A communication algorithm for teamwork in multi-agent environments

E. van Baars & R. Verbrugge

Ordina ICT B.V., Europaweg 31-33, 9723 AS Groningen, The Netherlands, e-mail egon@vanbaars.com & Department of Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands, email L.C.Verbrugge@rug.nl

ABSTRACT. Using a knowledge-based approach, we derive a protocol, MACOM, for the sequence transmission problem from one agent to a group of agents. The protocol is correct for communication media where deletion and reordering errors may occur. Furthermore, it is shown that after $k$ rounds the agents in the group attain depth $k$ general knowledge about the members of the group and the values of the messages. Then, we adjust this algorithm for multi-agent communication for the process of teamwork. MACOM solves the sequence transmission problem from one agent to a group of agents, but does not fully comply with the type of dialogue communication needed for teamwork. The number of messages being communicated per communication between the initiator and the other agents from the group can differ. Furthermore, the teamwork process can require the communication algorithm to handle changes of initiator. We show the adjustments that have to be made to MACOM to handle these properties of teamwork. For the new multi-agent communication algorithm, MACOM, it is shown that the gaining of knowledge required for a successful teamwork process is still guaranteed.

KEYWORDS: Epistemic logic, teamwork, communication protocol, dialogue

DOI:10.3166/JANCL.18.1–27 © Year of publication undefined Lavoisier, Paris

1. Introduction

For cooperative problem solving (CPS) within multi-agent systems, Wooldridge and Jennings give a model of four consecutive stages, namely potential recognition, team formation, plan formation, and plan execution (Wooldridge et al., 1999). Dignum, Dunin-Keplicz and Verbrugge present a more in-depth analysis of the communication and dialogues that play a role during these four stages of teamwork (Dignum et
al., 2001; Dunin-Keplicz et al., 2003). At every stage one agent of the group acts as an initiator, communicating with the other agents of the group. For a successful process of teamwork, the agents have to achieve an approximation of common knowledge through communication. In this article we aim to provide a means of reliable communication that can achieve such approximations of common knowledge in a teamwork setting. Agents communicate to each other by a communication system consisting of a connection in a communication medium between agents, together with a protocol by which the agents send and receive data over this connection.

One of the great advantages of epistemic logic is that it can be used to model communication in distributed systems (Fagin et al., 1995; Meyer et al., 1995). For example, in their classical paper (Halpern et al., 1987), Halpern and Zuck showed that epistemic logic enables perspicuous specification and verification for a number of protocols (like the alternating-bit protocol) that had been introduced for error-free transmission of sequences of messages over a distributed network. Let two processors be given, called the sender $S$ and the receiver $R$. The sender has an input tape with an infinite sequence $X$ of data elements. $S$ reads these elements and tries to send them to $R$, which writes the elements on an output tape.

The protocols are required to guarantee that (a) at any moment the sequence of data elements received by $R$ is a prefix of $X$ (safety) and (b) if the communication medium satisfies certain so-called fairness conditions, every data element of $X$ will eventually be written by $R$ (liveness). Fairness here means that infinitely many message instantiations from $S$ to $R$ and from $R$ to $S$ are delivered, guaranteeing that every message arrives eventually. To be reliable, the connection has to satisfy the fairness condition, leaving the protocol responsible for the liveness and safety properties (Halpern et al., 1987; van Baars, 2006).Besides the liveness and safety properties a protocol used in teamwork has to satisfy the requirements of teamwork explained below.

Knowledge-based algorithms like those for the alternating-bit protocol by Halpern and Zuck (Halpern et al., 1987) or TCP by Stulp and Verbrugge (Stulp et al., 2002) are meant for one-on-one communication. For announcements from one agent to a (finite) group, however, using TCP or a similar protocol with all agents separately often does not create sufficient knowledge. For example, during the stage of team formation in teamwork, the goal is to create a collective intention among a team (Dunin-Keplicz et al., 2002; Dignum et al., 2001). For this to happen, communication between the initiating agent and a group of potential agents is needed. One message with a certain content $\varphi$ is sent to all members of the group simultaneously, and a certain amount of meta-knowledge in the group about the message sent is needed, or more precisely, a $k$-fold iteration of “everyone in the group knows $\ldots \varphi$”.

The algorithms for one-on-one communication from (Halpern et al., 1987; Stulp et al., 2002) are not sufficient for this. Protocols like the alternating-bit protocol and TCP have been designed for sending a message to one other agent, not to a group. So even if one initiator sends messages simultaneously to a group of agents by TCP, this achieves meta-knowledge about these messages solely between the initiator and each agent individually, and no meta-knowledge among the group as a whole is acquired.
What is needed is an algorithm that guarantees reliable communication for one-on-group communication. In this article, two protocols that can handle one-on-group communication will be presented, in which at every round the agents in the group attain a higher level of general knowledge about the make-up of the group as well as the contents of the announced message.

We first derive an algorithm MACOM₁ for one-on-group messages. The algorithm ensures the liveness and safety property, but does not satisfy the other requirements of teamwork ¹. This algorithm solves the sequence transmission problem (Halpern et al., 1987) from one agent to a group of agents. However, the communication during teamwork is not a one-way transport of data as in the sequence transmission problem, but a dialogue. The next message is not pre-defined, but depends on the answers from the receivers (Dunin-Keplicz et al., 2003). Teamwork starts with an initiator communicating individually to the other agents, referred to as one-on-one communication. After a successful one-on-one communication, the initiator communicates the outcome to all the agents of the group, referred to as one-on-group communication. After a successful one-on-group communication, the initiator starts to communicate one-on-one again. Although ‘one-to-group’ is a more common term used in for example computer science to refer to broadcasting protocols, we rather use ‘one-on-group’ to underline that dialogue type protocols are used.

Teamwork demands flexible forms of communication. One of the properties of teamwork dialogues is that the number of messages communicated between the initiator and the other agents can differ per agent during one-on-one communication. If, for example, the initiator asks whether the abilities of the agents are sufficient for a goal, then some of the agents can answer directly. Other agents need more information to determine whether their abilities are sufficient and they answer with a request. This property is referred to as the asynchronous communication property.

Another property of teamwork is that the initiator can change. At the four different stages of teamwork (from potential recognition to plan execution), the initiator has to have different abilities (Wooldridge et al., 1999; Dunin-Keplicz et al., 2003). If an agent has all the required abilities, then it can fulfill the role of initiator throughout the whole process of teamwork. If not, then different agents can fulfill the role of initiator at different stages. During the transition from one-on-one to one-on-group communication, the initiator always stays the same, because the initiator during one-on-one communication is the only agent who has sufficient group knowledge to start communicating one-on-group (Dignum et al., 2001). At the transition from one-on-group to one-on-one communication, however, any agent from the group can bid to become the initiator. This property is referred to as the changing initiator property.

The algorithm MACOM₁ cannot handle asynchronous communication, because it uses only one index, representing the position of the message in a sequence of messages. Introducing a separate index for each sender-receiver pair solves this problem,

¹. A simulation of the protocol MACOM₁, first presented in (van Baars et al., 2006), can be found at www.ai.rug.nl/alice/mas/macom.
but only for the situation where the initiator does not change. The initiator is the sender and it increments the indices; the other agents are the receivers. When the initiator changes, the sender becomes one of the receivers and one of the receivers becomes the sender. This means that another agent now increments the indices, which can lead to several messages with the same index but containing different data, or to parallel communication processes. Introducing a two-index mechanism, where the sender and a receiver both increment their own index, partially solves these problems. The remaining problem is that an initiator change does not become general knowledge. The solution for this problem is a procedure that is not embedded in the algorithm. In this paper we show that MACOM\(_1\) can be modified to handle the asynchronous communication property and the changing initiator property. The modified communication algorithm MACOM\(_2\) (first presented in (van Baars \textit{et al.}, 2007)) guarantees a stream of accumulating messages during a teamwork process, meeting the requirements of teamwork concerning the gaining of group knowledge\(^2\).

The analysis of the algorithms yields some interesting results. For MACOM\(_1\), we will show that the depth of knowledge the sender and receiver can accumulate about messages sent is dependent upon the length of the tape and the position of information on the tape. If an infinite tape models the transmitted data, the following can be shown. For any \(k\) and any piece of data on the tape, at some point \(k\)-fold depth of general knowledge (‘everyone knows’) arises about the message, although common knowledge can never be achieved (cf. (Halpern \textit{et al.}, 1990)). Similar results are shown for MACOM\(_2\), but then the \(k\)-fold depth of knowledge about a fact depends on the number \(k\) of consecutive dialogue messages communicated after it was sent.

The rest of the paper is structured as follows. Section 2 gives a short review of epistemic and temporal notions needed for the rest of the paper. Section 3 presents an algorithm MACOM\(_1\) that solves the extension of the sequence transmission problem to one-on-group communication. Section 4 presents a proof that approximations of common knowledge are indeed attained by this algorithm. Section 5 and section 6 present the problems that arise in the flexible context of teamwork and their possible solutions, while section 7 gives the new algorithm MACOM\(_2\) incorporating the feasible solutions. In section 8 security issues with respect to MACOM\(_2\) are discussed and a possible solution is presented. Finally, section 9 closes the paper with some conclusions and ideas about further research.

2. Logical background: knowledge and time

When proving properties of knowledge-based protocols, it is usual to use semantics of interpreted systems \(\mathcal{I}\) representing the behavior of processors over time (see (Fagin \textit{et al.}, 1995)). In this formalism, one views the sender and receivers as agents, and the communication channel as the environment. For each of these, their local states,

actions and protocols can be modeled. The semantics is based on runs, which can be seen as sequences of worlds through discrete time. We give a short review. At each point in time, each of the processors is in some local state. All of these local states, together with the environment’s state, form the system’s global state at that point in time. These global states form the possible worlds in a Kripke model. The accessibility relations are defined according to the following informal description. The processor $R$ “knows” $\varphi$ if in every accessible world, namely in every global state having the same local state as processor $R$, $\varphi$ holds. In particular, each processor knows its own local state; for the environment, there is no accessibility relation. These accessibility relations are equivalence relations, obeying the well-known epistemic logic $S5^n$ (see (Fagin et al., 1995)). Next to all instantiations of propositional tautologies and the modus ponens rule, here follow the axioms and rule for individual knowledge for $i = 1, \ldots, n$:

\begin{align*}
\text{A2}_K & \quad K_i \varphi \land K_i (\varphi \rightarrow \psi) \rightarrow K_i \psi \quad \text{(Knowledge Distribution)} \\
\text{A3}_K & \quad K_i \varphi \rightarrow \varphi \quad \text{(Veracity of Knowledge)} \\
\text{A4}_K & \quad K_i \varphi \rightarrow K_i K_i \varphi \quad \text{(Positive Introspection)} \\
\text{A5}_K & \quad K_i \neg K_i \varphi \rightarrow K_i \neg K_i \varphi \quad \text{(Negative Introspection)} \\
\text{R2}_K & \quad \text{From } \varphi \text{ infer } K_i \varphi \quad \text{(Knowledge Generalization)}
\end{align*}

Here follow the axioms governing general knowledge $E_G \varphi$ (“everyone in group $G$ knows $\varphi$”) and common knowledge $C_G \varphi$ (“it is common knowledge among $G$ that $\varphi$”). Let $G \subseteq \{1, \ldots, n\}$:

\begin{align*}
\text{CK1} & \quad E_G \varphi \equiv \bigwedge_{i \in G} K_i \varphi \quad \text{(General Knowledge)} \\
\text{CK2} & \quad C_G \varphi \equiv E_G (\varphi \land C_G \varphi) \quad \text{(Common Knowledge)} \\
\text{CK3} & \quad C_G \varphi \rightarrow \varphi \quad \text{(Truth of Common Knowledge)} \\
\text{RCK1} & \quad \text{From } \varphi \rightarrow E_G (\psi \land \varphi) \text{ infer } \varphi \rightarrow C_G \psi \quad \text{(Induction Rule)}
\end{align*}

We use abbreviations for general knowledge at any finite depth. Inductively, $E^k_G \varphi$ stands for $E^k_G \varphi$ and $E^{k+1}_G \varphi$ is $E_G (E^k_G \varphi)$.

A run is a (finite or infinite) sequence of global states, which may be viewed as running through time. Time here is taken as isomorphic to the natural numbers. There need not be any accessibility relation between two global states for them to appear in succession in a run. As for notation, global states are represented as $(r, m)$ ($m$-th time-point in run $r$) in the interpreted system $I$. In particular for the temporal operators, we use the notation $\Box$ for “always from the present moment onwards” and $\lozenge$ for “at least once in the past”. Thus, we have the following truth definitions:

\begin{align*}
(I, r, m) & \models \Box \varphi \iff (I, r, m') \models \varphi \text{ for all } m' \geq m \\
(I, r, m) & \models \lozenge \varphi \iff (I, r, m') \models \varphi \text{ for some } m' < m \\
(I, r, m) & \models K_i \varphi \iff (I, r', m') \models \varphi \text{ for all } (r', m') \text{ such that } (r, m) \sim_i (r', m')
\end{align*}

Here, $\sim_i$ is the epistemic indistinguishability relation between worlds for agent $i$, where $(r, m) \sim_i (r', m')$ if and only if $r_i(m) = r_i'(m')$ (i.e. the local states for $i$ are
the same in the two global states \( r(m) \) and \( r'(m') \). Time clearly obeys the axioms of the basic temporal logic \( K_t \) (see (Goldblatt, 1992)), in which the following principle *(A)* is derivable:

\[
(A) \quad P(\Box \varphi) \rightarrow \Box \varphi
\]

To further model time, we extend \( S5^n + K_t \) with the following mixed axiom:

\[
KT1. \quad K_i \Box \varphi \rightarrow \Box K_i \varphi, i = 1, ..., n
\]

This axiom holds for systems with perfect recall (Halpern et al., 2004). Halpern et al. (Halpern et al., 2004) present a complete axiomatization for knowledge and time; however in this article we only need the axiom KT1.

### 3. An algorithm for one-on-group communication

The goal of one-on-group communication is that all the members of the group gain a certain level of knowledge about a fact \( \varphi \) sent by the sender, and that all the members gain a certain level of knowledge about the knowledge of the group of this fact \( \varphi \). This implies that the members have to gain a certain level of knowledge about which members the group consists of. When the group consists of only a sender and one receiver, we speak of one-on-one communication and the gaining of knowledge is quite straightforward as described in (Stulp et al., 2002; Halpern et al., 1987).

When the group consists of a sender and two or more receivers, it becomes a bit more complicated. The receivers of a certain fact now also have to know to whom the sender is sending this fact for gaining the above-mentioned knowledge. The solution for this is to send the information about the extension of the group together with the fact \( \varphi \). Considering the general form of a message this can be achieved in two ways. Analogously to the TCP (Postel, 1981), we will refer to the general form of a message as a package. A package consists of a data part which contains the fact to be sent and of a header which contains meta-information about the data part. Thus, the sender can put the group information \( R_G \) (the make-up of the group) in the data part of the message or in the header. The group to whom the sender is sending a certain fact \( \varphi \) is meta-information about this fact \( \varphi \), so it is preferred to store this group information in the header of a package instead of in the data part. For the one-on-group algorithm \( MACOM_1 \), we assume that the sender takes the content of his messages from an input tape with indexed data; the position of the data on the tape is included in the message package and turns out to be an important parameter when computing the iterated general knowledge gained during communication.

When the group of receiving agents is stored in the header, then as soon as any of the receiving agents \( R_i \) receives this package it knows the \( j \)-th fact \( \varphi_j \) stored in this package, \( K_{R_i} \varphi_j \), and it knows to which other receivers this message is sent, \( K_{R_i} R_G \). How does \( R_i \) know whether the other receivers received this package? The sender
has to wait with sending a package with the next fact $\varphi_{j+1}$ until it has received acknowledgements about the package with the previous fact $\varphi_j$ from all the receivers. The sender then knows that all the receivers know $\varphi_j$, and thus $K_SE_G\varphi_j$. Every receiver knows that the sender works this way, so when a receiver $R_i$ receives a package with the next fact $\varphi_{j+1}$, it knows that the sender knows that all other receivers did receive the previous package and thus know the previous fact $\varphi_j$, so $K_{R_i}K_SE_G\varphi_j$. With every repeating step of this cycle the knowledge of the sender and receivers of each others’ knowledge of the facts grows for previously sent facts and the knowledge about each others’ knowledge of the group they are in grows as well, $K_SE_G\varphi_j$ and $K_{R_i}K_SE_G\varphi_j$. The depth $k$ of knowledge gained by the members of a certain fact is equal to the amount of consecutive facts sent successfully after this fact. Because each message sent by the sender contains both a fact $\varphi$ and the make-up of the group $R_G$, the depth of knowledge within the group about the members of the group is equal to the depth of knowledge within the group about the first fact sent by the sender.

If one of the one-on-one algorithms from (Stulp et al., 2002; Halpern et al., 1987) had been used, the receiving agents would not have known that the facts $\varphi_j$ they received were sent to other receivers as well. Each of the receivers would not have known any more than that the group consisted of just the sender and itself $G = \{S, R_i\}$ instead of $G = \{S, R_1, ..., R_n\}$. The gaining of knowledge works in this case the same as mentioned above. However, the knowledge that is gained differs. The knowledge a receiver $R_i$ now has gained after having received two packages with the consecutive facts $\varphi_j$ and $\varphi_{j+1}$, is $K_{R_i}K_SE_{\{S,R_i\}}\varphi_j$, and not the much stronger $K_{R_i}K_SE_{G}\varphi_j$. So when the goal is to attain a certain depth of group knowledge, the algorithms from (Stulp et al., 2002; Halpern et al., 1987) are not sufficient.

3.1. The algorithm MACOM$_1$

The packages from our algorithm have the following form:

$$K_{\text{source}}(\text{destination}, -, \text{group}, \text{position}, -, \text{data})$$

$\text{source} =$ source port where this package is sent from $[S, R_i]$;
$K_{\text{source}} =$ the source who sends this package knows this package;
$\text{destination} =$ destination port of package $[S, R_i]$;
$\text{group} =$ group receivers to which the message is sent $[R_G, -]$ (“-” means that the sender communicates only to the destination (one-on-one communication));
$\text{position} =$ position of data from the input tape;
$\text{data} =$ data that has to be transmitted.

The next table explains variables used in the algorithm MACOM$_1$:

<table>
<thead>
<tr>
<th>Acknowledgement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ack}_{R_i}$</td>
<td>Used by $S$. Acknowledged sequence number received from $R_i$</td>
</tr>
<tr>
<td>$\text{ack}_{R_G}$</td>
<td>Used by $S$. Acknowledged sequence number for which acknowledgments were received from all $R_i$ in $G$</td>
</tr>
</tbody>
</table>
The fields filled with “−” are the checksum and window_size fields which deal with package mutation errors and congestion control (Douglas, 2006; Douglas et al., 1999). Because they are not of interest for investigating the gaining of knowledge, they are left out in this summarized protocol version. The algorithm for the sender as well as for the receiver consists of two parts, because the algorithm works in an asynchronous system in the sense that the sender and receivers do not have access to a shared clock. Thus the sending and receiving part of the algorithm work independently. One part handles the sending of the packages and the other part handles the received packages. The reception of messages is independent and works asynchronously to the sending process. These two processes affect the same local knowledge state of an agent. Though being independent, the sending and receiving algorithm influence each other’s behavior through the local knowledge state of an agent.

The algorithm \(MACOM_1\) consists of four parts. Both sender and receiver have an algorithm that handles incoming messages and an algorithm that handles outgoing messages. The lines in bold face are the lines from the algorithm and the lines between curly brackets contain some comments on them. The numbers at the beginning and at the end of the comments represent the line numbers at which the commented block of code begins or ends, respectively.

**Sender (incoming packages)**

1. \(\text{for } (i = 1 \text{ to } n)\)
   - [For all agents who sender is sending to, ...]
2. \(\text{ack}_R_i = 0\)
   - [... initialize the acknowledgement number.]
3. \(\text{end}\)
   - [\(\text{ack}_R_i\)’s initialized]
4. \(\text{while true do}\)
   - [Get ready for receiving acknowledgements from the receivers, ... [12]]
5. \(\text{when received } K_{R_i}(S, -,-, \text{seq}, -,-) \text{ do}\)
   - [You have received a package. Prepare for processing, ... [11]]
6. \(\text{if } (\text{seq} = \text{ack}_R_i+1) \text{ do}\)
   - [If this acknowledgement from \(R_i\) is equal to the next \(\text{ack}_R_i\), ... [10]]
7. \(\text{ack}_R_i = \text{seq}\)
   - [... this is the new current acknowledgement from \(R_i\), ...]
8. \(\text{store } K_SK_{R_i}(\text{seq}, -,-, \text{seq}, -,-)\)
   - [... store the fact that you know that \(R_i\) knows it.]
9. \(\text{ack}_R_G = \text{min}(\text{ack}_R_i \text{ for } i = 1 \text{ to } n)\)
   - [The highest acknowledgement by the group is equal to the lowest \(\text{ack}_R_i\).]
10. \(\text{end}\)
   - [[6] ... acknowledgement from \(R_i\), and highest group acknowledgement updated.]
11. \(\text{end}\)
   - [[5] ... finished processing of incoming package.]
12. \(\text{end}\)
   - [... [4].]
Sender (outgoing packages)

1. seq = 0

{Reading of a tape starts at position 0.}

2. while true do

{Start reading and sending an infinite tape, ... [13]}

3. read(seq, alpha)

{... read the value from the tape, ...}

4. store \( K_S(-, -, -, seq, -, alpha) \)

{... and store this information in your knowledge base.}

5. while (ack_RG \( \neq \) seq) do

{While not all receivers have acknowledged the package with sequence \( seq \) ...[11]}

6. for (i = 1 to n) do

{For all receiving agents, ...}

7. if not \( K_S K_R_i(-, -, seq, -, -) \) do

{... check if package ‘seq’ has been acknowledged yet by \( R_i \), ...}

8. send \( K_S(R_i, -, RG, seq, -, alpha) \)

{... (re)send the package to \( R_i \).}

9. end

{A package that was unacknowledged by \( R_i \), has been (re)sent.}

10. end

{A package has been (re)sent to all agents that didn’t acknowledge it.}

11. end

{[5] ... all agents \( R_i \) have acknowledged the package with sequence number \( seq \).}

12. seq = seq +1

{Move the sequence number to the next position.}

13. end

{... [2].}

Receiver (incoming packages)

1. while true do

{Get ready for receiving an infinite tape, ... [5]}

2. when received \( K_S(R_i, - , , - , seq, -, alpha) \) do

{You have received a package (from S). Prepare for processing, ... [4]}

3. store \( K_R_i K_S(-, -, RG, seq, -, alpha) \)

{Store the received package.}

4. end

{[2] ... finished processing incoming package.}

5. end

{... [1].}

Receiver (outgoing packages)

1. when \( K_R_i K_S(-, -, - , 0, -, -) \)

{Wait until the first message is received.}

2. seq = 0

{Initiate the sequence at 0.}

3. while true do

{Get ready to acknowledge incoming packages, ... [8]}

4. while not \( K_R_i K_S(-, -, seq +1, -, -) \) do

{Still not received package with sequence number ‘seq+1’, ...}

5. send \( K_R_i(S, -, -, seq + 1, -) \)

{... (re)send acknowledgement.}

6. end

{You’ve received message \( seq+1 \).}

7. seq = seq +1

{You know the sequence number of the next message. Increment seq.}

8. end

{... [3].}
4. Analysis of epistemic properties of the MACOM$_1$ algorithm

In this section we show that by passing more and more data segments, the group’s depth of knowledge about the initial message increases. For example, if a receiver stores the fact that he knows that the sender has sent the $k$-th data segment, then always in future the receiver will know that the sender knows there is a $k$-depth knowledge in the whole group that the sender knows the 0-th data segment:

$$R_i \text{ stores } K_{R_i} K_S (k, \alpha_k) \Rightarrow \Box K_{R_i} K_S (E_G K_S)^k (0, \alpha_0) .$$

4.1. Proof of the increase of group knowledge

For the readability of the proof, the form of the package is shortened to $K_{\text{source}}(\text{sequence, data})$. We assume that the group stays unchanged and that the sender $S$ sends to a receiver $R_i$ and vice versa, so the destination and group field are left out. Furthermore, we assume that no mutation errors occur, so the checksum field is also left out. We only use the sequence number in the proof; the acknowledgement number is left out. In the next table we present some relevant formulas with their informal meanings.

<table>
<thead>
<tr>
<th>Formulas</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{R_i}(j,\alpha)$</td>
<td>Receiver $i$ knows that the $j$-th data segment is $\alpha$; similar for $K_S(j,\alpha)$</td>
</tr>
<tr>
<td>$E_G(j,\alpha)$</td>
<td>Every agent in group $G$ knows that the $j$-th data segment is $\alpha$</td>
</tr>
<tr>
<td>$E^k_G\varphi$</td>
<td>Group $G$ has depth $k$ general knowledge of $\varphi$</td>
</tr>
<tr>
<td>$R_G$</td>
<td>$G$ is the current group of receivers</td>
</tr>
<tr>
<td>$P_\varphi$</td>
<td>At some moment in the past on this run, $\varphi$ was true</td>
</tr>
<tr>
<td>$\square \varphi$</td>
<td>$\varphi$ is now true and will always be true on this run</td>
</tr>
</tbody>
</table>

In the table above, the $j$ stand for natural numbers, and the data segments $\alpha$ may represent formulas in a given communication language, for example, the messages exchanged during team formation in (Dignum et al., 2001; Dunin-Keplicz et al., 2003). Note that the messages $\alpha$ always occur in the scope of knowledge operators. Just as in treatments of the bit-transmission protocol such as (Meyer et al., 1995, Section 1.9), where $K_{R_i}(x_i)$ stands for “$i$ knows what is data element $x_i$”, the notation $K_{R_i}(j,\alpha_j)$ just means that $R_i$ knows that the $j$-th data segment is $\alpha_j$. No claims are made about the truth of $\alpha_j$. We do, however, suppose that agents do not lie about the messages they have received.

**Theorem 1.** — Let $\mathcal{R}$ be any set of runs consistent with the algorithm MACOM$_1$ where:

- the environment allows for deletion and reordering errors, but no other kinds of error;
- The safety property holds (so that at any moment in an exchange between an $R_i$ and $S$, the sequence of data segments received by each $R_i$ is a prefix of the sequence of data segments sent by $S$).

Suppose that for each $j \geq 0$, the $j$-th data segment is represented by $\alpha_j$. Then for all runs in $\mathcal{R}$ and all $k \geq 0$, $j \geq 0$ the following hold:

[Forth]: $R_i$ stores $K_{R_i} K_S (j + k, \alpha_{j+k}) \Rightarrow \Box K_{R_i} K_S (E_G K_S)^k (j, \alpha_j)$.

[Back_i]: $S$ stores $K_S K_{R_i} (j + k, \alpha_{j+k}) \Rightarrow \Box K_{R_i} K_S (E_G K_S)^k (j, \alpha_j)$.

[Back_G]: If for all $R_i \in G$, $S$ stored $K_S K_{R_i} (j + k, \alpha_{j+k})$, then $\Box K_S (E_G K_S)^{k+1} (j, \alpha_j)$.
In the proof below we use a general principle from temporal logic (A), and some consequences we can derive from the assumptions of the theorem (B & C).

**A** \( P(\Box \varphi) \rightarrow \Box \varphi \)

**B** Because \( \mathcal{R} \) is consistent with the algorithm MACOM\(_1\), \( S \) and \( R_i \) store (and never forget) all relevant information from the packages that they receive. Moreover, packages that are sent have the following form: \( K_{R_i}\varphi \) or \( K_{S}\varphi \). Finally, we assume that agents observe and remember their own ‘receive’ and ‘store’ actions. From these reasonable assumptions, the following can be concluded. If \( R_i \) receives \( K_{S}\varphi \), then \( R_i \) stores \( K_{R_i}K_{S}\varphi \), thus also \( \Box K_{R_i}K_{S}\varphi \). Similarly for \( S \): If \( S \) receives \( K_{R_i}\varphi \), then \( S \) stores \( K_{S}K_{R_i}\varphi \), thus also \( \Box K_{S}K_{R_i}\varphi \).

**C** Under the same assumption of \( \mathcal{R} \) being consistent with the algorithm MACOM\(_1\), system \( \mathcal{R} \) can be viewed as a system of perfect recall. Now we have in general that \( K_{S}\Box \varphi \rightarrow \Box K_{S}\varphi \), see axiom KT1.

**Proof**

We prove theorem 1 by induction on \( k \). First we look at the situation for \( k = 0 \). From B follows the **Forth**-part for \( k = 0 \) namely

\[
\text{If } R_i \text{ stores } K_{R_i}K_{S}(j, \alpha_j) \Rightarrow \Box K_{R_i}K_{S}(j, \alpha_j). \tag{1}
\]

\( R_i \) sends an acknowledgement only if it received a package. Together with A and B we have:

- if \( R_i \) sends \( K_{R_i}(j, \alpha_j) \) then \( P(R_i \text{ stores } K_{S}(j, \alpha_j)) \),
  - so \( P\Box K_{R_i}K_{S}(j, \alpha_j) \), and \( \Box K_{R_i}K_{S}(j, \alpha_j) \).

\( S \) only stores an acknowledgement if it also received it from \( R_i \), thus it knows that \( R_i \) has sent it in the past.

- If \( S \) stores \( K_{S}K_{R_i}(j, \alpha_j) \) then \( K_{S}P(R_i \text{ sends } K_{R_i}(j, \alpha_j)) ... \tag{3} \)

With A, C and the fact proven at (2) it can now be derived that:

\[
K_{S}P(\Box K_{R_i}K_{S}(j, \alpha_j)) \text{, and } K_{S}\Box K_{R_i}K_{S}(j, \alpha_j) \text{, so } \Box K_{S}K_{R_i}K_{S}(j, \alpha_j). \tag{4}
\]

If (3) and (4) are combined, then we have the **Back**-i-part of the theorem for the \( j \)-th data segment \( (k = 0) \).

\( S \) receives acknowledgements from all the receivers and is able to retrieve information out of this. We go back two steps and look at another knowledge level of \( S \) instead of the knowledge level between \( S \) and just one receiver.

\( S \) only stores acknowledgements it received. If \( S \) has received acknowledgements of a certain package from all receivers \( R_i \in \mathcal{G} \), then \( S \) knows that all receivers have sent these acknowledgements in the past.

- If for all \( R_i \in \mathcal{G} \), \( S \) stores \( K_{S}K_{R_i}(j, \alpha_j) \) then \( K_{S}P \left( \bigwedge_{R_i \in \mathcal{G}} R_i \text{ sends } K_{R_i}(j, \alpha_j) \right) \)...
  \[ \tag{5} \]

With A, C, axiom CK1 and the fact proven at (2) it can now be deduced that:

\[
K_{S}P(\Box E_{\mathcal{G}}K_{S}(j, \alpha_j)) \text{, and } K_{S}\Box E_{\mathcal{G}}K_{S}(j, \alpha_j) \text{, so } \Box K_{S}E_{\mathcal{G}}K_{S}(j, \alpha_j). \tag{6}
\]
Induction step Suppose as induction hypothesis that \textbf{Back\_i}, \textbf{Back\_G} and \textbf{Forth} are valid for \(k-1\), with \(k \geq 1\). Now a proof follows that \textbf{Forth}, \textbf{Back\_i}, and \textbf{Back\_G} are also valid for \(k\).

\textbf{[Forth]}: \(S\) only starts sending packages with position mark \((j+k)\) if it has received from all the receivers \(R_i\) an acknowledgement for package with position mark \((j+(k-1))\):

\[
S \text{ sends } K_S \ (j+k, \alpha_{j+k}) \Rightarrow P \ (S \text{ stores } K_S \ E_G \ (j+(k-1), \alpha_{(j+k)-1})) \, . \tag{7}
\]

With the \textbf{Back\_G}-part of the theorem for \(k-1\) and \(A\), the following can be deduced:

\[
S \text{ sends } K_S \ (j+k, \alpha_{j+k}) \Rightarrow \Box K_S \ (E_G K_S)^k \ (j, \alpha_j) \, . \tag{8}
\]

\(R_i\) knows this fact. So if \(R_i\) receives a package from \(S\) with position mark \(j+k\), then \(R_i\) knows that \(S\) has sent this package somewhere in the past. From the fact given at (8) together with \(A\) and \(B\), the following can be derived:

\[
R_i \text{ stores } K_{R_i} K_S \ (j+k, \alpha_{j+k}) \Rightarrow \Box K_{R_i} K_S \ (E_G K_S)^k \ (j, \alpha_j) \, . \tag{9}
\]

This is exactly what the \textbf{Forth}-part of the theorem says.

\textbf{[Back\_i]}: \(R_i\) only sends an acknowledgement for the \((j+k)\)-th data element if he stored \(K_{R_i} K_S \ (j+k, \alpha_{j+k})\) in the past. By \textbf{Forth}, already proved above, this entails that \(P \Box K_{R_i} K_S \ (E_G K_S)^k \ (j, \alpha_j)\), so by \(A\), also \(\Box K_{R_i} K_S \ (E_G K_S)^k \ (j, \alpha_j)\).

Combining these two steps, we conclude that:

\[
R_i \text{ sends } K_{R_i} \ (j+k, \alpha_{j+k}) \Rightarrow \Box K_{R_i} K_S \ (E_G K_S)^k \ (j, \alpha_j) \, . \tag{10}
\]

\(S\) knows this fact. So if \(S\) receives an acknowledgement from \(R_i\) for the \((j+k)\)-th data segment, then \(S\) knows that \(R_i\) has sent this acknowledgement in the past. Using \(A\) and \(B\) it can now be concluded that:

\[
S \text{ stores } K_S K_{R_i} \ (j+k, \alpha_{j+k}) \Rightarrow \Box K_S K_{R_i} K_S \ (E_G K_S)^k \ (j, \alpha_j) \, , \tag{11}
\]

and this is exactly the \textbf{Back\_i}-part of the theorem.

\textbf{[Back\_G]}: \(S\) receives acknowledgements from all \(R_i \in G\). At a certain time \(S\) has received an acknowledgement for the \((j+k)\)-th data segment from all \(R_i\). Thus,

For all \(R_i \in G\), \(S\) stored \(K_S K_{R_i} \ (j+k, \alpha_{j+k})\).

With \(A\) and \(B\) and axiom \textbf{CK1} it can now be concluded that:

If for all \(R_i \in G\), \(S\) stored \(K_S K_{R_i} \ (j+k, \alpha_{j+k})\), then \(\Box K_S \ (E_G K_S)^{k+1} \ (j, \alpha_j)\), \tag{12}

and this is exactly the \textbf{Back\_G}-part of the theorem.

This finishes the proof of theorem 1. Note that, because the make-up of the group \(G\) (message \(R_G\)) is sent on a par with the initial data segment at position 0, it follows by the same proof as above that iterated general knowledge about the group accumulates during communication as follows:
A communication algorithm for teamwork

[Forth]: \( R_i \) stores \( K_R K_S (k, \alpha_k) \Rightarrow \Box K_R K_S (E_G K_S)^k R_G \).

[Back_i]: \( S \) stores \( K_S K_R (k, \alpha_k) \Rightarrow \Box K_S K_R K_S (E_G K_S)^k R_G \).

[Back_G]: If for all \( R_i \in G \), \( S \) stored \( K_S K_R (k, \alpha_k) \), then \( \Box K_S (E_G K_S)^{k+1} R_G \).

Is theorem 1 the best one can do? Certainly by the well-known result of Halpern and Moses, common knowledge within a group can never be achieved in communication environments that allow for any errors (Halpern et al., 1990). In fact, the bounds shown above are tight. So, for example, we have on the positive side Forth:

\( R_i \) stores \( K_R K_S (j + k, \alpha_{j+k}) \Rightarrow \Box K_R K_S (E_G K_S)^k (j, \alpha_j) \).

Nevertheless, at the same stage of communication \( k + 1 \)-fold knowledge has not been achieved: \( R_i \) stores \( K_R K_S (j + k, \alpha_{j+k}) \Rightarrow \neg \Box K_R K_S (E_G K_S)^{k+1} (j, \alpha_j) \).

The proof is analogous to the proof of Theorem 4 of (Stulp et al., 2002).

5. Adjusting the algorithm MACOM\(_M\) for asynchronous communication

As explained in the introduction, teamwork demands more flexibility than simple one-on-group sequence transmission: communication may be asynchronous in the sense of different numbers of messages exchanged between the sender and different receivers; and during the transition from one stage of teamwork to the next, a change of initiator may be called for. We first handle the challenge of asynchronous communication.

To handle the asynchronous communication property, a separate index is needed for every sender-receiver communication. This solution works for the situation where the initiator stays the same. For example, we take one group \( G \) consisting of three agents \( R_1, R_2, \) and \( R_3 \), \( G = \{R_1, R_2, R_3\} \). Agent \( R_3 \) is the initiating (sending) agent, temporarily denoted as \( S_3 \), and the two other agents \( R_1 \) and \( R_2 \) are the receivers. The index that \( S_3 \) uses to communicate with \( R_1 \) starts at 100 and the index that \( S_3 \) uses to communicate with \( R_2 \) starts at 200. Let us work out an example. \( S_3 \) sends three messages to \( R_1 \), which are received and answered by \( R_1 \). These answers can be an answer to a question or request sent by \( S_3 \) or just an acknowledgement if \( S_3 \) sent a statement.

In the notation below, the agents are identified by the numbers 1, 2, and 3. If an agent acts as a sender or receiver, then this is denoted by \( S_1 \) or \( R_1 \) respectively. The agents exchange messages and the arrow \( \rightarrow \) indicates the direction of each message. The messages are of the form \( (100,,\_\_\_\_\_,data) \). The first field contains a sequence number. The second field contains the group information. In the case of one-on-one communication the value of this field is ‘\( G \)’ and in the case of one-on-group communication the value of this field is ‘\( G \)’. The last field contains the data that is sent.

1. \( S_3 (100,,\_\_\_\_\_,data) \rightarrow R_1 \)
2. \( S_3 <-(100,,\_\_\_\_\_,answ) R_1 \)
3. \( S_3 (101,,\_\_\_\_\_,data) \rightarrow R_1 \)
4. \( S_3 <-(101,,\_\_\_\_\_,answ) R_1 \)
5. \( S_3 (102,,\_\_\_\_\_,data) \rightarrow R_1 \)
6. \( S_3 <-(102,,\_\_\_\_\_,answ) R_1 \)

This moves the index for the next message to be sent to \( R_3 \) to 103. \( S_3 \) communicates two messages with \( R_2 \), which are answered by \( R_2 \), as follows:
1. S3 (200,_,data)-> R2
2. S3 <- (200,_,answ) R2
3. S3 (201,_,data)-> R2
4. S3 <- (201,_,answ) R2

This moves the index for the next message to be sent to \( R_2 \) by \( S_3 \) to 202. During both these one-on-one communications, \( S_3 \) has reached the goal for this phase and is now ready to communicate the outcome one-on-group to \( R_1 \) and \( R_2 \). To announce the outcome, \( S_3 \) has to communicate two messages one-on-group, which are answered by \( R_1 \) and \( R_2 \):

1. R1 <- (103,G,data) S3 (202,G,data)-> R2
2. R1 (103,_,answ)-> S3 <- (202,_,answ) R2
3. R1 <- (104,G,data) S3 (203,G,data)-> R2
4. R1 (104,_,answ)-> S3 <- (203,_,answ) R2

After this successful one-on-group communication, \( S_3 \) enters the next stage in order to communicate one-on-one again with the others in \( G \). The indices for the next message to \( R_1 \) and \( R_2 \) are 105 and 204, respectively. Introducing a separate index for each sender-receiver pair in the communication solves the problem of the different numbers of messages sent during the one-on-one communication phase. Does this solution also work for the situation where the initiator changes after one-on-group communication?

6. Adjusting the algorithm MACOM₁ for changing initiators

The last example from the previous section ends with a successful one-on-group communication. Let us go from there while \( R_2 \) now takes over the role of initiator, temporarily denoted as \( S_2 \). The previous initiator \( S_3 \) is denoted again as \( R_3 \). The communication between \( S_2 \) and \( R_1 \) is straightforward. Because \( S_2 \) did not communicate to \( R_1 \) before, \( S_2 \) sets a new index. The last communication between \( S_2 \) and \( R_3 \) was the message \((203,\_,answ)\), sent from \( R_2 \) to \( S_3 \). Now, \( S_2 \) wants to send some data to \( R_3 \). Which index does it have to use? One possibility could be that \( S_2 \) sets a new index for this communication, starting for example at 400. Another possibility is that \( S_2 \) continues with the index used by \( S_3 \) while communicating one-on-group to \( R_2 \). In this case, \( S_2 \) can use the same index number, 203, as used during its last answer message to \( S_3 \). Alternatively \( S_2 \) can use the next index number, 204. Let us develop these three options. The last two communication lines of the previous one-on-group communication are taken as a starting point and are repeated in the examples.

**Option 1, \( S_2 \) sets new index:**

1. R1 <- (104,G,data) S3 (203,G,data)-> R2
2. R1 (104,_,answ)-> S3 <- (203,_,answ) R2
3. R3 <- (400,_,data) S2
4. R3 (400,_,answ)-> S2
5. R3 <- (401,_,data) S2
Option 2, $S_2$ reuses the last index number:

1. $R_1 <-(104,G,data) S_3 (203,G,data)-> R_2$
2. $R_1 (104,_,answ)-> S_3 <-(203,_,answ) R_2$
3. $R_3 <-(203,_,data) S_2$
4. $R_3 (204,_,answ)-> S_2$
5. $R_3 <-(204,_,data) S_2$

Option 3, $S_2$ uses the next index number:

1. $R_1 <-(104,G,data) S_3 (203,G,data)-> R_2$
2. $R_1 (104,_,answ)-> S_3 <-(203,_,answ) R_2$
3. $R_3 <-(204,_,data) S_2$
4. $R_3 (204,_,answ)-> S_2$
5. $R_3 <-(205,_,data) S_2$

All the above options show some anomalies in the index numbering with respect to being an accumulating stream of messages.

For option 1, two different consecutive communication streams run between agent 2 and agent 3. This can lead to parallel communication streams if agent 3 continues communicating as initiating agent $S_3$ to agent 2, while agent 3 as receiver $R_3$ also receives messages from $S_2$. Two parallel communication processes between two agents about the same process are prone to communication errors and should be avoided.

For option 2, two anomalies can occur. The first one is that in one-on-one communication the receiver increases the index with every answer instead of the sender. So, when $R_3$ sends an answer, it acknowledges an index it did not receive yet. The second anomaly can arise when agent 2 uses the same index number for the next consecutive message. If the previous message was just an acknowledgement, then there is no problem. Acknowledgements do not occupy an index number, otherwise we would end up with acknowledging acknowledgements (Postel, 1981). If $R_2$ sent data instead of just an acknowledgement to agent 3 in the previous message, then agent 2 cannot send the next consecutive message with the same index number. When agent 3 answers with just an acknowledgement, agent 2 does not know which of the two different consecutive messages is acknowledged by agent 3.

For option 3, agent 3 might send a next message (204,_,data) to agent 2 and receive from agent 2 a message (204,_,data) instead of (204,_,answ). Thus, both agents sent a data message with index 204 and also received a data message while both agents expected an answer message. This situation should be avoided.

6.1. Two-index mechanism to the rescue

How can these problems be solved? The transmission control problem (TCP) makes use of two indices per connection (Stulp et al., 2002; Postel, 1981). One index is configured by the sender and the other index is configured by the receiver. Thus every message contains a sequence number as well as an acknowledgement of the last consecutive sequence number that is received. Could this two-index system solve the index numbering problems?

Let us look at a one-on-one communication process ending with a one-on-group communication. Agent $S_3$ sends two messages to agent $R_1$ which are answered by
agent $R_1$, and sends one message to agent $R_2$ which is answered by agent $R_2$. Next, agent $S_3$ sends one message one-on-group to agent $R_1$ and $R_2$ which is answered by both agents after which agent $S_3$ starts communicating one-on-one to agent $S_1$ and $S_2$ again. In his first message to an agent, the sender conveys only its own sequence number. When the receiver receives this, it initiates its own sequence number and answers with a message containing this number together with the acknowledged sequence number from the sender. Thus after two messages, the sender and receiver know one another’s sequence numbers. The messages are now of the form $(100,200,\_\_\_,data)$. The first field contains the sequence number of the agent that sends the message. The second field contains the acknowledged sequence number of the message the agent is reacting to. The third field contains the group information and the last field contains the data that is sent.

1. $R_1 \leftarrow (100,\_\_\_,\_\_\_,data) \rightarrow S_3$
2. $R_1 (200,100,\_\_\_,answ)\rightarrow S_3$
3. $R_1 \leftarrow (101,200,\_\_\_,data) \rightarrow S_3 (300,\_\_\_,\_\_\_,data)\rightarrow R_2$
4. $R_1 (201,101,\_\_\_,answ)\rightarrow S_3 \leftarrow (400,300,\_\_\_,answ) \rightarrow R_2$
5. $R_1 \leftarrow (102,201,G,\_\_\_,data) \rightarrow S_3 (301,400,G,\_\_\_,data)\rightarrow R_2$
6. $R_1 (202,102,\_\_\_,answ)\rightarrow S_3 \leftarrow (401,301,\_\_\_,answ) \rightarrow R_2$

This works straightforwardly, so let us look how this two-index mechanism works when the initiator changes. Lines 5 and 6 from the previous communication schema are used as starting point, and agent 2 becomes the sender. The first option with the one index mechanism is that $S_2$ sets a new index to communicate with $R_3$. There are already two indices between $S_2$ and $R_3$, so it is not necessary to set a new index. $S_2$ and $R_3$ start communicating one-on-one, continuing the use of the indices they already used during the previous one-on-group communication. This eliminates the problem of two parallel communication processes between both agents. Now two options are left for $S_2$ when using the two-index number mechanism. The first option is to reuse the last index number and the second option is to use the next index number. Worked out, these options look as follows.

**Option 1, $S_2$ reuses the last index number:**

1. $R_1 \leftarrow (102,201,G,\_\_\_,data) \rightarrow S_3 (301,400,G,\_\_\_,data)\rightarrow R_2$
2. $R_1 (202,102,\_\_\_,answ)\rightarrow S_3 \leftarrow (401,301,\_\_\_,answ) \rightarrow R_2$
3. $R_3 \leftarrow (401,301,\_\_\_,data) \rightarrow S_2$
4. $R_3 (302,401,\_\_\_,answ)\rightarrow S_2$
5. $R_3 \leftarrow (402,302,\_\_\_,data) \rightarrow S_2$

**Option 2, $S_2$ uses the next index number:**

1. $R_1 \leftarrow (102,201,G,\_\_\_,data) \rightarrow S_3 (301,400,G,\_\_\_,data)\rightarrow R_2$
2. $R_1 (202,102,\_\_\_,answ)\rightarrow S_3 \leftarrow (401,301,\_\_\_,answ) \rightarrow R_2$
3. $R_3 \leftarrow (402,301,\_\_\_,data) \rightarrow S_2$
4. $R_3 (302,402,\_\_\_,answ)\rightarrow S_2$
5. $R_3 \leftarrow (403,302,\_\_\_,data) \rightarrow S_2$

For option 1, the anomaly of the receiver increasing the index (as happened with one index) does not occur. However, the second anomaly still exists. Agent 2 still sends two messages with the same index number containing different data. For option 2, agent 2 sends two messages with the same acknowledgement number, but it increases its own sequence number. Again a similar problem can arise as with the single index.
mechanism. It is possible that agent 3 sends a next message, (302,401,_,data), to agent 2 while it receives from agent 2 a message (402,301,_,data) instead of (402,302,_,answ). As can be seen, the index numbering is now completely messed up. Both agents will not know how to proceed so this situation should be avoided.

6.2. Who’s the ‘boss’

Using a two-index mechanism solves some but not all of the problems that arise while the initiator changes. The problems that are left have a single cause. When a new agent becomes the initiator, this is not general knowledge. Another agent from the group can start acting as an initiator while the current initiator continues acting as an initiator as well. This leads to problems between these two agents as discussed in the previous subsection, but also leads to problems for the other agents in the group, continuing to act as receivers. These agents start getting one-on-one communication messages about the next stage from different agents acting as initiator. Obviously this is not a workable situation. To solve this problem, the algorithm has to provide a mechanism that prevents multiple concurrent initiators.

An initiator change takes place at the transition from a successful one-on-group communication to the next one-on-one communication process. The solution for preventing multiple concurrent initiators is that if any new agent wants to act as initiator, then this agent notifies the current initiator of this fact. Every potential new initiator sends a request with its acknowledgement of the last one-on-group message. The current initiator now knows whether there are other candidate initiators and can decide whether it continues as an initiator itself, or allows one of the other agents to act as initiator.

If the current initiator decides to stay on, then it continues communicating one-on-one concerning the next stage. As soon as an agent that announced itself as a new initiator receives the first one-on-one communication message from the sender, it knows that it should not act as initiator. If the current initiator decides that one of the other agents can take over, then it sends a message one-on-one to this agent confirming that it is the new initiator. After the initiator for the next stage receives this message, it knows its new role and starts communicating messages one-on-one concerning the next stage. As soon as the other agents that announced themselves as new initiator receive the first one-on-one communication message from the new initiator, they know that they should not act as initiator. We assume that agents involved in teamwork are cooperative, so if one of the other agents has better resources for being the new initiator, then the current initiator transfers the role of initiator to that agent.

Let us develop two examples. In both, the current initiator and two other agents want to act as initiator. In the first example, the initiator changes, while in the second example, the initiator stays the same. An agent announcing itself as a potential initiator for the next stage is represented by the value init in the data field. If the current initiator decides that another agent can have the role of initiator, then it sends a message containing answ into the data field.
Example 1, \(S_2\) as initiator after init request from \(R_1\) and \(R_2\).

1. \(R_1 <-(102,201,G,\text{data}) S_3 (301,400,G,\text{data}) \rightarrow R_2\) 
2. \(R_1 (202,102,_,\text{init}) \rightarrow S_3 <-(401,301,_,\text{init}) R_2\) 
3. \(S_3 (302,401,_,\text{answ}) \rightarrow R_2\) 
4. \(R_3 <-(402,302,_,\text{data}) S_2 (500,_,_,\text{data}) \rightarrow R_1\) 
5. \(R_3 (303,402,_,\text{answ}) \rightarrow S_2 <-(600,500,_,\text{answ}) R_1\) 
6. \(R_3 <-(403,303,_,\text{data}) S_2 (501,600,_,\text{data}) \rightarrow R_1\)

Example 2, \(S_3\) stays the initiator after init request from \(R_1\) and \(R_2\).

1. \(R_1 <-(102,201,G,\text{data}) S_3 (301,400,G,\text{data}) \rightarrow R_2\) 
2. \(R_1 (202,102,_,\text{init}) \rightarrow S_3 <-(401,301,_,\text{init}) R_2\) 
3. \(R_1 <-(103,202,_,\text{data}) S_3 (302,401,_,\text{data}) \rightarrow R_2\) 
4. \(R_1 (203,103,_,\text{answ}) \rightarrow S_3 <-(402,302,_,\text{answ}) R_2\) 
5. \(R_1 <-(104,203,_,\text{data}) S_3 (303,402,_,\text{data}) \rightarrow R_2\) 
6. \(R_1 (204,104,_,\text{answ}) \rightarrow S_3 <-(403,303,_,\text{answ}) R_2\)

In the above two examples, no anomalies in the index numbering are present. The combination of the two-index mechanism with the mechanism regulating the change of initiator handles the problems that could occur when the initiator changes during the teamwork process. In the next section, we outline a new algorithm, \(MACOM_2\), guaranteeing knowledge transfer even in those flexible teamwork situations where initiators may change.

7. An algorithm for flexible team dialogues

In sections 5 and 6, it was shown which adjustments had to be made to ensure the group’s appropriate gain of knowledge for the asynchronous communication and changing initiators. Let us have a look at the adjusted algorithm. The messages from the \(MACOM_1\) algorithm have the following form:

\[
K_{\text{source}}(\text{destination}, -, \text{group}, \text{position}, -, \text{data})
\]

The fields filled with “-” are the checksum and window_size fields, dealing with package mutation errors and congestion control (Douglas, 2006; Stulp et al., 2002). As discussed in section 6, an algorithm for teamwork needs an index mechanism consisting of two indices. The window_size is used for the sliding window (Postel, 1981) mechanism which is not used during dialogue. This allows us to use the window_size field as the second index field. Because the checksum field does not contribute to the gaining of knowledge, it is filled with “-”. The first index contains the sequence number of the agent who sends the message, and the second index field contains an acknowledgement of the sequence number of the message this agent reacts to. These fields are called the sequence field and the acknowledgement field, respectively. The message used by the team-specific algorithm \(MACOM_2\) has the following form:

\[
K_{\text{source}}(\text{destination}, -, \text{group}, \text{sequence}, \text{acknowledgement}, \text{data})
\]
A communication algorithm for teamwork

Here follows a description of the fields in the messages used in MACOM$_2$; these are the same as those used for MACOM$_1$ in section 3.

- **source** = source port where this message is sent from $[S, R_i]$;
- **K$_{source}$** = the source who sends this message knows this message;
- **destination** = destination port of message $[S, R_i]$;
- **group** = group receivers to which the message is sent $[R_G, -]$ ("-") means that the sender communicates only to the **destination** (one-on-one communication);
- **sequence** = sequence number of message from agent who sends this message;
- **acknowledgement** = sequence number of message that agent is reacting to;
- **data** = data that has to be transmitted.

The next table explains variables and functions used in the teamwork-specific algorithm MACOM$_2$:

<table>
<thead>
<tr>
<th>Acknowledgement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ack$_{R_i}$</strong> : Used by $S$. Acknowledged sequence number received from $R_i$</td>
</tr>
<tr>
<td><strong>seqSR$_i$</strong> : Used by $S$. Sequence number of messages $S$ is sending to $R_i$</td>
</tr>
<tr>
<td><strong>seqS</strong> : Used by $R_i$. Sequence number of messages $R_i$ is receiving from $S$</td>
</tr>
<tr>
<td><strong>seqR$_i$</strong> : Used by $R_i$. Sequence number of messages $R_i$ is sending to $S$</td>
</tr>
<tr>
<td><strong>seqR$_i$</strong> : Used by $S$. Sequence number of messages $S$ is receiving from $R_i$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>compose()</strong> : Used by $S$ and $R_i$. Agent makes up the data it wants to send</td>
</tr>
</tbody>
</table>

7.1. The team dialogue algorithm MACOM$_2$

Just like MACOM$_1$, the algorithm MACOM$_2$ consists of four parts. Both sender and receiver have an algorithm that handles incoming messages and an algorithm that handles outgoing messages. The lines in bold face are the lines from the algorithm and the lines between curly brackets contain some comments on them. The numbers at the beginning and at the end of the comments represent the line numbers at which the commented block of code begins or ends, respectively.

Compared to MACOM$_1$, the new algorithm has quite similar parts for the incoming messages for both Sender and Receivers, but the ones for outgoing messages are considerably different. Most adaptations for asynchrony and changing initiator are reflected in the sender’s part concerning outgoing messages. We do assume that there is one designated initiator who has the role of Sender $S$ at the start of the process of teamwork.
Sender (incoming packages)

1. for (i = 1 to n) do
   [For all agents who sender is sending to, ... ]
   2. ack_Ri = seqSRi
      [... initialize the acknowledgement number.]
   3. end
      [ack_Ri’s initialized]
   4. while true do
      [Get ready for receiving acknowledgements from the receivers, ... [11]]
   5. when received K_Ri(S, 0, 0, seqRi, seqSRi, data) do
      [You have received a package. Prepare for processing, ... [10]]
   6. if (seqSRi = ack_Ri + 1) do
      [If this acknowledgement from Ri is equal to the next ack_Ri, ... [9]]
   7. ack_Ri = seqSRi
      [... this is the new current acknowledgement from Ri, ...]
   8. store K_S K_Ri(S, 0, 0, seqRi, seqSRi, data)
      [... store that you know that Ri knows it.]
   9. end
      [16] ... acknowledgement from Ri, and highest group acknowledgement updated.]
10. end
      [[5] ... finished processing of incoming package.]
11. end
      [[4].]

Sender (outgoing packages)

1. for (i = 1 to n) do
   [For all receiving agents.]
   2. if not seqSRi do
      [If S did not communicate to Ri before]
   3. seqSRi = x
      [Initiate own sequence number for Ri at x]
   4. end
      [seqSRi initiated.]
   5. end
      [seqSRi for all receiving agents initiated.]
   6. while true do
      [Start sending sequence of messages, ... [20]]
   7. compose(data)
      [... make up the data for this package, ...]
   8. store K_S(S, 0, 0, seqRi+1, SeqSRi, data)
      [... and store this information in your knowledge base.]
   9. while (∃ack_Ri ≠ seqSRi) do
      [While not all receivers acknowledged the package with sequence seqSRi, ... [15]]
  10. for (i = 1 to n) do
      [... and for all receiving agents, ... [14]]
  11. if not K_S K_Ri(S, 0, 0, seqRi, seqSRi, data) do
      [... check if package ‘seqSRi’ has not been acknowledged yet by Ri, ... [13]]
  12. send K_S(Ri, 0, 0, seqRi, seqSRi, data)
      [... (re)send the package to Ri.]
  13. end
      [[11] ... A package that was unacknowledged by Ri, has been resent.]
  14. end
      [[10] ... A package has been resent to all agents that didn’t acknowledge it.]
  15. end
A communication algorithm for teamwork

for (i = 1 to n) do
  for all receiving agents, ...
  seqRi = seqRi + 1
    Sequence number of next message from Ri is known. Increment seqRi.
  seqSRi = seqSRi + 1
    Increment own sequence number for Ri.
end

Receiver (incoming packages)

while true do
  for receiving sequence of messages, ...
  when received \( K_S(R_i, G, \ldots, seqS, seqR_i, data) \) do
    You have received a package (from S). Prepare for processing.
  end
  store \( K_R, K_S(-, -, G, seqS, seqR_i, data) \)
    Store the received package.
  end
  [[2] ... finished processing incoming package.]
end

Receiver (outgoing packages)

when \( K_R, K_S(R_i, -, G, x, \emptyset, data) \)
  The first message is received.
seqS = x
  The first sequence number from S is x.
seqRi = y
  Initiate own sequence number at y.
while true do
  for getting ready to acknowledge incoming packages, ...
  when not \( K_R, K_S(R_i, -, G, seqS + 1, seqR_i, data) \) do
    Still not received package with ‘seqS+1’ (and ‘seqR’). ...
  end
  send \( K_R, (S, -,-, seqR_i, seqS, data) \)
    [... (re)send data package.]
  end
  [[6] ... You’ve received message seqS+1 wiht acknowledgement seqRi]
seqS = seqS+1
  You know the sequence number of the next message. Increment seqS.
seqRi = seqRi+1
  Increment own sequence number, seqRi.
end
[[4].]

For the adjustments for asynchronous communication and changing initiators, we showed informally in this article in sections 5 and 6 that they ensure the required knowledge gaining for teamwork. The formal proof that MACOM ensures appropriate levels of general knowledge is an adaptation of the proof in subsection 4.1. The gain of knowledge applies to each exchange between one sender and set of receivers separately. Moreover, at each stage in a team dialogue, the receiving agents
know the identity of the sender (initiator) from whom they receive a message. Thus, the information gain from successive stages with possibly different initiators can be appropriately combined.

8. The teamwork algorithm MACOM$_2$ and security

Security in the MACOM$_2$ protocol can be viewed on two levels: the security of the reliability of the communication; and the security of the information that is being communicated. Let us investigate both in turn.

8.1. Security and reliability of the communication

A communication system is reliable if it satisfies the safety, liveness and the fairness properties. As discussed in section 1, the protocol is responsible for safety and liveness. An attacker could try the following options during execution of MACOM$_2$: Read package; delete package; insert package; or modify package. What are the consequences for the reliability of MACOM$_2$ if an attacker executes the above-mentioned actions?

Read An attacker reading one, or more, of the packages being sent has no consequences for the reliability of the communication. The packages sent are delivered as usual by the receivers.

Delete An attacker deleting randomly one (or a few) packages has no consequence for the reliability of the communication. The deleted packages will be resent and none of the reliability properties are violated. However, if the attacker can read all packages and deletes all instances of one package, then the fairness property is violated.

Insert Without specific information about the communication of a group, an attacker inserting a package has no consequence for the reliability of communication. An attacker needs to know one agent it can send the package to and needs to know a second agent it can pretend the package is being sent from. One of these agents has to be the current initiator. Besides knowing such a sender-receiver pair, the attacker needs to know the actual sequence and acknowledgement number, as well as the right moment to insert a package. The inserted package has to be received by the destination agent before this agent receives the original package with the same sequence numbers. An attacker can obtain this information by reading the packages being sent. Finally, the attacker needs to know how the checksum is calculated. It cannot obtain this knowledge by reading packages, because the method for calculating the checksum is not presented there. The attacker needs another source where it can obtain knowledge about calculating the checksum. A successful inserted package violates the safety property.

Modify An attacker modifying a package has in principle no consequence for the reliability of communication. An attacker needs to know how to calculate the checksum, because this needs to be recalculated after the message has been modified. An attacker also needs to know the right moment to modify a package, because if it modifies an instance of a package after the receiver already received another instance of this package, then the receiver will delete the modified package. The attacker can determine the right moment from reading the packages sent. A successfully modified package violates the safety property.
Summing up, it is hard for an attacker to do real harm. To successfully perform deletion, insertion or modification, an attacker must read the packages being sent. Thus if a security mechanism is added to the MACOM₂ protocol that prevents an attacker to read packages, the reliability of the communication will be guaranteed.

8.2. Security and the information being communicated

A package consists of a header and a data field. The header consists of different fields containing meta-information about the package. Which information can an attacker attain from these fields while eavesdropping on the communication?

**Source field** tells the attacker the identity of one agent, and also divulges that this agent is communicating with at least one other agent. If the group field of the package is filled, then the source field also tells the attacker that this agent is the initiator and that this agent is communicating one-on-group.

**Destination field** tells the attacker the identity of an agent and that this agent is communicating with another agent.

**Checksum field** tells the attacker what the value is of the checksum of a package. This value does not reveal any information about the group or what they are communicating about.

**Group field** can be filled or can be empty. In case it is filled, it concerns a package sent by the initiator during one-one-group information. A filled group field tells the attacker the identity of all the receivers of the group as well as the fact that these agents are involved in teamwork. In case this field is empty, it can concern a package sent during one-on-one communication by the initiator or by the receiver. The group field is also empty when a receiver acknowledges a one-on-group message from the initiator. Thus, an empty group field is not necessarily an indication for one-on-one communication.

**Sequence field** tells the attacker what the current sequence number is from the source agent in his communication with the destination agent. This value does not reveal any information about the group or what they are communicating about.

**Acknowledgement field** tells the attacker which sequence number belongs to the destination agent to whom the source agent reacts. This value does not reveal any information about the group or what they are communicating about.

**Data field** contains the messages sent during the teamwork dialogue. This field can tell the attacker a lot of information, such as the goal of teamwork, the current stage of teamwork, as well as information about the qualities of the individual members of the team. So if an attacker is able to eavesdrop on communication, then even from reading one or a few packages, it can obtain a lot of information about the teamwork. The consequences of an attacker eavesdropping on communication depends on the possible damage for the goal of teamwork. For a team of scientists developing a new medicine, it can be disastrous if a third party gets their hands on the information communicated. Costly investments can be lost if another company also knows how to produce a new medicine without having any research costs.
Summing up, the security of the communicated information is violated if an attacker is able to read the messages sent: a similar conclusion as we drew before about the security and the reliability of the communication. The solution for both security problems is to prevent an attacker to read the packages. In the sequel, we sketch some possible solutions.

8.3. Encryption as a solution for security of teamwork communication

One solution for preventing an attacker from reading packages is to build a network for the members of the team only, physically inaccessible for any other agents. This solution might work for the last two stages in teamwork, namely plan formation and plan execution, when the extension of the team has been settled (Wooldridge et al., 1999; Dunin-Keplicz et al., 2003). A team is established at the end of the second stage of teamwork (Dignum et al., 2001). Because there is no team yet during potential recognition, building a network for team members only is not the security solution for teamwork.

Another solution is to encrypt the packages. The attacker can still read part of the packages, but is not able to retrieve the information which is stored in its encrypted fields. Let us examine which fields of a package can be encrypted and how that affects security. The only field that cannot be encrypted is the destination field. The communication medium used to transmit the packages needs to know to which agent a package has to be delivered. Therefore, an attacker will always be able to obtain the identity of the destination agent. The attacker will also know that this agent is communicating with at least one other agent. The checksum field does not reveal any information about the group or what they are communicating about and has no value for an attacker trying to violate the reliability of the communication. So there is no need for encrypting this field.

The MACOM\textsubscript{2} algorithm provides a dialogue type of communication. All packages sent by an initiator will be acknowledged by the receiver. Because an attacker can read the destination agent of every package, it is able to reveal the identity of all agents sending packages in a communication medium. By reading the destination fields, the attacker cannot obtain information about the make-up of current teams.

In case of a one-on-group package, the group and source fields reveal the members of a team and reveal which agent is the initiator. Encrypting these fields prevents the attacker from directly obtaining information about a team from one package. Even when the source field is encrypted, an attacker can still retrieve information about which sender-receiver pairs are communicating by reading the sequence and acknowledgement fields. The data field can contain valuable information about the team’s goal. It is obvious that this field should also be encrypted. To prevent an attacker from violating the reliability of the communication and from obtaining information communicated within the group, the following fields should be encrypted: source, group, sequence, acknowledgement, and data field. In order to formally investigate the security of MACOM\textsubscript{2} combined with appropriate encryption, one could use model checking methods as proposed in (Armando et al., 2009; Boureanu et al., 2009).
9. Discussion, conclusion and future work

This research falls in the tradition, starting with Halpern and Zuck’s landmark paper (Halpern et al., 1987), of using interpreted multi-agent systems to analyze communication protocols, and including the analysis of file transmission protocols such as (Lomuscio et al., 2004). In (Stulp et al., 2002) Stulp and Verbrugge extend the knowledge-based approach to the Transmission Control Protocol (TCP), which is indispensable for the Internet today. The main contribution there is the modeling of a sliding window, allowing agents to make use of available bandwidth, and an epistemic analysis in which exact lower and upper bounds on the attained knowledge of the participants at every moment in the communication process are proved. However, TCP does not allow for creating iterated general knowledge within a group.

Our aim in this paper has been to make communication protocols much more flexible than file transmission protocols, in order to adapt them to dialogue-based teamwork. There, more interactive inter-group communication is needed than can be achieved by simply transmitting the same infinite sequence of bits from an initiator to each agent in the rest of his team individually. In this paper a knowledge-based algorithm $MACOM_1$ for multi-agent communication is presented, and subsequently adjusted for dialogue communication in teamwork. It is shown how the adapted protocol $MACOM_2$ handles the different numbers of messages between the initiator and different members and the changing initiator property, guaranteeing the knowledge gain required for teamwork. An algorithm supporting the dynamic properties of teamwork communication enables a flexible approach to teamwork. In order to make $MACOM_2$ secure against attackers, however, it needs to be complemented with encryption of the relevant fields of the packages.

This research complements other literature that aims to make Wooldridge’s and Jennings’ cooperative problem solving model (Wooldridge et al., 1999) more flexible, for example, (Dunin-Keplicz et al., 2004) where the needed group attitudes for teamwork are adjusted to properties of the environment and the organization. Durfee et al. present another model of cooperative problem solving (Cox et al., 2005). Their idea of partial global planning interleaves plan execution with stages of gradually specifying the global plan in more detail. This seems to be an appropriate model for long term software development projects, where teams change over time. It would be interesting to see whether communication during teamwork based on such more flexible models can be handled similarly to the knowledge-based algorithm presented here, by a modular approach that can be instantiated for specific models of teamwork.

In the present work, we have concentrated on the types of dialogues needed during team formation. Future work will include an investigation how protocols establishing binary social commitments during plan formation can be developed and analyzed in an interpreted multi-agent systems framework. Chopra and Singh have presented relevant work on commitment protocols, based on the formalism of transition systems (Chopra et al., 2006). Lomuscio and Sergot (Lomuscio et al., 2004) investigate the possibility of applying deontic logic in order to study agents’ violations of file transmission protocols. We have not yet investigated this issue for our protocols, but it is interesting future research. It is also interesting to design a logic exactly suited to communication protocols such as $MACOM_1$ and $MACOM_2$, in a similar fashion as the sound and complete system TDL developed by Lomuscio and Wóżna for authentication protocols (Lomuscio et al., 2006). For such a system with a computationally grounded semantics of interpreted systems, it may even be possible to develop model checking techniques in order to check relevant properties automatically.
Acknowledgements

We would like to thank several anonymous referees for their helpful comments on this work. Furthermore, we would like to thank Hans van Ditmarsch, Jan van Eijck and Philippe Balbiani for organizing the wonderful workshop Logics for Information Security.

10. References


