# A Model for Semantics and Pragmatics in Communication

Christoph Brandt
*Université du Luxembourg*
*SECAN-Lab, Campus Kirchberg*
*6, rue Richard Coudenhove-Kalergi*
*L-1359 Luxembourg-Kirchberg, EU*
christoph.brandt@uni.lu
Marcus Kracht
*Fakultät LiLi*
*Postfach 10 01 31*
*D-33501 Bielefeld*
christoph.brandt@gmx.net
marcus.kracht@uni-bielefeld.de

March 25, 2009

**Abstract**

We define a formal model of situated agents with channels for communication that handles interaction between humans and machines alike. The model develops a formal semantics that allows for agents to reason both about their own situation as well as about the situation of other agents.

# 1 Introduction

This paper describes a comprehensive scheme to define and model communication between independent agents, which can be either humans or machines. Communication proceeds through various channels, between independent agents. Communication is regulated through a set of rules protocol. Although in principle an integrative approach, allowing both for actions and verbal exchanges, the primary focus here is on the communicative aspect. We shall look in particular at the way excanges are structured.

One of the main features of the present model are the following.

① Agents act independently; moreover, they each have their own private model corresponding to the data directly accessible to them. This would be, for example, a bank having access to various bank accounts but not to how much cash their clients have, while clients have access to their own cash, and to their bank account (via the bank). Whether or not the models are seen as part of of a big model need not be a concern.

② Agents exchange messages through various channels. The channels can be physical (air, wires) or logical (ports). Channels are a way to select the group of addressees. Sensitive information is protected either by encryption or by using a particular channel.

③ Protocols regulate the sequence of messages and actions. Some transactions require a cascade of both (see [Weigand and Van Den Heuvel, 1998].) Protocols must be obeyed by each agent.

④ Exchanges between agents allow each of them (and eavesdroppers) to gain access to each others models. The present paper develops a framework to study what knowledge the participants gain.

There are also certain aspects that we shall not deal with. We shall not engage in a particular speech act classification, for example. Moreover, utterances by definition have exactly one force, which is given by the protocol (see also [Kimbrough and Thornburg, 1989]). Thus we assume that what has the form of a question also is a question if that is specified by the protocol. Thus indirect speech acts are left out of consideration here. This allows to ignore the problem of speech

act recognition, a thorny issue. Second, we shall not deal with speech acts that involve anything but factual talk. Thus, promises and obligations, for example, are left out of consideration. Third, although the exchange proceeds using natural language, underlyingly the meaning of the words is completely fixed. Thus what we have effectively is a version of a FLBC (formal language for business communication, [Kimbrough and Thornburg, 1989]). ), though it is on the surface a natural language.

Though each of these areas are interesting in their own right, it would complicate the matter beyond need. For there are already numerous useful applications for the present approach. We shall explore below one specific application in detail. The scenario is that of several banks and clients. Clients can request banks to perform various actions, such that opening and closing accounts, paying and transferring money. Money transfer is perhaps the most interesting type of action. For it involves not only several actions but also a chain of requests and acknowledgments.

# 2 Channels, Networks and Messages

We begin by outlining the formal aspects of networks and the organisation of communication in terms of a protocol. The basic framework is this. Agents can communicate through channels. The communication is between different groups.

A **communication network** is a pair $\langle A, \mathcal{C}\rangle$, where $A$ is a set, the set of **agents**, and $\mathcal{C} \subseteq \wp(A)$ a set of **connections** or **channels**. ($\wp(A)$ is the powerset of $A$.) The sets describe the range of physical media through which messages can be sent. These can be wires, or air to send radio waves or simply sound. We say that $a$ and $b$ are connected if there is a $C \in \mathcal{C}$ that contains both $a$ and $b$. If a connection exists between $a$ and $b$, either $a$ may send a message to $b$ or $b$ may send a message to $a$. Indeed, it is therefore assumed that the channel always works both ways. Furthermore, it is assumed that if $a$ uses $C$ to communicate to $b$ then the message reaches all members of $C$ indiscriminately, as in the case of radio signals. One should distinguish this physical notion of a channel from the notion of a logical channel, often used in computer science. In the latter, message flow may be unidirectional. Unidirectional messaging can be seen instead as a consequence of machine design (one machine is able to send, the other to receive, but not vice versa) and not as a property of the (physical) channel. In actual fact,

the logical channel can be replicated here in the assignment of languages to sender and addressee.

For this reason any message sent out by an agent will have to contain information as to who sent it and to whom it is adressed. In practice the address is not always a single agent but will specify a set that is more or less explicitly described (think of an IP address). Thus, in addition to the network we also need a means to identify addressees. In natural language this is either left implicit (through gaze contact, for example) or given in the form of formal address (in an opening of a letter), or in form of an apposition to the personal pronoun (/`you, Martha, will perform the next solo`/). In an email, the recipient machine is specified in the header. Whatever the means, these methods rely on special devices to single out a group of agents.

**Definition 1** *A **communication structure** is a quadruple $\langle A, \mathcal{C}, \mathcal{P}, g\rangle$, where $\langle A, \mathcal{C}\rangle$ is a network, $\mathcal{P}$ a finite set, the set of predicates, and $g : \mathcal{P} \to \wp(A)$, a function, called a **group intepretation**.*

Given a communication structure, we can form descriptions of agents. We shall confine ourselves to a simple boolean structure. An **agent selector** is

- a member of $\mathcal{P}$,
- of the form $\neg e$, where $e$ is an agent selector,
- of the form $(e \wedge e')$ or $(e \vee e')$, where $e$ and $e'$ are agent selectors.

$a$ **satisfies** $e$, in symbols $a \vDash e$, if

- $e \in L$ and $a \in g(e)$,
- $e = \neg e'$, and $a \nvDash e'$,
- $e = (e' \wedge e'')$ and $a \vDash e'$ as well as $a \vDash e''$,
- $e = (e' \vee e'')$ and $a \vDash e'$ or $a \vDash e''$ (or both).

We write $g(e)$ is the set of all $a$ that satisfy $e$. If $e \in \mathcal{P}$ this coincides with the original function $g$.

We say that a **message** is a triple

(1) $m = \langle a, e, s \rangle$

where $a$ is an agent, $e$ an agent selector and $s$ is the **sentence** or **message body** (see below). $a$ is the sender (or speaker) and $e$ the address. From an address $e$ the set of addressees is calculated as $g(e)$.

An **utterance** (or *messaging event*) is a triple

(2) $u = \langle t, C, m \rangle$

where $t$ is a timestamp, $C$ a channel, and $m$ a message. We say that $u$ is **earlier** than $u' = \langle t', C', m' \rangle$ if $t < t'$; in this case we write $u < u'$. We say that $u$ and $u'$ are **concurrent**, in symbols $u \circ u'$, if $t = t'$. $u$ is **well addressed** if $m$ has the form $\langle a, e, p \rangle$ and $a \in C$ as well as $g(e) \subseteq C$. This is to say, all addressees must be physically reachable via $C$ from $a$, which requires that $C$ contains $a$. In practice, people allow the channel to implicitly restrict the set of addressees; rather than saying whom they address, they simply let everyone in $C$ be the addressee, and only add as much information in $e$ to narrow down that set to satisfaction. In these circumstances, the intended set of recipients is $C \cap g(e)$. We shall opt here for the strictier version for simplicity.

In terms of the previous we can now say the following. $b \neq a$ is a **recipient** of the message if $b \in C$. $b$ is **ratified** $b$ satisfies $e$ ([McCawley, 1999]). Notice that none of these concepts are applied to $a$. This gives us a fourfold classification. There are ratified recipients, non-ratified recipients (eg bystanders), there are ratified non-recipients and non-ratified non-recipients. (See [Goffmann, 1976] for a finer classification.) As we have said, in natural conversation there is a possibility that $b$ is ratified but not a recipient. In this case $a$ made an error of issuing a message for a recipient that is unable to receive the message and the event is unsuccessful. The notion of a ratified recipient is important for the following reason. If $a$ sends out a command to do something, then the obligation to perform the action is only on the ratified recipients and not on any eavesdropper. If $a$ promises something to $d$ then the promise is valid for any ratified recipient but not for a non-ratified one.

We ignore logical complications of the following sort. English "you" means a singular addressee or a plural addressee. The difference comes out in the promise *I will give you 5 Euros*. We can understand "you" here in the singular, in which case

any ratified recipient will receive 5 Euros, or we can understand it collectively (in the plural that is), in which case the entire group will receive 5 Euros. In what is to follow the second possibility is ignored. Thus, a promise is issued to any ratified recipient indidually, a question is asked to any ratified recipient individually, and so on. This is for the reason that in return the recipients must answer as a group. However, the issuer of a message is a single agent. There is no equivalent of the English expression "we". This is not imlement because issuing a promise, say, will require that not only the sender of the message but the enture group knows about it, and so the sender must make the group aware of the promise.

For simplicity we assume that for each agent $a$ there is a selector $e$ such that only $a$ satisfies $e$. In this case the communication structure is called **full**. If that is the case, we select for each $a$ and expression $e_a$ that only $a$ satisfies. A **simple message** has the form

(3) $$m = \langle a, e_b, p \rangle$$

for some $b$. This means that the address contains a single agent only, namely $b$.

# 3 Dialogues and Protocols

A **dialogue** is a sequence $\Delta = \langle u_i : i \in n \rangle$ of distinct message events such that if $i < j$ then it is not the case that $u_j < u_i$. In other words, the dialogue is chronologically ordered. (We could have said it is a set of message events and leave the ordering implicit.) Notice that the same message can occur many times in a dialogue. Notice that each agent may use its own clock. A **protocol** is a set of conditions on dialogues (see [Fernandez and Endriss, 2007]). This usage differs from usage in computer science where protocol is an observational record. Here it is a set of normative statements. These conditions can be of various sorts and for various reasons. There are conditions that are merely ensuring coherence; there are others that ensure security.

Communication consists in a structured dialogue. The structure is defined by certain ways in which messages must be sent around the network. Recall that for example a question needs an answer. The pair of question and answer is also called an **adjacency pair**. If $a$ asks $b$ for something, it is a violation of protocol not to answer back. $b$ may either answer the question or state that he cannot do so.

We shall say, however, that not only questions but any type of message (command, promise, statement) must be matched by another message that goes back to sender. This means that the control structure for a dialogue between two people uses a pushdown automaton (see [Fernandez and Endriss, 2007]). Here however we prefer not to talk about control structure. Thus message types are distinguished by what kind of answer they require. In electronic communication it is made clear what message is a reply to what other message. We can implement this as follows. A **turn structure** is a set $\tau$ of pairs $\{u, u'\}$ of utterance events. Given two utterance events it is clear that the later utterance is in reaction to the previous; for example, if $u$ occurs later than $u'$ then $u$ cannot be a question, instead must be an answer to $u'$. These properties are regulated by the protocol. A **structured dialogue** is a pair $\langle \Delta, \tau \rangle$ such that $\Delta$ is a dialogue and $\tau \subseteq \wp(\Delta)$ is a turn structure.

Turn structures have to satisfy various restrictions, some of which will be spelled out below.

In real situations, the turn structure is not always explicitly given, mostly only partially. For example, in email communication each message has an identifier and that identifier is used for reference when using the "reply" function. This then establishes a turn. These turns can be used to create threads automatically. There answering an email by using the "reply" option means that the email is flagged as an answer to a specific email message. Alternatively, we can answer an email by sending a *new* message and refer verbally to the previous message, but that way the thread is no longer automatically recoverable. Notice that message can in principle answer several previous messages; we ignore that point here. Rather than taking the existence of a full turn connection for granted we allow for the turn connection to be given only partially. In a real life dialogue the turns must be reconstructed from the actual dialogue.

**Definition 2** *A **protocol** is a property* $\Pi$ *of structured dialogues. A structured dialogue* $\langle \Delta, \tau \rangle$ *is said to **fulfill*** $\Pi$ *if there is a* $\tau' \supseteq \tau$ *such that* $\langle \Delta, \tau' \rangle$ *satisfies* $\Pi$.

Thus, while satisfaction of the protocol means having the property $\Pi$, fulfillment merely requires satisfaction of some extension. It is of course desired that there be at most one $\tau'$ such that $\tau' \supseteq \tau$ and $\langle \Delta, \tau' \rangle$ satisfies $\Pi$. We call such protocols **stable**. Another property we want to have of a protocol is that it is **locally testable**. This means that a dialogue satisfies $\Pi$ if every agent agrees that it does. This is nontrivial. If $a$ asks a question into the network to a set of addressees then $a$ can

check whether every addressee answered it (we assume no data loss here). However, none of the other participants may be able to check that property (suppose, for example, that each addressee sends his answer only to $a$). Thus, there is no single agent that can check the satisfaction of the protocol. On the other hand, it makes no sense to require properties of the dialogue that no one can effectively check, although that is in principle possible. (An example: suppose that the protocol requires that $a$ may ask $b$ a question only if $c$ does not know the answer. Assume furthermore that $a$ cannot come to know what $c$ knows. Then it is not possible for $a$ to know whether he is permitted to ask $b$.)

In the rest of this section we shall look at the interaction between mood types and protocol. A **sentence** is a pair $\langle \tau, p \rangle$, where $\tau$ is a **mood type** and $p$ an open proposition, the **content**. Here is the list of mood types that we shall use in this paper:

① **Statement**, abbreviated $\vdash$.

② **Yes-No-Question**, abbreviated ?.

③ **Command**, abbreviated !.

④ **Promise**, abbreviated ☞.

⑤ **Acknowledgment**, abbreviated ☑.

In ordinary language, a command is accompanied either by the execution of the order or by an acknowledgment. If John asks Peter to clean the bathroom, Peter may respond /`Yes.`/. This is both an acknowledgment of receipt of the command and a commitment that he will perform the requested action. So, we may say that the protocol contains a condition that for every command $u = \langle t, C, \langle a, e_b, \langle !, p \rangle \rangle \rangle$ there is an acknowledgment $u' = \langle t', C', \langle b, e_a, \langle ☑, \texttt{Yes} \rangle \rangle \rangle$ such that $\{u, u'\} \in \tau$ and $t' > t$. (The fact that $b$ has to perform an action is left out of consideration here.) Note that there are several types of acknowledgments. There is /`OK`/ or /`Yes.`/, which signals receipt and compliance, but there are others, including those that signal receipt and non-compliance or failure. If $a$ orders $b$ something, then $b$ may either comply, or refuse compliance, or tell $a$ than he cannot comply, and so on.

A very important property of dialogues is the non-nestingness, which we shall assume throughout.

**Definition 3** *A structured dialogue* $\langle \Delta, \tau \rangle$ *is said to be* ***nesting*** *if there are* $\{u, u'\}, \{v, v'\} \in \tau$ *such that* $u < v < u' < v'$.

It is known not to hold universally for human dialogues (see [Asher, 1998] and [Ginzburg, 2008]).

As there can be several addressees of a single message we need to look more closely at the way a protocol handles this situation. Suppose $a$ sends out a question to several people. Then it is not necessary for $a$ to send the same question to each and every addressee; nevertheless, every addressee has to answer that question. The speech act is distributive: it distributes to all addressees. This is not always the case. If I order a group of people to answer a rather complicated question then I may have asked the entire group as a whole, and then only one person is asked to reply for the whole group. This can be modeled only if we allow for groups of speakers, which we did not. Instead, we assume here total distributivity. We shall indicate a few consequences of this property.

In real situations, it is not the case that the protocol is used to check the well-formedness of a dialogue; rather, the dialogue is structured with the help of protocol rules. The protocol defines what can constitute a reply to an utterance, and the actual reply is taken to be the first such utterance. On the basis of the protocol the utterance-reply pairs are defined and next it is checked that the utterance-reply pairs never cross for each pair of agents. For example: $a$ asks $b$ whether he, $a$, has to go shopping. Instead of issuing a statement (which would then constitute a reply to that question), $b$ responds with a question, asking $a$ how late it is. $a$ answers that question upon which $b$ says: /`Yes.`/. This last statement can, and therefore must be, seen as an answer to the first question of $a$s. The dialogue is transparent for both $a$ and $b$ since the protocol is something they both share.

Compliance with a protocol is a formal matter. It is not defined via content. Thus, if $b$ does not answer $a$'s question truthfully, say, if it is not the case that $a$ has to go shopping, then $b$ has kept to the protocol but has not answered according to fact. If additionally $b$ knows that $a$ does not have to go shopping then $b$ has certainly lied. Yet he *has* given an answer to the question so again he has kept the protocol. In certain situations this is even to be expected. In a court hearing we expect the defendant to lie. The protocol only enforces that the defendant answers the questions put forward formally.

It is perhaps worthwhile to illustrate the necessity of protocols. Suppose that

$a$ asks $b$ two questions in a row: whether he, $a$, has to go shopping, and whether $b$ needs to go to the bank. If $b$ simply answers /`Yes.`/ then we are clueless which of the questions has been answered. For in and of itself the answer /`Yes.`/ does not provide enough of a proposition. Formally, we represent this as

(4) $\langle b, e_a, \langle \vdash, \texttt{true} \rangle \rangle$

So, $b$ simply says: true. Only because the message is read as a reply to a question this statement can be interpreted appropriately. We shall see how that kind of dialogue interpretation can be achieved. Let us simply note here that protocol failures lead to a different kind of violation than lying. If $b$ has lied then the proposition he has stated is known to be wrong to $b$; if he has not kept the protocol we do not know what he actually said.

Alternatively, we may read $b$'s message as

(5) $\langle b, e_a, \langle \vdash, p \rangle \rangle$

where $p$ is a variable that needs to be instantiated. Thus, $b$ is herewith claiming the truth of some proposition that needs to be established. In ordinary conversation that may be a better way to proceed: for $b$ may enclose some hints in his message that allow to settle the question which of $a$'s questions $b$ intends this message to be a reply of (eg by saying /`Yes, I have to.`/ or /`Yes, you have to.`/). In that case, one may dispense with the nesting rule. This is how human dialogue proceeds ([Fernandez and Endriss, 2007]). However, for present purposes it is better not to allow for this possibility.

# 4 Inside the Agents

With each agent we associate a language $L_a$ and model structure $\mathfrak{M}_a$ for the language $L_a$. There are no conditions on the nature of this structure. We may have, for example, a language that allows expressions of the form /`The temperature is 5 degrees.`/, and the model structure is simply a number. Or the language is a first-order predicate language without functions and the model structure is a relational database. It is important that the truth of an $L_a$ expression in $\mathfrak{M}_a$ be defined. If $\varphi$ is an $L_a$ expression we write $\mathfrak{M}_a \models \varphi$ to say that $\varphi$ is true in $\mathfrak{M}_a$. If $\mathfrak{M}_a \models \varphi$ we also say that $a$ **knows** that $\varphi$. For purposes of the present discussion,

a model structure is a function from $L_a$ to $\{\top, \bot\}$. It is not defined anywhere else. Thus, effectively, the model structure decides for every string whether or not it represents a true proposition. $\mathfrak{M}_a$ interprets a string $\varphi$ if it is defined on $\varphi$.

Several points need to be made about the languages and the models. The models correspond to the facts directly observable by the agent (similar the observables of [Alechina and Logan, 2009]). Of course, the models can be seen as part of a bigger reality. Let us imagine a general $g$ sending out two spies $s$ and $s'$ into two different enemy camps. Imagine that the two spies cannot contact each other but they can talk to the general. Then we have the network $\langle\{g, s, s'\}, \{\{g, s\}, \{g, s'\}\}\rangle$. The general may know that the camp of $s$ and that of $s'$ are within sight of each other at some distance. He may now test his spies asking each whether they see a camp nearby. Suppose each of them reports they can see just one, and that $s$ says that the camp is 5 km away while $s'$ reports it is 15 km away. This information is incompatible. For the the models of each of the spies are part of one bigger model, and distance is symmetric. Also, the camp about which $s$ reports is the camp that $s'$ sits in, and conversely the camp that $s'$ reports seeing is the one that $s$ is in. Thus, the language $s$ and $s'$ use to talk is mapped to the same underlying model, which could be the one of the general but need not be, in case ....

We shall suppress the discussion of compatibility of models. This means, effectively, that the models are each disjoint. It does not mean, however, that the languages are disjoint. It is certainly convenient to assume that when two models are disjoint then so are the languages. We shall make no such assumption. This has the effect that a message sent out to a group of recipients can be understood by several of them. For example, the general can issue the same utterance to both spies asking them to report what they are seeing. It is one utterance of a message with two ratified recipients. We require that ratified recipients be able to understand a message that is directed to them. For the purpose of the next definition, a language is a set of strings.

**Definition 4** *A **language community** is a pair $\Lambda = \langle \mathcal{S}, L\rangle$ where*

1. *$\mathcal{S} = \langle A, \mathcal{C}, \mathcal{P}, \iota\rangle$ is a communication structure; and*
2. *$L$ a function assigning a language to each agent.*

This is the definition of the model structure for independent agents. Each agent therefore comes equipped with a model structure plus a first-order language to talk

about it. Of course it is possible to talk about more than what we directly know about. So somewhere provision needs to be made for the fact that communication involves passing on information about models that are not under an agent's direct control.

The solution adopted here is to allow the language of communication be the union of all languages of the agents. Thus, it will be possible for *a* to talk about, say, the account of *b*, even if *a* does not have it in his model structure. Thus, if *b* says: /`I have less than 1000 dollars in my account.`/, *a* understands the message but cannot check whether it is true. Now, models determine the truth of every proposition in the language. So even if *a* were to introduce a replica of *b*s account into his domain he could not enter a specific amount there since the only thing he has been told is that there is less than a thousand dollars in it. Thus we assume that if anything, *a* will simply record *b*s statement as having been made by *b* and being true, without looking into the model structure.

# 5 Communicative Side Effects

As we said earlier, protocols may also be a way to ensure security. In many circumstances it is vital to know which information is accessible to what party after a dialogue has taken place. For example, if *a* tells *b* a secret in presence of *c* then not only does *c* now know that secret, it is common knowledge to all of them that he now does. If such a situation is not supposed to arise, *a* must choose his channel wisely.

If *a* intends to say something to *b* he may not be able to do so if he does not know how to talk to *b* at all. If *b* speaks only Bengali and *a* does not, then that is the end of the communication. Still, *a* may ratify *b* as recipient and talk English to *b* hoping *b* will pick up something.

Under normal circumstances it may be unnecessary to keep track of the languages involved. However, natural languages allow for communication whose side effect is the establishment of a new language. *a*, for example, may order *b* to open a file and name it `r2d2`. Subsequently he may order *b* to write some lines into the file. It may thus be common knowledge to both *a* and *b* that `r2d2` is the name of a file on *b*'s system. As we do not deal with shared models, the situation is this: *b* interprets the name `r2d2` as a file, while *a* only knows that *b* understands

that `r2d2` is a file (and that one can ask $b$ about its content). $a$ can at any moment retrieve the content of this file from $b$. `r2d2` has become a constant of $b$'s language.

Of course, in the previous example $b$ may know the content of the file by remembering what he asked $b$ to put into it. (This of course requires that $b$ complies with everything.) Now suppose the following. Unbeknownst to $a$, $c$ has also ordered $a$ to open a file under the name `rsd2`, and write some lines into it. In that situation neither $a$ nor $c$ fully know what the file contains, but each of them refers with the name `r2d2` to that same file. This frequently happens, for example when several people have write access to some database then each of them can change the content. On the other hand, it may be that $b$ protects both parties from interference and creates a separate file for $a$ and $c$. This situation is delicate. For now it is both the case that only $b$ interprets the utterances of $a$ and $c$, and that utterances of $a$ are interpreted differently from utterances of $c$.

Third, when $c$ orders to open a file `r3d3` then $a$ does not know about its existence, while $c$ does. Thus, the languages of communication are relative not just to one agent but to channels.

This creates a complication in the definition of a language community. Consider $a$ talking to $b$. Then we must ask which of the languages $a$ must use. It can be either $L(a)$ or $L(b)$. Which of them it is does actually depend on the type of message. In a question we must assume that $b$ knows the answer, so the language will have to be $L(b)$. In a statement it is different: it will come from $L(a)$. This is because the only source of factual knowledge is at this point the model structure. As soon as the agents come equipped with knowledge about other agents, things change. Then one can know facts without having first hand access.

**Definition 5** *The message type $\tau$ is **speaker centered** if $\tau \in \{\vdash, ☞\}$. Otherwise it is called **adressee centered**.*

**Definition 6** *A message $\langle a, e, \langle \tau, \varphi \rangle\rangle$ is **ratified** in a language community $\Lambda$ if for each $b \in \iota(e)$ and speaker oriented $\tau$, $\varphi \in L(a)$, and for each adressee oriented $\tau$, $\varphi \in L(b)$.*

The snag is that even though the languages are dependent on both agents interpretation is still one sided.

**Definition 7** *A **global** $\Lambda$-**state** is a function $\sigma$, which assigns to each agent $a$ an $L(a)$-structure $\sigma(a)$.*

So global states encode the stored knowledge of each agent. The models $\mu(a)$ can be regarded as local states.

**Definition 8** *A **history** is a sequence $\langle P_i : i < n\rangle$ where for evey $i < n$, $P_i = \langle \sigma_i, \mu_i\rangle$, where $\sigma_i$ is a global $\Lambda$-state, and $\mu_i$ is a message event such that if $i < j$ then the time stamp of $m_i$ may not be later than that of $m_j$.*

In place of a single message we can admit sets of messages, but this would complicate the picture unnecessarily; for the set can easily be decomposed as a sequence. Furthermore, the possibility of changing community structure is momentarily excluded.

The history is more than just a record of the dialogue. It also encodes the knowledge states of the agents. In this way we can also talk about proper messaging sequences. For example, if no message is sent out as an anwser to a question, this is in violation of the protocol. Apart from the first-order model that the agents keep they can gain additional knowledge about other agents and their models as well. If $b$ overhears $a$ saying that he wants to open an account he can deduce that $a$ will have an account. Also, if $a$ request his bank to disclose the balance of his account then he can deduce that what he is told *is* the balance. All this is of course subject to one important condition: that the protocol is honoured by all parties involved.

An agent can make full use of the situation by additionally keeping score of the communication. Thus agent $a$ will essentially keep a transcript of the entire dialogue; however, he can only gain access to messages that are sent through a channel that he is in. So, $a$ can record a message event $\langle t, C, m\rangle$ only if $a \in C$, that is, if $a$ is physically reachable via $C$. It is not necessary that $a$ is a ratified recipient. Any bystander can record a conversation. This complicates the situation for public announcement logic (see [Balbiani *et al.*, 2008]): a proposition is known between everyone after public announcement, but that is true only for the group $C$. For example, if I announce that there will be a test tomorrow, then this will be public knowledge for everbody present in the room but not necessarily for all students. If student $a$ is absent then I cannot deduce that he or she knows about the test. We shall return to these issues at the end.

# 6 Knowledge

Even though agents have limited access to other agent's model structure, they nevertheless can get information about it. In the scenario below the situation is that the account balance is known effectively only to the bank, not the client. If he wants to know what it is, he can ask the bank. After that he will know the balance at that moment. Some moments later however his knowledge will become obsolete. This is a situation we all know well.

First of all, to make all this work we assume that the languages used in communication are open. We simply drop the requirement that the language can be fully evaluated in the model. Alternatively, we introduce a common language for all languages and say that the interpretation function is partial.

We make three assumptions: participants are not lying, they keep the protocol, and messages are not encrypted. This means that if $a$ sends out a message $m$ at $t$ through channel $C$ then any recipient now knows that $m$. To model this, we do the following. All agents basically remember all messages they receive. This gives them maximal memory. Thus, the effect of an utterance is not only that the ratified receipient is obliged to perform certain actions, but also that all recipients, ratified or not, update their message-log. In order to effectively reason with this kind of knowledge they must of course know about the effects of actions. For example, knowing my account balance is $n$ euros, if I withdraw $m$ euros, the balance will be $n - m$. If someone paid into it without my knowledge then it may even be more.

Even if all agents record all messages they get it is not the case that they know everything. Consider for example the following question: is it possible for $b$ to know how much $a$ has in his account? To answer this question we must first of all see where the information about the account resides. In the scenario below it is $a$s bank. Thus only the bank knows the balance (not even $a$). Then we shall ask: how can that knowledge pass through the network? Here it is a matter of channels and the communication protocol. If there is no channel connection from the bank to $b$ then no message ever reaches $b$. This however is unlikely. More interesting is the case where it is the protocol that prevents certain information to be passed. Looking at the protocol specification for /`get`/ we see that the request for balance may be sent to anyone, but the bank may answer *only* if the sender is the owner of the account. Similarly we must plough through all the specifications to see how much information is disclosed and to whom. Finally, notice that even

if information may pass through the network, this consumes time and information may be useless simply because it is out of date. For example $b$ may inquire about $a$s account; if he has to wait 10 cycles to get the answer he knows the balance 10 cycles earlier, not the actual one.

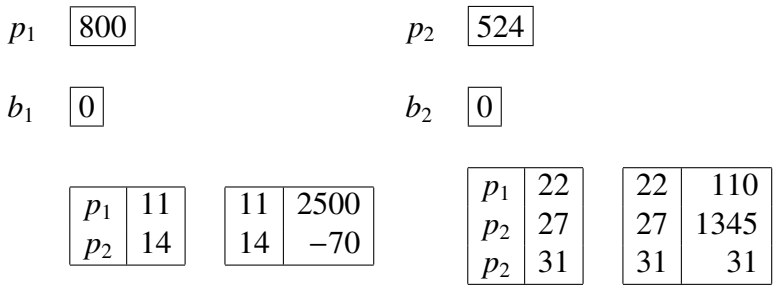
# 7 An Application

We shall discuss a scenario that exemplifies the notions discussed above. The communication structure consists of a number of banks, $b_i$, $i = 1, \cdots, m$, and a number of people $p_j$ for $j = 1, \cdots, n$, who have accounts with either or the banks. There is no limits on the number of accounts someone can have. The data structure $\Delta_i$ a bank keeps at any moment $i$ is (a) a list of pairs $(o, a)$ where $a$ is account number and $o$ its owner (one of $p_1$ through $p_n$), and (b) a list containing pairs $(a, b)$, where $a$ is the account number, and $b$ the balance. To keep track of its own actions a bank may keep a record of the $\Delta$ for past moments. A person may also have money.

There are various actions persons and banks can take. Accounts can be opened and closed. Anyone can open an account, but the number of the account is chosen by the bank (since it may not conflict with existing account numbers). The initial balance is zero. Only an owner may close an account. Accounts can be closed only if the balance is zero. Anyone can pay money into an account, but only an owner can pay out of account; the owner can pay out of his account only by ordering the bank to do so. The order is a one-time license for the bank to pay the requested sum. Money can be paid either to a person or into an account. At any stage the sum of money in the accounts and in all the person's hands is the same. Another action is a request for information. One may request the balance of an account. However, the balance may be disclosed only to the owner of the account. Generally, only information about accounts you own can be obtained.

In real life, communication with a bank is via a channel (eg the internet) and since the channel is open to anyone you need to authorise yourself in order to perform the actions. This will be ignored at the present stage. Communication is not encrypted either. The communication network is known to all participants. There are more interesting concepts left out of consideration such as the licence given by an owner of an account to another person to charge that account.

Figure 1: Two Banks and Two People




Money transfer is a complex action. If $p_i$ wishes to transfer money to $p_j$ then he chooses a bank $b$ and an account $a$ of his own, as well as a bank $b'$ and account $a'$ of $p_j$ and orders the bank to pay out of $a$ into the account $a'$ at bank $b'$. This transaction is the combination of (a) taking money out of $a$, (b) paying that same amount into $a'$. For action (a) the bank needs the permission of the owner, which we treat as implicitly given through the order. For action (b) two cases arise: $b = b'$, in which case the bank can simply write the amount into the account, or $b \neq b'$ in which case $b$ will have to ask $b'$ to put the requested amount into the account $a'$. In the latter case the communication chain is as follows: (1) $p_i$ requests from $b$ the transfer; (2) $b$ asks $b'$ to write into $a'$; (3) $b'$ acknowledges to $b$; (4) $b$ acknowledges to $p_i$. This is also the temporal order. Once $p_i$ gets the acknowledgment from the bank he knows that $p_j$ has received the money.

In practice there is a difference between on-line and off-line requests. Typically, off-line requests will not be explicitly answered. If the transaction fails, $p_i$ will simply never see the amount deducted. We shall ignore this point, however.

Lets give an example. We have two persons, $p_1$ and $p_2$ and two banks, $b_1$ and $b_2$. The network is $\{b_1, b_2, p_1, p_2\}$ and $\mathcal{C}$ is the set of all subsets of cardinality at least 2:

$$\mathcal{C} = \{\{b_1, b_2\}, \{p_1, p_2\}, \{b_1, p_1\}, \{b_1, p_2\}, \{b_2, p_1\}, \{b_2, p_2\}, \{b_1, b_2, p_1\}, \{b_1, b_2, p_2\}, \{b_1, p_1, p_2\}, \{b_2, p_1, p_2\}, \{b_1, b_2, p_1, p_2\}\} \tag{6}$$

We work with the following descriptors: $\mathcal{P} := \{\texttt{bk}, \texttt{pr}, \texttt{one}, \texttt{two}\}$. We put

$$g(x) := \begin{cases} \{b_1, b_2\} & \text{if } x = \texttt{bk} \\ \{p_1, p_2\} & \text{if } x = \texttt{pr} \\ \{b_1, p_1\} & \text{if } x = \texttt{one} \\ \{b_2, p_2\} & \text{if } x = \texttt{two} \end{cases} \tag{7}$$

This defines the communication structure.

At any given moment, the state is a quadruple $s = (m_1, m_2, \Delta_1, \Delta_2)$, where $m_1$ is the money in $p_1$'s pocket, $m_2$ the money in $p_2$'s pocket, and $\Delta_1$ and $\Delta_2$ are triples $\langle p, k, b\rangle$, where $p$ contains a number, and $k$ and $b$ are lists. An example is given in Figure 1. The following must be true:

1. $k$ consists of pairs $(p, n)$ such that $p$ is either $p_1$ or $p_2$, and $n$ is a number. No other pair $(p', n)$ may be in $k$;
2. if $b$ contains a pair $(n, b)$ then $k$ must contain a pair $(p, n)$ for some $p$; in other words, accounts must have an owner.

The first member, $p$, is a cash account (needed in transferring money).

A message event consists in an agent sending out a message $m$ (= a string) via channel $C$ at a moment $t$. The message contains an envelope with two components: sender (= $a$), adressee (= $e$) and message body. It is considered a protocol violation if $a$ is not the actual sender (as is often the case in fake emails). The remaining string is like a natural language string, containing the mood type and the proposition.

Language primitives: /open/, /close/, /get/, /pay/, /transfer/, /balance/, /from/, /to/, /EURO/, /at/, /OK/, /fail/. Also, we have the digits and the blank to form complex expressions and numbers.

In the semantics we have various types: $\pi$ the type of person, $\beta$ the type of bank, $\sigma$ the type of states, and $\nu$ the type of numbers. There are also the following semantic primitives (types are written in right associative form; eg $\alpha \to \beta \to \gamma$ is short for $\alpha \to (\beta \to \gamma)$):

① $\mathsf{open}' : \pi \to \beta \to \sigma \to \sigma$, a function from persons and banks to functions from states to states;

② $\mathsf{bic}' : \nu \to \beta$, the bank identifier code;

③ $\mathsf{close}' : \beta \to \nu \to \sigma \to \sigma$, a function from banks and numbers to state changers;

④ $\mathsf{balance}' : \beta \to \nu \to \sigma \to \nu$, a function from banks and numbers and states to numbers; this function is partial;

⑤ $\mathsf{pay}' : \beta \to \pi \to \sigma \to \sigma$, a function from banks and numbers to state changers;

⑥ $\mathsf{transfer}' : \beta \to \nu \to \nu \to \nu \to \nu \to \sigma \to \sigma$, a function that takes a bank and four numbers and changes the state.

The syntax, semantics and pragmatics of these primitives is now as follows. The syntax is given through the following context free grammar.

(8)

```
<sent>     → <command>! | <quest>? | <stat>. | <ackn>.
<quest>    → What␣is␣the␣of<acct>
<ackn>     → <iackn> | <tackn>␣<acct>
<iackn>    → OK | Sorry
<tackn>    → I␣opened | I␣closed
<command>  → <itr> | <trs>␣<acct>
             | <pay>␣<amount>␣into␣<acct>
             | <pay>␣<amount>␣out␣of␣<acct>
             | <dtrs>␣<amount>␣from␣<acct>
             ␣to␣<acct>␣at␣<bank>
<itr>      → Open␣an␣account
<trs>      → Close | Get␣the␣balance␣of
<dtrs>     → Transfer
<amount>   → <currency>␣<number>
<number>   → <digit> | <digit><number>
<currency> → EURO
<digit>    → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<acct>     → account␣<number>
<bank>     → bank␣<number>
```

This determines the syntax of messages. The following strings can be generated.

(9) `Open an account!`

(10) `Sorry.`

(11) `Transfer EURO 560 from account 017 to account 018 at bank 2!`

Message events are constructed from these strings as above. Given an event $\langle t, C, \langle a, e, \vec{x}\rangle\rangle$, where $\vec{x}$ is a string, this string is parsed and translated as follows. Since $\vec{x}$ is a sentence, it can be decomposed into a mood marker $\tau$ and a string $\vec{y}$. The complete form is thus

(12) $\langle t, C, \langle a, e, \langle \tau, \vec{y}\rangle\rangle\rangle$

In principle, one can write a Categorial Grammar for this. We prefer to spell out the definitions in a different way. Before we do so, a few general rules of protocol.

① It is required that messages are well-adressed, that is, there are no ratified non-recipients.

② All speech acts are distributive. If someone is a non-ratified recipient he must respond with a special message, say /`Sorry.`/.

③ Acknowledgments have to be sent back in this example via a channel that includes only the original sender and the recipient. This is due to privacy requirements.

We shall detail below a few characteristics of the language elements.

/`open`/.

**Protocol:** If $\Delta$ contains $\langle t, C, \langle a, e_b, \langle !, \texttt{Open}\rangle\rangle\rangle$ then $\tau$ contains $\{u, u'\}$ such that $u' = \langle t', \{a, b\}, \langle b, e_a, \langle \vdash, \texttt{I␣opened␣}\vec{x}\rangle\rangle$ or $u' = \langle t', \{a, b\}, \langle b, e_a, \langle \vdash, \texttt{Sorry}\rangle\rangle$ where $t' > t$. (Issuing the response to recipients other than $a$ is a violation of protocol. Using a channel other than $\{a, b\}$ in the response is likewise in violation of protocol. Also, no bank may open an account without being requested to do so.)

**Semantics:** For all $b \in g(e')$, the action $\mathsf{open}'(a)(b)$ is executed, where $a$ is a person and $b$ is a bank. Here, $\mathsf{open}'(a)(b)(s)$ is the following state $s' = (m_1, m_2, \Delta_1', \Delta_2')$. If $b = b_1$ then $\Delta_2' = \Delta_2$. Given $\Delta_1 = \langle \ell_1, \ell_2, \ell_3 \rangle$, we have $\Delta_1' = \langle \ell_1', \ell_2', \ell_3' \rangle$, where $\ell_1' = \ell_1$, $\ell_2' = \ell_2 \cup \{\langle a, \vec{x} \rangle\}$ and $\ell_3' = \ell_3 \cup \{\langle i, 0 \rangle\}$, where $i$ is the smallest number not assigned in $\ell_2$. If $b = b_2$ then the result is analogous with 1 and 2 interchanged. (This action is deterministics; one need not do that but it is simpler.)

So, if $a$ asks bank $b$ to open an account, the bank opens an account and communicates the account number to $a$. If $a$ asks a person, that person will respond with a failure. Normally, banks may refuse to open an account, but this is not allowed for here. Thus the protocol specifies that the request must be honoured.

*/*`close`*/.*

**Protocol:** If a person is the addressee, he must respond with `fail`. For a bank, the appropriate response is either $u' = \langle t', \{b, a\}, \langle b, e_a, \langle \vdash, \texttt{closed } \vec{x} \rangle \rangle \rangle$ where $\vec{x}$ is a number expression, or $u' = \langle t', \{b, a\}, \langle b, e_a, \langle \vdash, \texttt{fail} \rangle \rangle \rangle$. It is not licit to use a channel other than $\{b, a\}$ is a violation of protocol. (Notice that in that case it is needless to specify the recipient.) The bank issues a failure message (eg /`Sorry`/) if

1. the account balance is not 0 or
2. there is no account with that number or
3. the sender does not own the account.

**Semantics:** Let $\sigma = (m_1, m_2, \Delta_1, \Delta_2)$. $\mathsf{close}'(b)(n)(\sigma)$ is the state $(m_1, m_2, \Delta_1', \Delta_2')$ such that if $b = b_1$ then $\Delta_2' = \Delta_2$. Furthermore, if $\Delta_1 = \langle \ell_1, \ell_2, \ell_2 \rangle$ and $\langle a, n \rangle \in \ell_2$ and $\langle n, 0 \rangle \in \ell_3$ then $\Delta_1' := \langle \ell_1, \ell_2 - \{\langle a, n \rangle\}, \ell_3 - \{\langle n, 0 \rangle\} \rangle$, otherwise $\Delta_1' := \Delta_1$.

*/*`get`*/.*

**Protocol:** An appropriate response is either $\langle t', \{a, b\}, \langle b, e_a, \langle \vdash, \vec{y} \rangle \rangle$ where $\vec{y}$ is the value of $\vec{x}$ or $\langle b, e_a, \langle \vdash, \texttt{fail} \rangle \rangle$. Using a channel other than $\{a, b\}$ is prohibited. Also, the account balance may not be communicated unless $a$ owns the account.

**Semantics:** $\mathsf{balance}'(b)(n)(\sigma)$ where $\sigma = (m_1, m_2, \Delta_1, \Delta_2)$ is the unique number $m$ such that $\langle n, m\rangle \in \ell_3$, if it exists. Otherwise it is undefined.

If $a$ asks $b$ to get $\vec{x}$ (for example the balance of an account), the $b$ tries to determine the value of $\vec{x}$. If $b$ is not able to determine it, then the request is pragmatically illicit. On the other hand, in certain cases $a$ may request something that $b$ can determine but is not allowed to disclose (eg the balance of an account that $a$ does not own). In that case $b$ will also issue a failure. In the present situation we have made the choice of letting the function $\mathsf{balance}'$ be defined but prohibit the communication by the protocol. A different solution is to make $\mathsf{balance}'$ be a function that also takes a person argument and checks whether the person is allowed to gain access to the balance.

There are more functions, which we shall describe more informally. First, one can pay into an account and pay out of it. The syntax is that of a command, so it is an order to the bank to pay. When the amount is positive, the bank is asked to deduct the amount from the account and hand it to the person. When the amount is negative, the person is passing that amount to the bank and it is written into the account. To get money from an account you must be the owner. Transfer of an amount $n$ from an account $c_1$ to an account $c_2$ at bank $\beta$ works as follows. If $a$ requests from $\beta'$ the transfer, $a$ must own the account number $c_1$ at $\beta'$ and there must be enough money on it. In that case $\beta'$ pays the money into the cash account of the bank $\beta$ and sends a message to $\beta$ asking it to write the amount into the account $c_2$. If such an account exists, $\beta$ performs the requested action and sends back an acknowledgment; if no account exists, it returns the money and sends a failure notice, whereupon $\beta'$ writes the amount back into the original account. Upon completion, $\beta'$ sends an acknowledgment to $a$. The bank $\beta$ is found as follows. When $a$ issues the request, he names a bank via a number $n$. Then we have $\beta = \mathsf{bic}'(n)$.

Notice also that the same account number can be used by different banks, as is ordinarily the case. Thus, in a transfer the account numbers belong to accounts of different banks.

We shall give an example. Suppose at $t_0$, $p_1$ issues the following string with address $/\mathsf{bk} \wedge \mathsf{one}/$ via channel $\{p_1, b_1\}$.

(13) `Open an account!`

This is then represented as

(14) $\langle t_0, \{p_1, b_1\}, \langle p_1, \mathtt{bk} \wedge \mathtt{one}, \langle !, \mathtt{Open} \rangle \rangle$

$b_1$ is a ratified receipient, and will therefore perform the requested action. It chooses the number 1, adds the pair $(p_1, 1)$ to the list of accounts, and $(1, 0)$ to the account balance. Upon completion it returns acknowledgment, that is, we have the utterance

(15) $\langle t_1, \{p_1, b_1\}, \langle b_1, \mathtt{pr} \wedge \mathtt{one}, \langle \boxtimes, \mathtt{I\_opened\_account\_1} \rangle \rangle$

# 8 Modelling

We shall briefly say a few words about the actual modelling. Histories have been defined above as sequences of pairs $\langle \sigma, u \rangle$, where $\sigma$ is a state and $u$ an utterance. Because synchronicity is not of essence it is enough to consider just one utterance. In the present circumstances actions are always performed by request. Thus, in the present situation we can calculate the current state given only the protocol.

It would be nice if it were enough to just know the previous state and the previous utterance. However, this cannot be guaranteed. One problem is transfer of money. Here, a bank requests another bank to write some money into an account because a person has requested it to do so. The next step to be taken thus depends on several steps before. Thus, we need the additional notion of *session* to predict the next state. Notice also that the fact that answers cannot be adjacent to their questions in a protocol means that it cannot be the physically previous question but it must be the logically previous utterance (in terms of the turn structure) that are taken into account. However, as the previous example showed, even that may not be sufficient. Thus a *session* is defined to be a particular structure kept by each agent in order to know what to do in the next state without taking recourse to the previous states. A **session** is a pair consisting of (a) a stack of message events ("inbox") and (b) a queue of pairs of messages and channels ("outbox"). Each agent has his own session. When a message comes in it is put on the session stack. Furthermore, each time the agent is free to send a message, it outputs a message from the queue. The agents monitor at each step their respective sessions. If a message is on in the inbox stack they process it. Processing means several things; on the one hand it may consist in doing what the protocol requires. On the other it

may consist in deducing valuable information about the states of the others. The message gets removed when the protocol allows to do so (for example, when an answer is sent for a question, or an acknowledegment for a command). The nestingness of the dialogue is automatically ensured by the architecture of the session as a stack.

# References

[Alechina and Logan, 2009] Natasha Alechina and Brian Logan. A Logic of Situated Resource-Bounded Agents. *Journal of Logic, Language and Information*, 18:79–95, 2009.

[Asher, 1998] Nicholas Asher. Varieties of Discourse Structure in Dialogues. In *Proceedings of the Second Workshop on the Semantics and Pragmatics of Dialogue (Twendial '98)*, 1998.

[Balbiani *et al.*, 2008] Philippe Balbiani, Alexandru Baltag, Hans van Ditmarsch, Andreas Herzig, Tomohiro Hosi, and Santiago de Lima. 'Knowable' as 'Known After An Announcement'. *The Review of Symbolic Logic*, 1:305–334, 2008.

[Fernandez and Endriss, 2007] Raquel Fernandez and Ulle Endriss. Abstract models for dialogue protocols. *Journal of Logic, Language and Information*, 16:121–140, 2007.

[Ginzburg, 2008] Jonathan Ginzburg. Semantics for conversation. to appear, 2008.

[Goffmann, 1976] E. Goffmann. Replies and Responses. *Language and Society*, 5:257 – 313, 1976.

[Kimbrough and Thornburg, 1989] S.O. Kimbrough and M.J. Thornburg. On semantically-accessible messaging in an office environment. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Vol.III: Decision Support and Knowledge Based Systems Track*, volume 3, pages 566 – 574, 1989.

[McCawley, 1999] James McCawley. Participant Roles, Frames, and Speech Acts. *Linguistics and Philosophy*, 22:595–619, 1999.

[Weigand and Van Den Heuvel, 1998] H. Weigand and W. Van Den Heuvel. Meta-patterns for electronic commerce transactions based on FLBC. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, volume 4, pages 261 – 270, 1998.