \mathsf{accept}(\mathbf{m}) \mid \mathsf{handle}(\mathbf{m})$$

$$S ::= a; S \mid S_1 + S_2 \mid S_1 \,\&\, S_2 \mid \mathsf{skip}$$

$$A ::= \langle i, S, (\varphi, G), \theta \rangle$$

If an *action* a is of the form $X \leftarrow \varphi$ it denotes the assignment of the formula φ to the formula variable X, while an action of the form $X \leftarrow a$ stands for the assignment of the result of executing the action a to X. The action $\mathsf{query}(?\mathbf{x}\varphi)$ denotes the query whether the formula φ follows from the agent's belief state modulo a ground substitution of the variables $\mathbf{x}$, while the action $\mathsf{update}(\varphi)$ represents the update of the belief state with φ. An agent executes the action $\mathsf{request}(\iota x.\psi, m(\varphi))$ to send an agent x that satis es the formula ψ a request to execute the method m with actual parameter φ. An agent executes the action $m(\varphi)$ to invoke its method m with parameter φ. The action $\mathsf{accept}(\mathbf{m})$ denotes the acceptance of a request to execute a method in $\mathbf{m}$, while the action $\mathsf{handle}(\mathbf{m})$ is employed to select and execute an invocation of one of the methods $\mathbf{m}$ from the goal state. The difference between both actions is that the former lls the goal state whereas the latter empties it. Finally, the action $\mathsf{integrate}(C, S)$ is used to integrate an instance of the agent class C in the agent system, which thereupon will start to execute the statement S. A *class* is de ned as a collection of method declarations of the form $m(\varphi) :- S$, where m is the name of the method, φ denotes its formal parameter and S is its body statement.

A *statement* is either the sequential composition $a; S$ of an action and a statement, the non-deterministic choice $S_1 + S_2$ between two statements, the parallel composition $S_1 \,\&\, S_2$ of two statements or an empty statement skip.

An *agent* A consists of an identi er i that distinguishes it from all other agents, a program S that governs its behaviour, a mental state (for which we will use the symbol Λ) consisting of a belief state φ and a goal state G. For technical convenience we also include in the agent con guration an auxiliary function θ that is used to map formula variables to formulae. Implicitly, we assume that each agent is an instance of an agent class, which means that the methods the agent can invoke are those that are de ned in its class. Finally, a *multi-agent system* $\mathcal{A}$ is a set of agents.

3 Operational Semantics

An elegant mechanism of de ning the operational semantics of a programming language is that of a transition system. Such a system, which was originally developed by Plotkin (cf. [14]) constitutes a means to formally derive the individual computation steps of a program. In its most general form a *transition* looks as: $P, \sigma \longrightarrow P', \sigma'$ where P and P' are two programs and σ and σ' are some stores of information. The transition denotes a computation step of the program P which changes the store of information σ to σ', where P' is identi ed to be the part of the program P that still needs to be executed. Transitions are formally derived by means of *transition rules* of the form:

$$\frac{P_1, \sigma_1 \longrightarrow P_1', \sigma_1' \quad \cdots \quad P_n, \sigma_n \longrightarrow P_n', \sigma_n'}{P, \sigma \longrightarrow P', \sigma'}$$

Such a rule denotes that the transition below the line can be derived if the transitions above the line are derivable. Sometimes, we will write transition rules with several transitions below the line. They are used to abbreviate a collection of rules each having one of these transitions as its conclusion. A rule with no transitions above the line is called an *axiom*. A collection of transition rules de nes a *transition system*.

The advantage of using transitions systems is that they allow the operational semantics to closely follow the syntactic structure of the language. As an effect, if we view the con gurations P, σ as an abstract model of a machine then the transitions specify the actions that this machine can subsequently perform. In fact, this machine would act as an interpreter for the language.

The transition system for our programming language derives transitions of the form: $\mathcal{A} \longrightarrow \mathcal{A}'$. Usually, we will omit from notation the agents in $\mathcal{A}$ that do not affect nor are affected by the transition. Additionally, besides agent con gurations of the form $\langle i, S, \Lambda, \theta \rangle$, we will employ con gurations that are of the form $\langle i, a, \Lambda, \theta \rangle$ and $\langle i, \varphi, \Lambda, \theta \rangle$ in which the second element is not a statement but an action a or a result φ of executing an action.

De nition 5 *(Transition for the substitution of formula variables)*

$$\langle i, a, \Lambda, \theta \rangle \longrightarrow \langle i, a \cdot \theta, \Lambda, \theta \rangle$$

where $a \cdot \theta$ denotes the obvious extension of the operator $\cdot$ to atomic actions: e.g., $(X \leftarrow \varphi) \cdot \theta = X \leftarrow (\varphi \cdot \theta)$ and $\mathsf{query}(\varphi) \cdot \theta = \mathsf{query}(\varphi \cdot \theta)$.

In the transition rule all formula variables that occur in a formula in the action a are replaced by their associated value given by the function θ.

De nition 6 *(Transitions for assignment)*

$$\frac{\langle i, a, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda', \theta'\rangle}{\langle i, X \leftarrow a, \Lambda, \theta\rangle \longrightarrow \langle i, X \leftarrow \varphi, \Lambda', \theta'\rangle} \qquad \langle i, X \leftarrow \varphi, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda, \theta\{\varphi/X\}\rangle$$

The rst rule states how the transition for $X \leftarrow a$ can be derived from the transition for a, viz. the result φ of executing a is assigned to the formula variable X. The second rule shows that the assignment $X \leftarrow \varphi$ results in an update of θ such that it yields φ for the input X. We take the result of executing $X \leftarrow \varphi$ to be φ in order to model nested assignments as for instance $X' \leftarrow (X \leftarrow \varphi)$, which assigns φ to both X and X'.

De nition 7 *(Transition for query)*

$$\langle i, \mathsf{query}(?\mathbf{x}\psi), (\varphi, G), \theta\rangle \longrightarrow \langle i, \psi', (\varphi, G), \theta\rangle,$$

where we distinguish the following options for ψ':

$$\begin{array}{l}(1)\ \text{a substitution } \Delta, \text{ where } \Delta \in \mathcal{S}\\ (2)\ \text{a formula } \psi \oplus \Delta, \text{ where } \Delta \in \mathcal{S}\\ (3)\ \text{the formula } \bigwedge_{1\leq i\leq n}(\psi \oplus \Delta_i)\end{array}$$

and $\mathcal{S} = \{\Delta_1, \ldots, \Delta_n\}$ denotes the set of distinct ground uni ers for $(\varphi, ?\mathbf{x}\psi)$.

The choices above model different interpretations of querying the agent's belief state. The rst is one in which the result of the query is a ground uni er (provided that such a uni er exists). It is either the empty substitution denoting that the formula ψ is a classical consequence of the belief state, or a non-empty substitution de ning a value for the variables $\mathbf{x}$. For instance, the following transition is derivable:

$$\langle i, \mathsf{query}(?xP(x)), \Lambda, \theta\rangle \longrightarrow \langle i, x = a, \Lambda, \theta\rangle$$

where $\Lambda = (P(a) \wedge P(b), G)$. Note that this transition is not necessarily unique, as shown by the following transition that is derivable as well:

$$\langle i, \mathsf{query}(?xP(x)), \Lambda, \theta\rangle \longrightarrow \langle i, x = b, \Lambda, \theta\rangle$$

The second option is to apply the derived substitution to φ and yield this formula as the result of the query. In this case, the transition looks like:

$$\langle i, \mathsf{query}(?xP(x)), \Lambda, \theta\rangle \longrightarrow \{\langle i, P(a), \Lambda, \theta\rangle\}$$

Finally, there is the option to yield an exhaustive description comprised of all instances of the formula φ. The transition is then given by:

$$\langle i, \mathsf{query}(?xP(x)), \Lambda, \theta\rangle \longrightarrow \langle i, P(a) \wedge P(b), \Lambda, \theta\rangle$$

We refer to the rst option as the *substitution* interpretation, while the latter two are called the *derive-one* and the *derive-all* interpretations, respectively. Unless indicated otherwise, we will in this paper assume that the derive-one interpretation is in force.

De nition 8 *(Transition for update)*

$$\langle i, \mathsf{update}(\psi), (\varphi, G), \theta\rangle \longrightarrow \langle i, \mathit{true}, (\varphi \circ \psi, G), \theta\rangle$$

The transition for updates amounts to the incorporation of the formula ψ in the agent's belief state, where it is required that ψ does not contain any formula variables. We will not go into the details of the operator $\circ$, we assume it to be a parameter of the framework (see [12] for more details). We take the formula true as the result of the update.

De nition 9 *(Transition for sending a request)* If $\varphi_1 \vdash_{x=i_2} \psi$ and $m \in \mathbf{m}$ then:

$$\langle i_1, \mathsf{request}(\iota x.\psi, m(\varphi)), \Lambda_1, \theta_1\rangle, \langle i_2, \mathsf{accept}(\mathbf{m}), \Lambda_2, \theta_2\rangle \longrightarrow \\ \langle i_1, \mathsf{wait}(i_2), \Lambda_1, \theta_1\rangle, \langle i_2, \mathit{true}, \Lambda_2', \theta_2\rangle$$

where $\Lambda_1 = (\varphi_1, G_1)$ and $\Lambda_2 = (\varphi_2, G_2)$ and $\Lambda_2' = (\varphi_2, G_2 \cup \{m(\varphi) \Rightarrow i_1\})$.

In the classical notion of a rendezvous, the computation step of the agent i_2 that follows the rst synchronisation, would be the execution of the method invocation $m(\varphi)$. However, as mentioned before, a crucial characteristic of agent-oriented programming is that computations are performed relative to a mental state. Hence, the decision when to execute the method invocation should be based upon this state rather than that it is executed without any regard for the agent's current beliefs and goals. This is why the invocation $m(\varphi)$ is not executed immediately but added to the agent's goal state, along with the identi er i_1 representing the agent that the result of the invocation is to be sent back to. The construct $\mathsf{wait}(i_2)$ is used in the operational semantics to mark that the execution of this thread in i_1 is *blocked* until the result from i_2 has been received (see also de nition 12). We remark that in order to keep things simple here, we assume that i_1 cannot concurrently invoke (in another thread) any other method of i_2.

Finally, the construct $\iota x.\psi$ denotes a witness for x that satis es the formula ψ. The condition that $x = i_2$ is a ground uni er for $(\varphi_1, ?x\psi)$ requires this witness to be i_2. For instance, we have $\mathit{Agent}(i_2) \vdash_{x=i_2} ?x\,\mathit{Agent}(x)$, where $\mathit{Agent}(x)$ is used to express that x is an agent in the system (see also de nition 13).

De nition 10 *(Transition for method invocation)* If m is declared as $m(X) :- T$ then:

$$\langle i, m(\psi), \Lambda, \theta\rangle \longrightarrow \langle i, S \Rightarrow i, \Lambda, \theta\rangle,$$

where S equals $T[\psi/X]$, which denotes the body statement T of m in which the actual parameter ψ has been substituted for the formal parameter X.

Note that in comparison with standard concurrent programming, the parameter-passing mechanism in our framework is rather high-level, as formulae themselves constitute rst-class values with which methods can be invoked. The construct $S \Rightarrow i$ denotes that the result of executing the statement S should be sent back to the agent i.

De nition 11 *(Transition for handling goals)*
If m is declared as $m(X) :- T$ and $m \in \mathbf{m}$ then:

$$\langle i, \mathsf{handle}(\mathbf{m}), \Lambda, \theta\rangle \longrightarrow \langle i, S \Rightarrow j, \Lambda', \theta\rangle,$$

where $S = T[\psi/X]$ and $\Lambda = (\varphi, G \cup \{m(\psi) \Rightarrow j\})$ and $\Lambda' = (\varphi, G)$.

The presence of goal states yields the need for a mechanism of controlling the selection and execution of goals, which is a mechanism that is not present in the traditional concurrent language POOL. The above transition re ects a straightforward approach in which one of the invocations of a method m (if present) is taken from the goal state and identi ed to be subsequently executed.

De nition 12 *(Transitions for returning the result)*

$$\langle i_1, \mathsf{wait}(i_2), \Lambda_1, \theta_1\rangle, \langle i_2, \varphi \Rightarrow i_1, \Lambda_2, \theta_2\rangle \longrightarrow \langle i_1, \varphi, \Lambda_1, \theta_1\rangle, \langle i_2, \mathit{true}, \Lambda_2, \theta_2\rangle$$

$$\langle i, \varphi \Rightarrow i, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda, \theta\rangle$$

If a computation of a method invocation has terminated with a result φ then the second synchronisation of the rendezvous takes place, in which this result is communicated back to the agent i_1. The second rule deals with the case the invocation has been executed on behalf of the agent itself.

De nition 13 *(Transition for integration)*

$$\langle i, \mathsf{integrate}(C, S), \Lambda, \theta\rangle \longrightarrow \langle i, \mathit{Agent}(j), \Lambda, \theta\rangle,\ \langle j, S, (\mathit{true}, \{\}), \bot\rangle,$$

where $j \in \mathit{Ident}$ is a *fresh* agent identi er and $\bot$ denotes the totally unde ned function.

The transition rule de nes the integration of an instance of the agent class C. It shows how the agent system is expanded with a new agent j that starts its execution with the statement S where its initial mental state is de ned to be the empty one. The methods that the integrated agent can invoke are given by the methods of C.

The result of the integration is formulated by the information $\mathit{Agent}(j)$, which expresses that the agent with identi er j is part of the multi-agent system.

De nition 14 *(Transition rules for the construct $S \Rightarrow j$)*

$$\frac{\langle i, S, \Lambda, \theta\rangle \longrightarrow \langle i, S', \Lambda', \theta'\rangle}{\langle i, S \Rightarrow j, \Lambda, \theta\rangle \longrightarrow \langle i, S' \Rightarrow j, \Lambda', \theta'\rangle} \qquad \frac{\langle i, S, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda', \theta'\rangle}{\langle i, S \Rightarrow j, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi \Rightarrow j, \Lambda', \theta'\rangle}$$

The transition for the construct $S \Rightarrow j$ can be derived from the transition for the statement S. The rst transition rule deals with the case that S does not terminate after one computation step: S' denotes the part of S that still needs to be executed, while the second rule deals with the case that S terminates with a result φ.

De nition 15 *(Transition rule for sequential composition)*

$$\frac{\langle i, a, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda', \theta'\rangle}{\langle i, a; S, \Lambda, \theta\rangle \longrightarrow \langle i, S, \Lambda', \theta'\rangle}$$

The transition for the sequential composition of an action a and a statement S can be derived from the transition for the action a. Note that the result φ of executing a is simply ignored, because (possibly) it has already been processed in Λ' and θ'. For instance, we can derive the transition $\langle i, (X \leftarrow \varphi); S, \Lambda, \theta\rangle \longrightarrow \langle i, S, \Lambda, \theta\{\varphi/X\}\rangle$.

De nition 16 *(Transition rules for non-deterministic choice)*

$$\frac{\langle i, S_1, \Lambda, \theta\rangle \longrightarrow \langle i, S_1', \Lambda', \theta'\rangle}{\begin{array}{l}\langle i, S_1 + S_2, \Lambda, \theta\rangle \longrightarrow \langle i, S_1', \Lambda', \theta'\rangle \\ \langle i, S_2 + S_1, \Lambda, \theta\rangle \longrightarrow \langle i, S_1', \Lambda', \theta'\rangle\end{array}} \qquad \frac{\langle i, S_1, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda', \theta'\rangle}{\begin{array}{l}\langle i, S_1 + S_2, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda', \theta'\rangle \\ \langle i, S_2 + S_1, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda', \theta'\rangle\end{array}}$$

The transition for the non-deterministic choice $S + T$ between two statements can be derived from the transition of either S or T. The second rule deals with termination.

De nition 17 *(Transition rules for internal parallelism)*

$$\frac{\langle i, S_1, \Lambda, \theta\rangle \longrightarrow \langle i, S_1', \Lambda', \theta'\rangle}{\begin{array}{l}\langle i, S_1 \,\&\, S_2, \Lambda, \theta\rangle \longrightarrow \langle i, S_1' \,\&\, S_2, \Lambda', \theta'\rangle \\ \langle i, S_2 \,\&\, S_1, \Lambda, \theta\rangle \longrightarrow \langle i, S_2 \,\&\, S_1', \Lambda', \theta'\rangle\end{array}} \qquad \frac{\langle i, S_1, \Lambda, \theta\rangle \longrightarrow \langle i, \varphi, \Lambda', \theta'\rangle}{\begin{array}{l}\langle i, S_1 \,\&\, S_2, \Lambda, \theta\rangle \longrightarrow \langle i, S_2, \Lambda', \theta'\rangle \\ \langle i, S_2 \,\&\, S_1, \Lambda, \theta\rangle \longrightarrow \langle i, S_2, \Lambda', \theta'\rangle\end{array}}$$

The internal parallel composition $S \,\&\, T$ of two statements is modeled by means of an interleaving of the computation steps of S and T.

De nition 18 *(Transition rule for skip statement)*

$$\langle i, \mathsf{skip}, \Lambda, \theta\rangle \longrightarrow \langle i, \mathit{true}, \Lambda, \theta\rangle$$

The statement skip does not effect Λ and θ and yields the result true.

Example. Let $\mathcal{A}$ be an agent system (used in a library, for instance) consisting of a client agent i and additionally two server agents j and k. Consider the situation that the client i is looking for a biography on Elvis Presley and hence, asks the serving agent j for it.

The methods of the servers j and k are de ned as follows:

$$\begin{array}{lll} \mathit{answer} & :- & (\mathsf{accept}(\mathit{ask}) + \mathsf{handle}(\mathit{ask}));\, \mathit{answer} \\ \mathit{ask}(X) & :- & \mathsf{query}(X) + \mathsf{request}(\iota x.\mathit{Server}(x), \mathit{ask}(X)) \end{array}$$

The method answer is a recursive procedure in which in each round either a new ask-message is accepted or an ask-message is selected from the goal state and subsequently executed. Additionally, the method ask corresponds to a choice between querying the private information store (belief state) for an appropriate document and passing the question on to another server. We assume that the mental state Λ_2 of agent j is given by $(\neg P(a) \wedge \mathit{Server}(k), \{\})$ and the mental state Λ_3 of k is given by $(P(b), \{\})$, where we use the predicate P to denote that a document is a biography of Elvis Presley. Consider the initial con guration of the agent system $\mathcal{A}$:

$\langle i, X \leftarrow \mathsf{request}(j, \mathit{ask}(?xP(x)));\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, \mathit{answer}, \Lambda_2, \bot\rangle,$
$\langle k, \mathit{answer}, \Lambda_3, \bot\rangle,$

The agent i thus asks the agent j for the information $?xP(x)$ and subsequently will add the result of the query to its belief state. The following then constitutes a sequence of transitions (computation) starting with the above con guration.

$\longrightarrow$
$\langle i, X \leftarrow \mathsf{request}(j, ask(?xP(x)))\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{accept}(ask) + \mathsf{handle}(ask))\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, answer, \Lambda_2', \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$
where $\Lambda_2' = (\neg P(a) \wedge Server(k), \{ask(?xP(x)) \Rightarrow i\})$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{accept}(ask) + \mathsf{handle}(ask))\,; answer, \Lambda_2', \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{query}(?xP(x)) + \mathsf{request}(\iota x.Server(x), ask(?xP(x))) \Rightarrow i)\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{query}(?xP(x)) + \mathsf{request}(\iota x.Server(x), ask(?xP(x))) \Rightarrow i)\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, (\mathsf{accept}(ask) + \mathsf{handle}(ask))\,; answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{wait}(k) \Rightarrow i)\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, answer, \Lambda_3', \bot\rangle \longrightarrow$
where $\Lambda_3' = (P(b), \{ask(?xP(x)) \Rightarrow j\})$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{wait}(k) \Rightarrow i)\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, (\mathsf{accept}(ask) + \mathsf{handle}(ask))\,; answer, \Lambda_3', \bot\rangle \longrightarrow$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{wait}(k) \Rightarrow i)\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, (\mathsf{query}(?xP(x)) \Rightarrow j)\,; answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (\mathsf{wait}(k) \Rightarrow i)\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, (P(b) \Rightarrow j)\,; answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, X \leftarrow \mathsf{wait}(j)\,;\mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, (P(b) \Rightarrow i)\,; answer, \Lambda_2, \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, X \leftarrow P(b); \mathsf{update}(X), \Lambda_1, \bot\rangle,$
$\langle j, answer, \Lambda_2, \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, \mathsf{update}(X), \Lambda_1, \{P(b)/X\}\rangle,$
$\langle j, answer, \Lambda_2, \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$

$\langle i, true, (P(b), \{\}), \{P(b)/X\}\rangle,$
$\langle j, answer, \Lambda_2, \bot\rangle,$
$\langle k, answer, \Lambda_3, \bot\rangle \longrightarrow$

4 Related Work

The research on multi-agent systems has resulted in the development of various agent communication languages, none of which however has yet been assigned a satisfactory formal semantics. And it is this lack of a clear semantics that in our opinion constitutes one of the major hindrances for an agent communication language to become widely accepted. In this section, we discuss the relation of our framework with two of these languages, viz. the agent communication languages KQML and FIPA-ACL.

KQML. The Knowledge Query and Manipulation Language (KQML) provides a language for the exchange of knowledge and information in multi-agent systems (cf. [10]). It de nes the format of a collection of messages that are exchanged between communicating agents. The main constituents of a KQML message are a *performative* that indicates the purpose of the message, its *content* expressed in some representation language and nally , the *identities* of the sender and the receiver.

The semantics of a KQML message are given by the following three ingredients: (1) a *precondition* on the mental states of the sender and receiver before the communication of the message, (2) a postcondition that should hold after the communication and (3) a *completion* condition that should hold after the conversation of which this message was a constituent, has terminated. The language in which these conditions are expressed consists of logical combinations of the following v e operators: $bel(i, \varphi)$ denoting that φ is in the knowledge base of i and $know(i, \varphi)$, $want(i, \varphi)$ and $intend(i, \varphi)$, expressing that i knows φ, wants φ and is committed to φ, respectively. Finally, $process(i, m)$ denotes that the message m will be processed by the agent i.

An important feature of the framework is that the communication of a message is not an action that occurs in isolation, but it takes place in the context of a conversation comprised of a sequence of communications. In this wider context, the semantics of the KQML messages can be thought of de ning correct conversations. That is, the precondition of a message determines the collection of messages that are allowed to precede the message, while the postcondition lays down the messages that are permitted to succeed the message. Additionally, in case the completion condition is a logical consequence of the postcondition, the conversation can be identi ed to have successfully terminated.

For instance, the following sequence of messages constitutes a typical conversation:

$$\mathbf{advertise}(i, j, \mathbf{ask\text{-}if}(j, i, \varphi)),\ \mathbf{ask\text{-}if}(j, i, \varphi) \text{ and } \mathbf{tell}(i, j, \varphi)$$

Let us examine the constituents of this conversation in more detail. First, the pre-, post- and completion conditions for a message $\mathbf{advertise}(i, j, m)$ are as follows, where in this case m is given by $\mathbf{ask\text{-}if}(j, i, \varphi)$:

$$\begin{array}{ll} (1) & \mathit{intend}(i, \mathit{process}(i, m)) \\ (2) & \mathit{know}(i, \mathit{know}(j, \mathit{intend}(i, \mathit{process}(i, m))))\wedge \\ & \quad \mathit{know}(j, \mathit{intend}(i, \mathit{process}(i, m))) \\ (3) & \mathit{know}(j, \mathit{intend}(i, \mathit{process}(i, m))) \end{array}$$

The intuitive meaning of the message $\mathbf{advertise}(i, j, m)$ amounts to letting the agent j know that i has the intention to process the message m. Additionally, the fact that the completion condition coincides with the postcondition, implies that this message constitutes a conversation by itself.

In our framework, we have a slightly different mechanism: there is the primitive of the form $\mathsf{accept}(\mathbf{m})$, which re ects the agent's intention to accept a message that is in the collection $\mathbf{m}$. It has the advantage above the KQML message $\mathbf{advertise}(i, j, m)$ that it is very e xible as it speci es the messages that will *currently* be accepted; a subsequent occurrence of the primitive in the agent program might specify a sub-, super- or even a completely disjoint set.

Secondly, the conditions for a message $\mathbf{ask\text{-}if}(j, i, \varphi)$ are as follows:

$$\begin{array}{ll} (1) & \bigvee_{\psi\in\Gamma}(\mathit{want}(j, \mathit{know}(j, \psi)))\wedge \\ & \quad \mathit{know}(j, \mathit{intend}(i, \mathit{process}(i, m)))\wedge \\ & \quad \mathit{intend}(i, \mathit{process}(i, m)) \\ (2) & \bigvee_{\psi\in\Gamma}(\mathit{intend}(j, \mathit{know}(j, \psi)))\wedge \\ & \quad \mathit{know}(i, \bigvee_{\psi\in\Gamma}(\mathit{want}(j, \mathit{know}(j, \psi)))) \\ (3) & \bigvee_{\psi\in\Gamma}(\mathit{know}(j, \psi)) \end{array}$$

where m is given by $\mathbf{ask\text{-}if}(j, i, \varphi)$ and Γ equals $\{\mathit{bel}(i, \varphi), \mathit{bel}(i, \neg\varphi), \neg\mathit{bel}(i, \varphi)\}$.

The intuitive meaning of the message $\mathbf{ask\text{-}if}(j, i, \varphi)$ amounts to letting the agent i know that j wants to know whether i believes φ, believes $\neg\varphi$ or does not believe φ. The second and third conjunct of the precondition re ect the requirement that the message is to be preceded by an $\mathbf{advertise}(i, j, m)$ message. Additionally, the completion condition indicates that the conversation of which this message is a constituent, has not successfully terminated until the agent j knows one of the formulae in k. As KQML abstracts away from the content language of messages, it is not clear to us why the formula $\mathit{bel}(i, \neg\varphi)$ is an element of the set Γ, as it refers to an operator $\neg$ in the content language (which in principle, does not need to be present at all). In particular, if the agent j wants to know whether i believes $\neg\varphi$ it can use the message $\mathbf{ask\text{-}if}(j, i, \neg\varphi)$.

Thirdly, the conditions for the message $\mathbf{tell}(i, j, \varphi)$ are as follows:

$$\begin{array}{ll} (1) & \mathit{bel}(i, \varphi) \wedge \mathit{know}(i, \mathit{want}(j, \mathit{know}(j, \mathit{bel}(i, \varphi))))\wedge \\ & \quad \mathit{intend}(j, \mathit{know}(j, \mathit{bel}(i, \varphi))) \\ (2) & \mathit{know}(i, \mathit{know}(j, \mathit{bel}(i, \varphi))) \wedge \mathit{know}(j, \mathit{bel}(i, \varphi)) \\ (3) & \mathit{know}(j, \mathit{bel}(i, \varphi)) \end{array}$$

The intuitive meaning of the message is to let the agent j know that i believes the formula φ to be true. The first conjunct of the precondition states that i should indeed believe the formula φ, where the second and third conjunct should indicate that the message is to be preceded by a message of the form $\mathbf{ask\text{-}if}(j, i, \varphi)$. Here we see why it is so important to have a formal framework to give semantics to a language: mistakes are easily made. The postcondition $\bigvee_{\psi \in \Gamma}(intend(j, know(j, \psi)))$ where $\Gamma = \{bel(i, \varphi), bel(i, \neg\varphi), \neg bel(i, \varphi)\}$ of the *ask-if* message, does not entail the precondition $intend(j, know(j, bel(i, \varphi)))$ of the *tell* message. Hence, with this semantics the reception of an $\mathbf{ask\text{-}if}(j, i, \varphi)$ message is not sufficient to be able to send a $\mathbf{tell}(i, j, \varphi)$ message. Additionally, the nesting of knowledge operators seems to be quite arbitrary: it is unclear why the nesting of knowledge operators in the postcondition is restricted to depth two, and for instance does not include $know(j, know(i, know(j, bel(i, \varphi))))$.

Our concern with defining the semantics of communication in the manner as employed for KQML, is that it still does not yield an exact meaning of agent communication. The KQML semantics is defined in terms of operators for which the semantics themselves remain undefined. For instance, the exact difference between the operators *want* and *intend* remains unclear.

Moreover, in contrast to the approach employed in this paper, there is a gap between the syntactic structure of the language and its semantics, which is due to the use of rather high-level operators. In our framework however, the operational semantics closely follow the syntactic structure of the language, which opens up the possibility of developing an interpreter for the language on the basis of this operational description. This is due to the fact that the configurations of an agent system can be seen as an abstract model of a machine, where its transitions define the actions that it can perform. In fact this machine would act as an interpreter for the language.

Thirdly, we believe that the KQML semantics impose too strong requirements on the agents with respect to their reactions towards incoming messages. We believe that these reactions are agent-dependent and hence should not be part of the semantics of messages. For instance, in our framework, consider three agents each having one the following definitions of the method $ask(X)$:

$$\begin{array}{lll} ask(X) & :- & \mathsf{query}(X) + \mathsf{query}(\neg X) \\ ask(X) & :- & X' \leftarrow \mathsf{query}(X); X'' \leftarrow \neg X' \\ ask(X) & :- & \mathsf{request}(\iota x.Agent(x), ask(X)) \end{array}$$

The first agent tests its belief state and checks whether it entails the formula X or its negation. This corresponds to the reaction imposed by the KQML semantics for the $\mathbf{ask\text{-}if}(X)$ message. On the other hand, the second agent checks whether its belief state entails X and subsequently delivers the negation of what it believes to hold (note that the result of the sequential composition of two actions is given by the result of the second action). In KQML this reaction is simply ruled out. However, in our opinion this reaction is not a result of the semantics of the message but of the characteristics of the receiving agent. We believe that any reaction should be allowed, as for instance shown by the third agent, which simply passes the message along to another agent and delivers the result that it receives from this agent.

FIPA-ACL. Besides the language KQML there is a second proposal for a standard agent communication language, which is developed by the organisation FIPA. This language, which we will refer to as the FIPA-ACL, also prescribes the syntax and semantics of a collection of message types (cf. [11]). The format of these messages is almost equal to that of KQML, while their semantics are given by means of (1) a *precondition* on the mental state of the sender that should hold prior to the dispatch of the message and (2) the expected *effect* of the message.

There are four primitive messages; viz. $\langle i, \mathsf{inform}(j, \varphi)\rangle$ and $\langle i, \mathsf{confirm}(j, \varphi)\rangle$ in which the agent i tells the agent j that it believes the formula φ to hold and a message $\langle i, \mathsf{disconfirm}(j, \varphi)\rangle$ in which the agent i tells the agent j that it believes the negation of φ to hold. The difference between the inform and con rm message is that the former is employed in case agent i has no beliefs about the beliefs of j concerning φ, i.e., it does not believe that j believes φ or its negation or is uncertain about φ or its negation, whereas the latter is used in case i believes that j is uncertain about φ. Thirdly, the discon rm message is used for situations in which i believes the negation of φ and additionally that j believes φ or is uncertain about φ. The expected effect of informing φ, con rming φ and discon rming $\neg\varphi$ is the same: the receiver believes φ.

The fourth primitive message is of the form $\langle i, \mathsf{request}(j, a)\rangle$ in which i requests the agent j to perform the action a. The condition for this message is that i believes that the agent j will be the only agent performing a and that it does not believe that j has already an intention of doing a. Additionally, the part of the precondition of a that concerns the mental attitudes of i should additionally hold.

All other messages are de ned in terms of these primitives together with the operators ; for sequential composition and | for non-deterministic choice. An example of a composite message is $\langle i, \mathsf{query\text{-}if}(j, \varphi)\rangle$, which is an abbreviation for the construct $\langle i, \mathsf{request}(j, \langle j, \mathsf{inform}(i, \varphi)\rangle \mid \langle j, \mathsf{inform}(i, \neg\varphi)\rangle)\rangle$. Analogous to KQML, there remains a gap between the syntax of the communication language and the semantic description given in terms of high-level modal operators as those for (nested) belief and uncertainty. We think however that the operational model outlined in this paper, could act as a rst step in the development of an operational description of the FIPA-ACL.

5 Future Research

In this paper, we have outlined a basic programming language for systems of communicating agents that interact with each other via a rendezvous communication scheme. In subsequent research, we will study the extension of the framework with a notion of agent expertise in the form of a vocabulary or signature together with the incorporation of object-oriented features as subtyping and inheritance. Inheritance would then not be restricted to the inheritance of methods but could also involve the inheritance of expertise. Another issue is the study in what way the current framework could be used to develop an operational semantic model for existing agent communication languages.

References

1. P.H.M. America, J. de Bakker, J.N. Kok, and J. Rutten. Operational semantics of a parallel object-oriented language. In *Conference Record of the 13th Annual ACM Symposium on Principles of Programming Languages*, pages 194–208, St. Petersburg Beach, Florida, 1986.
2. G.R. Andrews. *Concurrent Programming, Principles and Practice*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.
3. A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2:39–59, 1984.
4. P. Cohen and H. Levesque. Communicative actions for arti cial agents. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.
5. M. Colombetti. Semantic, normative and practical aspects of agent communication. *In this volume*.
6. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. A language for modular information-passing agents. In K. R. Apt, editor, *CWI Quarterly, Special issue on Constraint Programming*, volume 11, pages 273–297. CWI, Amsterdam, 1998.
7. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Systems of communicating agents. In Henri Prade, editor, *Proceedings of the 13th biennial European Conference on Arti cial Intelligence (ECAI-98)*, pages 293–297, Brighton, UK, 1998. John Wiley & Sons, Ltd.
8. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Information-passing and belief revision in multi-agent systems. In J. P. M. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V — Proceedings of 5th International Workshop on Agent Theories, Architectures, and Languages (ATAL'98)*, volume 1555 of *Lecture Notes in Arti cial Intelligence*, pages 29–45. Springer-Verlag, Heidelberg, 1999.
9. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Open multi-agent systems: Agent communication and integration. In *Intelligent Agents VI, Proceedings of 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'99)*, Lecture Notes in Arti cial Intelligence. Springer-Verlag, Heidelberg, 2000.
10. T. Finin, D. McKay, R. Fritzson, and R. McEntire. KQML: An Information and Knowledge Exchange Protocol. In Kazuhiro Fuchi and Toshio Yokoi, editors, *Knowledge Building and Knowledge Sharing*. Ohmsha and IOS Press, 1994.
11. Foundation For Intelligent Physical Agents. Fipa'97 speci cation part 2 – agent communication language. Version dated 10th October 1997.
12. P. Gärdenfors. *Knowledge in ux: Modelling the dynamics of epistemic states*. Bradford books, MIT, Cambridge, 1988.
13. K.V. Hindriks, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. A formal semantics for an abstract agent programming language. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Proceedings of Fourth International Workshop on Agent Theories, Architectures and Languages (ATAL'97)*, volume 1365 of *LNAI*, pages 215–229. Springer-Verlag, 1998.
14. G. Plotkin. A structured approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
15. B. Thomsen. A calculus of higher order communicating systems. In *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 143–153, 1989.
16. M. Wooldridge and N. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.