Adjusting a Knowledge-Based Algorithm for Multi-agent Communication for CPS

E. van Baars and R. Verbrugge

Ordina Vertis B.V., Kadijk 1, 9747 AT Groningen, The Netherlands
egon@vanbaars.com
Department of Artificial Intelligence, University of Groningen,
P.O. Box 407, 9700 AK Groningen, The Netherlands
L.C.Verbrugge@rug.nl

Abstract. Using a knowledge-based approach we adjust a knowledge-based algorithm for multi-agent communication for the process of cooperative problem solving (CPS). The knowledge-based algorithm for multi-agent communication [1] solves the sequence transmission problem from one agent to a group of agents, but does not fully comply with the dialogue communication. The number of messages being communicated during one-on-one communication between the initiator and each other agent from the group can differ. Furthermore the CPS process can require the communication algorithm to handle changes of initiator. We show the adjustments that have to be made to the knowledge-based algorithm for multi-agent communication for it to handle these properties of CPS. For the adjustments of this new multi-agent communication algorithm it is shown that the gaining of knowledge required for a successful CPS process is still guaranteed.

1 Introduction

For cooperative problem solving (CPS) within multi-agent systems, Wooldridge and Jennings give a model of four consecutive stages [2]. Dignum, Dunin-Kęplicz and Verbrugge present a more in-depth analysis of the communication and dialogues that play a role during these four stages [3,4]. 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 CPS, the agents have to achieve an approximation of common knowledge through communication. This makes reliable knowledge-based communication essential for teamwork. 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. To be reliable, the connection has to satisfy the *fairness* condition, leaving the protocol responsible for the *liveness* and *safety* properties [5,6]. Besides the *liveness* and *safety* properties a protocol used in teamwork has to satisfy the requirements of CPS.

Van Baars and Verbrugge [1] derive a knowledge-based algorithm for multi-agent communication. The algorithm ensures the *liveness* and *safety* property,

M. Dastani et al. (Eds.): LADS 2007, LNAI 5118, pp. 89–105, 2008.

but does not satisfy the other requirements of CPS [1]. This algorithm solves the *sequence transmission* problem [5] from one agent to a group of agents. However, the communication during CPS is not a one-way transport of data as in the *sequence transmission* problem, but a dialogue. The next message is not predefined, but depends on the answers from the receivers [4]. A CPS process 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.

One of the properties of CPS 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 a CPS process is that the initiator can change. At the different stages of CPS the initiator has to have different abilities [2,4]. If an agent has all the required abilities, then it can fulfill the role of initiator throughout the whole process of CPS. 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 agent during one-on-one communication is the only agent who has sufficient group knowledge to start communicating one-on-group [3]. 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 knowledge-based algorithm from [1] cannot handle asynchronous communication, because it uses only one index. Introducing a separate index for each sender-receiver pair solves this problem, 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 the algorithm from [1] can be modified

[1] A simulation of the protocol can be found at `www.ai.rug.nl/alice/mas/macom`

to handle the asynchronous communication property and the changing initiator property. The modified multi-agent communication algorithm guarantees a stream of accumulating messages during a CPS process, meeting the requirements of CPS concerning the gaining of group knowledge.

As to the methodology of this research, we use *knowledge-based protocols* based on epistemic logic and analyse them using the formalism of *interpreted multi-agent systems*. 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 global states (tuples of local states of agents plus environment) through discrete time. This knowledge-based approach to communication protocols has been pioneered in the work of Halpern and colleagues [5,7]. Because of the use of epistemic logic, it fits very well to multi-agent systems based on BDI architectures. Nowadays in theoretical computer science, another formalism, that of strand spaces, has become very popular, especially in the context of security protocols. Halpern and Pucella have shown that strand spaces can be translated into interpreted multi-agent systems, but not vice versa, because strand spaces are less expressive: some interesting interpreted multi-agent systems cannot be expressed as a strand space [8].

The rest of the paper is structured as follows. Section 2 and section 3 present the problems that arise in the flexible context of teamwork and their possible solutions, while section 4 gives the new knowledge-based algorithm incorporating the feasible solutions. Section 5 presents a proof that approximations of common knowledge are indeed attained. Finally, section 6 closes the paper with some conclusions and ideas about further research.

2 Adjusting the Algorithm for 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 S1 or R1 respectively. The agents exchange messages and the arrow `->` 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 `'_'` 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. S3 (100,_,data)-> R1
2. S3 <-(100,_,answ) R1
3. S3 (101,_,data)-> R1
4. S3 <-(101,_,answ) R1
5. S3 (102,_,data)-> R1
6. S3 <-(102,_,answ) R1
```

This moves the index for the next message to be sent to R_1 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?

3 Adjusting the Algorithm 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 tot 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. R1 <-(104,G,data) S3 (203,G,data)-> R2
2. R1 (104,_,answ)-> S3 <-(203,_,answ) R2
3.                   R3 <-(203,_,data) S2
4.                   R3 (204,_,answ)-> S2
5.                   R3 <-(204,_,data) S2
```

Option 3, S_2 uses the next index number:

```
1. R1 <-(104,G,data) S3 (203,G,data)-> R2
2. R1 (104,_,answ)-> S3 <-(203,_,answ) R2
3.                   R3 <-(204,_,data) S2
4.                   R3 (204,_,answ)-> S2
5.                   R3 <-(205,_,data) S2
```

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 at the second time agent 2 sends a message with the same index. 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 [9]. If R_2 sent data instead of just an acknowledgement to agent 3 in the first message, then agent 2 cannot send another message with the same index number. When agent 3 answers with just an acknowledgement, agent 2 does not know whether agent 3 acknowledged the first or the second message. 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.

3.1 Two-Index Mechanism to the Rescue

How can these problems be solved? The transmission control problem (TCP) makes use of two indices per connection [9,10]. 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. R1  <-(100,_,_,data)  S3
2. R1 (200,100,_,answ)-> S3
3. R1 <-(101,200,_,data) S3 (300,_,_,data) --> R2
4. R1 (201,101,_,answ)-> S3 <-(400,300,_,answ) R2
5. R1 <-(102,201,G,data) S3 (301,400,G,data)-> R2
6. R1 (202,102,_,answ)-> S3 <-(401,301,_,answ) R2
```

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. R1 <-(102,201,G,data) S3 (301,400,G,data)-> R2
2. R1 (202,102,_,answ)-> S3 <-(401,301,_,answ) R2
3.                       R3 <-(401,301,_,data) S2
4.                       R3 (302,401,_,answ)-> S2
5.                       R3 <-(402,302,_,data) S2
```

Option 2, S_2 uses the next index number:

```
1. R1 <-(102,201,G,data) S3 (301,400,G,data)-> R2
2. R1 (202,102,_,answ)-> S3 <-(401,301,_,answ) R2
3.                       R3 <-(402,301,_,data) S2
4.                       R3 (302,402,_,answ)-> S2
5.                       R3 <-(403,302,_,data) S2
```

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.

3.2 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 section 3.1, 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 CPS 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. R1 <-(102,201,G,data) S3 (301,400,G,data)-> R2
2. R1 (202,102,_,init)-> S3 <-(401,301,_,init) R2
3.                       S3 (302,401,_,answ)-> R2
4.                       R3 <-(402,302,_,data) S2  (500,_,_,data)->  R1
5.                       R3 (303,402,_,answ)-> S2 <-(600,500,_,answ) R1
6.                       R3 <-(403,303,_,data) S2 (501,600,_,data)-> R1
```

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

```
1. R1 <-(102,201,G,data) S3 (301,400,G,data)-> R2
2. R1 (202,102,_,init)-> S3 <-(401,301,_,init) R2
3. R1 <-(103,202,_,data) S3 (302,401,_,data)-> R2
4. R1 (203,103,_,answ)-> S3 <-(402,302,_,answ) R2
5. R1 <-(104,203,_,data) S3 (303,402,_,data)-> R2
6. R1 (204,104,_,answ)-> S3 <-(403,303,_,answ) R2
```

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 CPS process.

4 CPS Specific Algorithm

In sections 2 and 3 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 algorithm from [1] have the following form:

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

The fields filled with "$-$" are the *checksum* and *window_size* fields, dealing with package mutation errors and congestion control [10,11]. As discussed in section 3, an algorithm for CPS needs an index mechanism consisting of two indices. The *window_size* is used for the sliding window [9] 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 CPS algorithm has the following form:

$$K_{source}(destination, -, group, sequence, acknowledgement, data)$$

Here follows a description of the fields in the messages used in the CPS algorithm.

source = source port where this message is sent from $[S, R_i]$;
Ksource = 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 CPS algorithm:

Acknowledgement	
ack_Ri	: Used by S. Acknowledged sequence number received from Ri
seqSRi	: Used by S. Sequence number of messages S is sending to Ri
seqS	: Used by Ri. Sequence number of messages Ri is receiving from S
seqRi	: Used by Ri. Sequence number of messages Ri is sending to S
seqRi	: Used by S. Sequence number of messages S is receiving from Ri
Data	
compose()	: Used by S and Ri. Agent makes up the data it wants to send

4.1 CPS Algorithm

The algorithm 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 **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, −, −, seqR_i, seqSR_i, 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, −, −, seqR_i, seqSR_i, data)
            {... store that you know that Ri knows it.}
9        end
         {[6] ... 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(−, −, G, −, −, 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(−, −, G, seqR_i + 1, SeqSR_i, data) do
            {... check if package 'seqSRi' has not been acknowledged yet by Ri, ... [13]}
12             send K_S(R_i, −, G, seqSR_i, seqR_i, 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
      {[9] ... all agents Ri have acknowledged package with sequence number seqSRi.}
```

16 **for (i = 1 to n) do**
{*For all receiving agents, ... [19]*}
17 **seqR$_i$ = seqR$_i$ + 1**
{*Sequence number of next message from R_i is known. Increment $seqR_i$.*}
18 **seqSR$_i$ = seqSR$_i$ + 1**
{*Increment own sequence number for R_i.*}
19 **end**
{*[16] ... Sequence numbers for and from R_i updated.*}
20 **end**
{*[6].*}

Receiver (incoming packages)

1 **while true do**
{*Get ready for receiving sequence of messages, ... [5]*}
2 **when received** $K_S(R_i, G, -, seqS, seqR_i, data)$ **do**
{*You have received a package (from S). Prepare for processing, ... [4]*}
3 **store** $K_{R_i} K_S(-, -, G, seqS, seqR_i, data)$
{*Store the received package.*}
4 **end**
{*[2] ... finished processing incoming package.*}
5 **end**
{*[1].*}

Receiver (outgoing packages)

1 **when** $K_{R_i} K_S(R_i, -, G, x, \varnothing, data)$
{*The first message is received.*}
2 **seqS = x**
{*The first sequence number from S is x.*}
3 **seqR$_i$ = y**
{*Initiate own sequence number at y.*}
4 **while true do**
{*Get ready to acknowledge incoming packages, ... [11]*}
5 **compose(data)**
{*Make up the data for this message. (Possibly a request to act as initiator.)*}
6 **while not** $K_{R_i} K_S(Ri, -, G, seqS + 1, seqR_i, data)$ **do**
{*Still not received package with 'seqS+1' (and 'seqR$_i$'), ... [8]*}
7 **send** $K_{R_i}(S, -, -, seqR_i, seqS, data)$
{*... (re)send data package.*}
8 **end**
{*[6] ... You've received message seqS+1 wiht acknowledgement seqRi*}
9 **seqS = seqS+1**
{*You know the sequence number of the next message. Increment seqS.*}
10 **seqR$_i$ = seqR$_i$+1**
{*Increment own sequence number, $seqR_i$.*}
11 **end**
{*[4].*}

5 Analysis of Epistemic Properties of the Algorithm

For the adjustments discussed in section 2 and 3, we showed informally that they ensure the required knowledge gaining for CPS. In this section we prove that if the adjusted algorithm is used during CPS communication, then the agents achieve an approximation of general knowledge.

5.1 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 behaviour of processors over time (see [7]). 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" φ if in every other global state having the same local state as processor R, φ holds. In particular, each processor knows its own local state; for the environment, there is no accessibility relation. The knowledge relations are equivalence relations, obeying the well-known epistemic logic $S5_n^C$ (see [7]), including the knowledge axiom $K_i\varphi \Rightarrow \varphi, i = 1, ..., n$, as well as axioms governing general and common knowledge such as $E_G\varphi \Leftrightarrow \bigwedge_{i\in G} K_i\varphi$ and $C_G\varphi \Rightarrow E_G(\varphi \wedge C_G\varphi)$. We use abbreviations for general knowledge at any finite depth. Inductively, $E_G^1\varphi$ stands for $E_G\varphi$ and $E_G^{k+1}\varphi$ is $E_G\varphi\left(E_G^k\varphi\right)$.

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. Time clearly obeys the axioms of the basic temporal logic K_t (see [12]), in which the following principle (A) is derivable:

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

To further model time, we extend $S5_n^C$ with the following mixed axiom:

KT1. $K_i\Box\varphi \rightarrow \Box K_i\varphi, i = 1, ..., n$

This axiom holds for systems with perfect recall [13]. Halpern et al. [13] present a complete axiomatization for knowledge and time, however in this article we only need the axiom KT1.

As for notation, global states are represented as (r, m) (m-th time-point in run r) in the interpreted system $\mathcal{I}$. In particular for the temporal operators, we have the following truth definitions:

$(\mathcal{I}, r, m) \models \Box\varphi$ *iff* $(\mathcal{I}, r, m') \models \varphi$ *for all* $m' \geq m$
$(\mathcal{I}, r, m) \models P\varphi$ *iff* $(\mathcal{I}, r, m') \models \varphi$ *for some* $m' < m$

5.2 Proof of the Increase of Group Knowledge

For the readability of the proof, the form of the package is shortened to $Ksource(sequence, data)$. We assume that the group stays unchanged and we

assume 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.

Formulas	Descriptions
$K_{R_i}(p, \alpha)$	Receiver i knows that the p-th data segment is α; similar for $K_S(p, \alpha)$
$K_{R_i}(p, -)$	Receiver i knows the value of the p-th data segment; similar for $K_S(p, -)$
$E_G(p, \alpha)$	Every agent in group G knows that the p-th data segment is α
$E_G(p, -)$	Every agent in group G knows the value of the p-th data segment
$E_G^k \varphi$	Group G has depth k general knowledge of φ
R_G	G is the current group of receivers
$P\varphi$	At some moment in the past on this run, φ was true
$\Box\varphi$	φ is now true and will always be true on this run

Theorem 1. *Let $\mathcal{R}$ be any set of runs consistent with the knowledge-based algorithm from section 4 where:*

- *the environment allows for deletion and reordering errors, but no other kinds of error;*
- *The safety property holds (so that at any moment the sequence Y of data elements received by each R_i is a prefix of the infinite sequence X of data elements on S's input tape).*

Then for all runs in $\mathcal{R}$ and all $k \geq 0, j \geq 0$ the following hold:

[Forth]: R_i *stores* $K_{R_i} K_S (j+k, \alpha) \rightarrow \Box K_{R_i} K_S (E_G K_S)^k (j, \alpha)$.
[Back_i]: S *stores* $K_S K_{R_i} (j+k, -) \rightarrow \Box K_S K_{R_i} K_S (E_G K_S)^k (j, -)$.
[Back_G]: S *stores* $K_S E_G (j+k, -) \rightarrow \Box K_S (E_G K_S)^{k+1} (j, -)$.

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 knowledge-based algorithm, S and R_i store all relevant information from the packages that they receive. Moreover, packages that are sent have the following form: $K_{R_i}\varphi$ or $K_S\varphi$, from which 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.

C Under the same assumption of $\mathcal{R}$ being consistent with the knowledge-based algorithm, 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 $\mathbf{k = 0}$. From B follows the **Forth**-part for $(k = 0)$ namely

$$R_i \text{ stores } K_{R_i} K_S (j, \alpha) \rightarrow \Box K_{R_i} K_S (j, \alpha) . \tag{1}$$

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

$$\begin{gathered} \text{if } R_i \text{ sends } K_{R_i} (j, -) \text{ then } P (R_i \text{ stores } K_S (j, \alpha)) , \\ \text{so } P \Box K_{R_i} K_S (j, \alpha) , \text{ and } \Box K_{R_i} K_S (j, \alpha) . \end{gathered} \tag{2}$$

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.

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

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

$$K_S P (\Box K_{R_i} K_S (j, -)) , \text{ and } K_S \Box K_{R_i} K_S (j, -) , \text{ so } \Box K_S K_{R_i} K_S (j, -) . \tag{4}$$

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 R_G where $G = \{1, ..., n\}$ then S knows that $R_{i<i=1..n>}$ have sent these acknowledgements in the past.

$$\text{If } S \text{ stores } K_S E_G (j, -) \text{ then } K_S P (R_{i<i=1..n>} \text{ sends } K_{R_i} (j, -)) ... \tag{5}$$

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

$$K_S P (\Box E_G K_S (j, -)) , \text{ and } K_S \Box E_G K_S (j, -) , \text{ so } \Box K_S E_G K_S (j, -) . \tag{6}$$

If (5) and (6) are combined, then we have the **Back_G**-part of the theorem for the j-th data segment ($k = 0$). What knowledge about the j-th data segment emerges for $k \neq 0$? This is shown in the induction step.

Induction step. Suppose as induction hypothesis that **Back_i**, **Back_G** and **Forth** are valid for $k - 1$, with $k \geq 1$. Now a proof follows that **Forth**, **Back_i**, and **Back_G** are also valid for k.

[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 $(j + (k - 1))$:

$$S \text{ sends } K_S (j + k, \alpha) \rightarrow P (S \text{ stores } K_S E_G (j + (k - 1) , -)) . \tag{7}$$

With the **Back_G**-part of the theorem for $k - 1$ and A, the following can be deduced:

$$S \text{ sends } K_S (j + k, \alpha) \rightarrow \Box K_S (E_G K_S)^k (j, -) . \tag{8}$$

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:

$$R_i \text{ stores } K_{R_i}K_S\left(j+k,\alpha\right) \rightarrow \Box K_{R_i}K_S\left(E_GK_S\right)^k\left(j,-\right). \tag{9}$$

This is exactly what the **Forth**-part of the theorem says.

[Back_i]: R_i only sends an acknowledgement for the $(j+k)$-th data element if he stored $K_{R_i}K_S\left(j+k,-\right)$ in the past. With A, now the following can be derived:

$$R_i \text{ sends } K_{R_i}\left(j+k,-\right) \rightarrow \Box K_{R_i}K_S\left(E_GK_S\right)^k\left(j,-\right). \tag{10}$$

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:

$$S \text{ stores } K_SK_{R_i}\left(j+k,-\right) \rightarrow \Box K_SK_{R_i}K_S\left(E_GK_S\right)^k\left(j,-\right), \tag{11}$$

and this is exactly the **Back_i**-part of the theorem.

[Back_G]: S receives acknowledgements from all R_i. At a certain time S has received an acknowledgement for the $(j+k)$-th data segment from all R_i. Thus,

$$S \text{ stores } K_SE_G\left(j+k,-\right).$$

With A and B it can now be concluded that:

$$S \text{ stores } K_SE_G\left(j+k,-\right) \rightarrow \Box K_S\left(E_GK_S\right)^{k+1}\left(j,-\right), \tag{12}$$

and this is exactly the **Back_G**-part of the theorem.

6 Conclusion and Future Work

This research falls in the tradition of using interpreted multi-agent systems to analyze communication protocols, and extends knowledge-based analysis of file transmission protocols such as [5,10,14]. Our aim has been to make communication protocols much more flexible than file transmission protocols, in order to adapt them to dialogue-based cooperative problem solving (CPS). There, more interactive inter-group communication is needed than can be achieved by simple broadcasts from an initiator to the rest of his team. In this paper a knowledge-based algorithm for multi-agent communication [1] is adjusted for dialogue communication in teamwork. It is shown how the protocol handles the different numbers of messages between the initiator and different members and the changing initiator property, guaranteeing the knowledge gain required for CPS. An algorithm supporting the dynamic properties of CPS communication provides a flexible approach for CPS.

This research complements other literature that aims to make Wooldridge's and Jennings' CPS model [2] more flexible, for example, [15] where the needed

group attitudes for teamwork are adjusted to properties of the environment and the organization. Durfee et al. present another model of CPS [16]. 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 CPS 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 CPS.

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 [17]. Lomuscio and Sergot [14] 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 the one-to-many protocol from [1] and the CPS adjusted algorithm given here, in a similar fashion as the sound and complete system TDL developed by Lomuscio and Woźna for authentication protocols [18]. 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 three anonymous referees for their helpful comments.

References

1. van Baars, E., Verbrugge, R.: Knowledge-based algorithm for multi-agent communication. In: Bonanno, G., et al. (eds.) Proceedings of the 7th Conference on Logic and the Foundations of Game and Decision Theory, University of Liverpool, pp. 227–236 (2006)
2. Wooldridge, M., Jennings, N.R.: The cooperative problem-solving process. Journal of Logic and Computation 9(4), 563–592 (1999)
3. Dignum, F., Dunin-Kęplicz, B., Verbrugge, R.: Creating collective intention through dialogue. Logic Journal of the IGPL 9(2), 289–303 (2001)
4. Dunin-Kęplicz, B., Verbrugge, R.: Dialogue in teamwork. In: Fonseca, J.M., et al. (eds.) Proceedings of The 10th ISPE International Conference on Concurrent Engineering: Research and Applications, Rotterdam, A.A. Balkema, pp. 121–128 (2003)
5. Halpern, J.Y., Zuck, L.D.: A little knowledge goes a long way: Simple knowledge-based derivations and correctness proofs for a family of protocols. In: Proceedings of the 6th ACM Symposium on Principles of Distributed Computing, pp. 269–280 (1987); Full version including proofs appeared in Journal of the ACM 39(3), 449–478 (1992)

6. van Baars, E.: Knowledge-based algorithm for multi-agent communication. Master's thesis, Department of Artificial Intelligence, University of Groningen (2006), www.ai.rug.nl/alice/mas/macom
7. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.: Reasoning About Knowledge. MIT Press, Cambridge (1995)
8. Halpern, J.Y., Pucella, R.: On the relationship between strand spaces and multi-agent systems. ACM Trans. Inf. Syst. Secur. 6(1), 43–70 (2003)
9. Postel, J.: Transmission control protocol (TCP). Technical Report RFC 793, Internet Society (September 1981), ftp://ftp.rfc-editor.org/in-notes/rfc793.txt
10. Stulp, F., Verbrugge, R.: A knowledge-based algorithm for the internet protocol TCP. Bulletin of Economic Research 54(1), 69–94 (2002)
11. Douglas, D.E.: Internetworking with TCP/IP. Principles, Protocols and Architectures, vol. 1. Pearson Prentice Hall, Upper Saddle River (2006)
12. Goldblatt, R.: Logics of Time and Computation. CSLI Lecture Notes, vol. 7. Center for Studies in Language and Information, Palo Alto (1992)
13. Halpern, J., van der Meyden, R., Vardi, M.: Complete axiomatizations for reasoning about knowledge and time. SIAM Journal on Computing 33(3), 674–703 (2004)
14. Lomuscio, A., Sergot, M.: A formulation of violation, error recovery, and enforcement in the bit transmission problem. Journal of Applied Logic 2, 93–116 (2004)
15. Dunin-Kęplicz, B., Verbrugge, R.: A tuning machine for cooperative problem solving. Fundamenta Informaticae 63, 283–307 (2004)
16. Cox, J.S., Durfee, E.H., Bartold, T.: A distributed framework for solving the multi-agent plan coordination problem. In: Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M. (eds.) AAMAS, pp. 821–827. ACM, New York (2005)
17. Chopra, A.K., Singh, M.P.: Contextualizing commitment protocols. In: Nakashima, H., Wellman, M.P., Weiss, G., Stone, P. (eds.) AAMAS, pp. 1345–1352. ACM, New York (2006)
18. Lomuscio, A., Woźna, B.: A complete and decidable security-specialised logic and its application to the TESLA protocol. In: Stone, P., Weiss, G. (eds.) Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 145–152. ACM Press, New York (2006)