Knowledge-based Algorithm for Multi-agent Communication {Master Thesis}

E. van Baars, egon@vanbaars.com
Student number: s09811254

August 29, 2006

Internal advisor: dr. R. Verbrugge
[Artificial Intelligence, University of Groningen]
Referee and external advisor: ir. S. Achterop
[Computing Science, University of Groningen]
External advisor: H. Paas
[Computing Science, University of Groningen]

Institute of Artificial Intelligence,
University of Groningen,
Grote Kruisstraat 2/1, 9712 TS Groningen, The Netherlands

Knowledge-based Algorithm for Multi-agent Communication {Master Thesis}

E. van Baars, egon@vanbaars.com
Student number: s0981125

August 29, 2006

Internal advisor: dr. R. Verbrugge
[Artificial Intelligence, University of Groningen]
Referee and external advisor: ir. S. Achterop
[Computing Science, University of Groningen]
External advisor: H. Paas
[Computing Science, University of Groningen]

Institute of Artificial Intelligence,
University of Groningen,
Grote Kruisstraat 2/1, 9712 TS Groningen, The Netherlands

Abstract

Software agents are more and more used to perform autonomous tasks in real world environments. This can be a single agent which is performing a task or a group of agents cooperating to perform a task. A system that consists of more than one agent is called a Multi-agent System (MAS). Multi-agent Systems is a research field within the research domain of Artificial Intelligence. One of the processes studied within MAS is cooperative problem solving (CPS). This process describes the problem of one agent that sees a goal for which it needs other agents to cooperate with to achieve this goal. Communication is essential during CPS processes because successful cooperative problem solving requires reliable one-on-group communication to attain an approximation of common belief among the the agents cooperating in a team. This will be shown during a brief introduction of cooperative problem solving. To attain an approximation of common belief it is necessary that the agents gain a certain level of group knowledge about the facts communicated. More specifically, the agents have to know the facts that are communicated, the agents have to know to whom those facts are communicated, and the agents have to know that the agents to whom these facts are communicated know these facts.

A simple problem is the sequence transmission problem where one agent communicates a sequence of data to another agent while both agents gain a certain level of knowledge about this data. The sequence transmission problem becomes more complicated when one agent wants to communicate a sequence of data to a group of agents. To attain the desired level of knowledge gaining, somehow the group information has to be involved in the communication. In this thesis a general knowledge-based algorithm is presented that solves the sequence transmission problem for one-on-group communication. This general knowledge-based algorithm is correct for communication media where typical communication errors occur as long as the connection satisfies the fairness condition. It is shown that the agents from a group when using this algorithm for n cycles, gain depth n of general knowledge about the members of the group and about the facts communicated.

The communication involved in the CPS process is a bit more complicated than the sequence transmission problem. The sequence transmission problem concerns the transport of data from one agent to one or more other agents while the CPS communication is more a dialogue between two or more agents. The requirements of reliability and gaining of knowledge are the same for both communication processes. A specific knowledge-based algorithm for CPS communication is presented. This CPS algorithm is a modified version of the general algorithm adjusted for the specific demands of CPS communication. It is shown that the agents from a group when using this algorithm for n cycles during

a CPS process, gain depth n of general knowledge about the members of the group and about the facts communicated.

In general, software agents are connected to each other in a network. The most common network architecture for computers is the internet network architecture. For the CPS algorithm to be of use for software agents in a MAS environment involved in CPS processes, the CPS algorithm should be implemented somewhere in the internet architecture. In this thesis the feasibility of such an implementation is discussed. As a result of this discussion a design specification for an implementation is presented.

Contents

Chapter 1

Introduction

Today's society becomes more and more information driven. This growing amount of information produces complex problems. Autonomous software agents can be used to solve these problems or to act as a supportive system to help humans solving these problems. Depending on the problem domain, this can be one software agent performing a task by itself or it can be a group of software agents cooperating in performing a task. Reasons for agents to cooperate can be that the problem is geographically distributed. This means that agents are needed at different locations to perform subtasks while cooperating to achieve the main task. Another reason for agents to cooperate can be that the complexity of a problem requires agents with different skills to solve this problem. A group of cooperating autonomous agents is called a Multi-agent System, henceforth MAS. The study of MAS is a research field within the research domain of Artificial Intelligence.

One of the research fields within MAS is Cooperative Problem Solving, henceforth CPS. In this field research is done about the process of agents cooperating to solve a certain goal. Wooldridge and Jennings give a model for CPS consisting of four consecutive stages [21]. These four stages are *potential recognition, team formation, plan formation* and *team action*. The CPS process covers the entire process that starts with one initiating agent that sees an overall goal which it can not or does not want to achieve by itself, and ends with a group of agents cooperating in achieving this goal. Communication is essential for CPS and Dignum, Dunin-Kęplicz and Verbrugge give a more in-depth analysis of the dialogues that play a role during the four stages of CPS in [1, 4]. Most writings about the communication involved with CPS make one and the same assumption. This assumption is that whatever is communicated between agents will arrive at the desired agents and will arrive correctly. This is not as obvious as it seems. During communication, errors can occur such as messages that get lost or mutated. Possible communication errors should be overcome by a communication algorithm.

1.1 Group communication

For one-on-one communication, algorithms exist which guarantee a reliable knowledge-based communication. Examples of such an algorithm are presented by Halpern and Zuck in [11] and by Stulp and Verbrugge in [18]. These algorithms can handle the sequence transmission problem for one-on-one communication. The sequence transmission problem can be stated as two agents, a sender and a receiver, where the sender sends a sequence of data packages to the receiver. These data packages have to arrive at the receiver and have to arrive in the right order. During this process both agents gain a certain level of knowledge about the sent data, and gain a certain level of knowledge of each other's knowledge of the sent data.

In the second, team formation stage of CPS, the agents have to reach a collective intention. For this to happen communication between the initiating agent and a group of potential agents is needed. During this one-on-group communication a certain level of general group knowledge has to be attained. General group knowledge means that all the agents the initiating agent is communicating with have to gain a certain level of knowledge about the facts the initiator is sending, that all these agents have to gain knowledge about to whom the initiator is sending these facts, and that all the agents gain a certain level of knowledge about the knowledge of the agents from this group. The algorithms for one-on-one communication from [18, 11] are not sufficient for this communication because they do not provide a mechanism for attaining group knowledge. What is needed is a knowledge-based algorithm that guarantees a reliable communication for one-on-group communication involved in a CPS process.

1.2 Research questions

A first step towards an algorithm for one-on-group CPS communication is to extend a reliable one-on-one communication algorithm so that it can handle the sequence transmission problem for one-on-group communication. This brings us to the first research question (RQ).

RQ1. Is it possible to design a knowledge-based algorithm for multi-agent communication that can handle the one-on-group sequence transmission problem?

This research question is answered in this thesis by the presentation of a general knowledge-based algorithm for multi-agent communication. This algorithm will also be denoted as general algorithm.

The sequence transmission problem is a one-way data transport problem. The general knowledge-based algorithm handles this problem and as a result is a data transport algorithm. The communication involved in a CPS process is more a dialogue type of communication than it is a transport of data. In the sequence transmission problem and accordingly in the general algorithm, there is one and the same agent that acts as sender during the transmission process.

In a CPS process there is also one agent that acts as a sender but over time this role can be taken by different agents. Out of these differences of communication comes the second research question.

RQ2. Is it possible to design a knowledge-based algorithm for CPS communication?

The answer to this research question is a knowledge-based CPS algorithm for multi-agent communication that will be presented in this thesis. This algorithm will also be denoted as CPS algorithm. For the CPS algorithm to be of use for software agents operating in the real world, it should be implemented. In general, software agents are interconnected to each other in a computer network. The most common network architecture for computer networks is the internet architecture. The most useful way of implementing the CPS algorithm would thus be somewhere in the internet architecture. This brings us to the third and last research question.

RQ3. What are the possibilities of implementing the CPS algorithm in the internet architecture?

In this thesis the feasibility of such an implementation is discussed. The result of this discussion is a design specification for an implementation.

1.3 Structure of thesis

In the next chapter an overview is given of the process of CPS. The four stages will be discussed and when one-on-group instead of one-on-one communication is needed, this will be pointed out. Readers who are already familiar with CPS can skip this chapter. In chapter 3, knowledge and communication will be discussed. An overview is given of knowledge and knowledge creation within a group as can be found in CPS. The communication part of this chapter handles the errors that can occur during communication and how a communication protocol can overcome these errors. In chapter 4, the knowledge-based algorithm for MAS is presented together with an epistemic analysis and proof. chapter 5, discusses the specific communication problems involved with CPS. An adjusted algorithm for CPS which handles these problems is presented in chapter 6 together with an epistemic proof. In chapter 7, design specification are given for the implementation of the CPS algorithm. chapter 8, finally, presents a discussion of related literature and directions for future research. Appendix A, shows a graphical overview of the CPS process.

Chapter 2

CPS

The process of CPS starts with an initiating agent which sees an overall goal that it wants to be achieved. This agent can not or does not want to achieve this goal by itself so it has to look for a group of agents that can cooperate in achieving this goal. This process can be subdivided in four consecutive stages: *potential recognition, team formation, plan formation,* and *team action* [1]. Every stage of CPS starts with an initial situation and a goal. Achievement of this goal results in the outcome situation for that stage which on its turn is the initial situation for the next stage. The goals of the four different stages can be described formally by using multi-modal logic. In this chapter a description will follow of the initial situation, the goal and the corresponding outcome situation of the four stages of CPS based on [1, 4]. A graphical overview of a complete CPS process can be found in appendix A.

2.1 Logical background on CPS

In the descriptions of the four stages of CPS the terms collective intention and collective commitment are used. For an intention and a commitment to become collective among a group of agents there has to be a common belief among these agents. For a better understanding, brief descriptions of these concepts will follow. The description of common belief is based on [8] and the descriptions of collective intention and collective commitment are based on [1, 4, 3, 6].

Common belief is the notion of group belief which is constructed in a similar way as common knowledge, except that a collective belief among a group that ψ need not imply that ψ is true. Here follows a short reminder of the axioms governing group beliefs. Let $G \subseteq \{1, \ldots, m\}$ be a group of m agents. The formula E-BEL$_G(\psi)$ (group G has ψ as a *general belief*) is meant to stand for "every agent in group G believes ψ":

$$\text{E-BEL}_G(\psi) \leftrightarrow \bigwedge_{i \in G} \text{BEL}(i, \psi)$$

C-BEL$_G(\psi)$ (group G has ψ as a *common belief*) is meant to be true if everyone in G believes ψ, everyone in G believes that everyone in G believes ψ, etc.

Let E-BEL$^1_G(\psi)$ be an abbreviation for E-BEL$_G(\psi)$, and let E-BEL$^{n+1}_G(\psi)$ for $n \geq 1$ be an abbreviation of E-BEL$_G$(E-BEL$^n_G(\psi)$). Thus, we have intuitively C-BEL$_G(\psi)$ iff E-BEL$^n_G(\psi)$ for all $n \geq 1$.

A collective intention is the intention of a group G of agents to achieve some goal. Let $G \subseteq \{1, \ldots, m\}$ be a group of m agents. The formula E-INT$_G(\psi)$ (group G has ψ as a *general intention*) is meant to stand for "every agent in group G has the same individual intention ψ":

$$\text{E-INT}_G(\psi) \leftrightarrow \bigwedge_{i \in G} \text{INT}(i, \psi)$$

The mutual intention M-INT$_G(\psi)$ is meant to be true if every agent in G intends ψ, every agent in G intends that every agent in G intends ψ, etc. Denoting this in a formula it looks like E-INT$_G(\psi \wedge$ E-INT$_G(\psi \wedge ...))$. Infinite formulas are not allowed but intuitively the following recursive formula for mutual intention denotes the same: M-INT$_G(\psi) \leftrightarrow$ E-INT$_G(\psi \wedge$ M-INT$_G(\psi))$. For a formal deduction see [3]. For an intention to become collective there not only has to be a mutual intention to ψ, the agents from G also need to have a common belief about this mutual intention. This is denoted by the following axiom for collective intention: C-INT$_G(\psi) \leftrightarrow$ (M-INT$_G(\psi) \wedge$ C-BEL$_G$(M-INT$_G(\psi)$))

A strong collective commitment SC-COMM$_{G,P}(\psi)$ means that all the agents from group G collectively believe that the actions $< \delta_j, ...\delta_n >$ from social plan P are allocated to agents who have socially committed themselves to another agent to perform these actions. For a collective commitment there first has to be a collective intention C-INT$_G(\psi)$. Second there has to be a plan P which constitutes this goal ψ, *constitute*(ψ, P), i.e. if fully executed the plan leads to achievement of the goal. This plan P consists of actions $<\delta_1, \ldots, \delta_n>$ which should be executed by the agents from group G to achieve ψ. The agents also need to have a common belief that plan P constitutes the goal ψ. For every action from plan P there has to be an agent from G who committed itself to another agent to perform this action, $\bigwedge_{\delta \in P} \bigvee_{i,k \in G}$ COMM(i, k, δ). Again the agents also need to have a common belief about these commitments. When putting these parts together we get the following formula for a strong collective commitment:

$$\begin{aligned}\text{SC-COMM}_{G,P}(\psi) \leftrightarrow\ & \text{C-INT}_G(\psi) \wedge \textit{constitute}(\delta, P) \wedge \\ & \text{C-BEL}_G(\textit{constitute}(\psi, P)) \wedge \textstyle\bigwedge_{\delta \in P} \bigvee_{i,k \in G} \text{COMM}(i, k, \delta) \wedge \\ & \text{C-BEL}_G(\textstyle\bigwedge_{\delta \in P} \bigvee_{i,k \in G} \text{COMM}(i, k, \delta)).\end{aligned}$$

More varieties of collective commitments others then the strong collective commitment mentioned here can be found in [5].

An overview of the formulas just discussed and other also used in the rest of this chapter can be found in the next table.

Formulas	Description in text
$GOAL(a, \psi)$	ψ is a goal of agent a
$PotC(a, \psi)$	Agent a sees a potential for cooperation with respect to ψ
$\mathrm{INT}(i, \psi)$	Agent i has has the intention to achieve ψ
$\text{M-INT}_G(\psi)$	Group G has a mutual intention to achieve ψ
$\text{C-INT}_G(\psi)$	Group G has a collective intention to achieve ψ
$\mathrm{BEL}(i, \psi)$	Agent i believes that ψ
$\text{E-BEL}_G(\psi)$	Group G has a general belief that ψ
$\text{E-BEL}_G^{n+1}(\psi)$	$\text{E-BEL}_G(\text{E-BEL}_G^n(\psi))$
$\text{C-BEL}_G(\psi)$	Intuitively; $\text{E-BEL}_G^n(\psi)$, for all $n \geq 1$. Group G has a common belief that ψ
$realize(<\psi_1, ..., \psi_n>, \psi)$	Realisation of the subtasks $<\psi_1, ..., \psi_n>$ leads to the main goal ψ
$\mathrm{COMM}(i, k, \delta)$	Agent i socially committed itself to agent k to perform action δ
$\text{SC-COMM}_{G,P}(\psi)$	Group G has a strong collective commitment based on P to achieve ψ

2.2 Potential recognition

The initial situation of potential recognition is an initiating agent a that sees an overall goal ψ. This initial situation can be described formally as $\mathrm{GOAL}(a, \psi)$ which means that ψ is a goal of a [1]. Initiating agent a can not or does not want to achieve the overall goal ψ by itself so it needs to find other agents to cooperate with in achieving the overall goal ψ. Thus the goal of this stage is that agent a has to find one or more groups of agents that have the potential for cooperation with respect to ψ. To achieve this, agent a has to find out what the *abilities, opportunities, willingness* and *commitment strategies* with respect to ψ of all the agents are [1]. In the graphical representation this will be denoted as $\psi(a, o, w, c)$. To get this information from these agents, agent a communicates to them through one-on-one communication. The dialogue type involved in this communication is information seeking. For a graphical representation of this communication see figure 2.1. A successful outcome of this information seeking dialogue is that agent a sees one or more groups of agents that have the potential to cooperate in achieving the overall goal ψ. This can be formally denoted as $\mathrm{PotC}(a, \psi)$ which means that ψ is the goal of agent a, which can not or does not want to achieve ψ by itself, and there are one or more groups G such that agent a believes that each of these groups G can collectively achieve ψ.

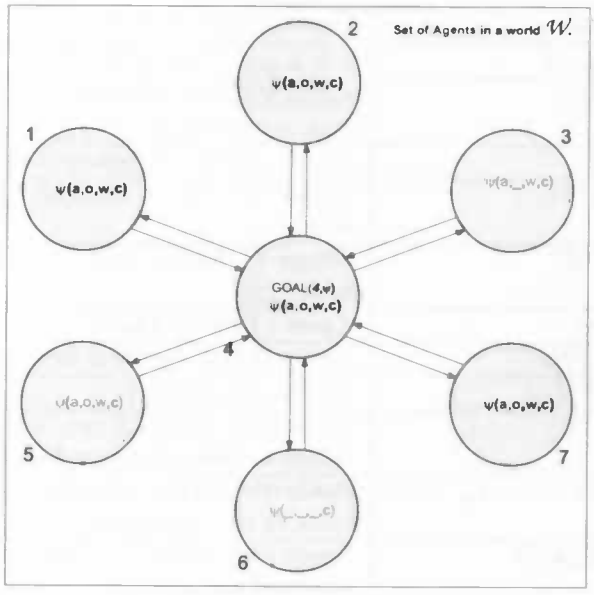

Figure 2.1: One-on-one information seeking dialogue communications between initiating agent 4 and the other agents

2.3 Team formation

The initial situation of the team formation level is PotC (a, ψ), which is the same as the outcome situation of the preceding stage. The goal of the team formation stage is that agent a finds one group G among all the potential groups which has a collective intention to achieve ψ. A formal notation for this collective intention is C-INT$_G$ (ψ) as defined in [3]. To establish this situation, agent a has to persuade all agents i of a potential group to take on an individual intention towards ψ, and to take on the intention that there be a mutual intention among that group, INT $(i, \psi \wedge$ M-INT$_G$ $(\psi))$. Agent a communicates during this persuasion dialogue one-on-one to each of the agents. A successful outcome of these dialogues is that all the members of one of the potential groups have taken on the intention INT $(i, \psi \wedge$ M-INT$_G$ $(\psi))$. We are now one step away from the outcome situation of this stage. This last step is that the individual intentions of all the agents becomes a collective intention. This collective intention C-INT$_G$ (ψ) emerges from the fact that it has become a common belief that all the agents in this group have taken on the intention INT $(i, \psi \wedge$ M-INT$_G$ $(\psi))$. To establish this situation agent a communicates the fact that all the members of this group have taken on this intention through one-on-group communication to this group. See figure 2.2. Because agent a has to communicate one-on-group, the earlier mentioned one-on-one communication algorithms are not sufficient. For this one-on-group communication, the algorithms that are presented in this thesis have been developed. If the one-on-group communication between agent

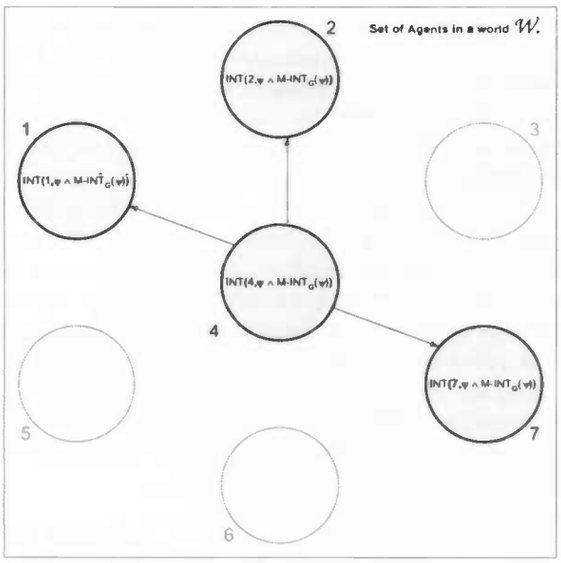

Figure 2.2: Initiating agent 4 communicates one-on-group the outcome situation to the other agents

a and all the members of this group is successful then the collective intention C-INT$_G$ (ψ) has been established and the outcome situation of the team formation stage has been reached.

2.4 Plan formation

The preceding stage ended with a group G together with the collective intention C-INT$_G$ (ψ), which is also the initial situation for the plan formation stage. In the plan formation stage, group G has to go from a collective intention to achieve ψ, to the goal of this stage which is a collective commitment based on a social plan P which will lead to the achievement of ψ. The formation of this plan starts with one of the agents from group G who takes on the role of initiator. This initiating agent doesn't have to be the same as in the previous stages and can thus be any agent from the group. This initiator will here be mentioned as agent b. Agent b generates a plan by an adequate task division of ψ into a sequence of subtasks $<\psi_1, \ldots, \psi_n>$. During this task division agent b communicates with the agents in the group through one-on-one communication. The dialogue type that is used during this communication is deliberation. The task division is successful if all the agents i of group G have taken on the belief that these subtasks $<\psi_1, \ldots, \psi_n>$ lead to the main goal ψ, BEL $(i, realize\,(<\psi_1, \ldots, \psi_n>, \psi))$. Agent b communicates then to all the agents of group G through one-on-group communication that each agent

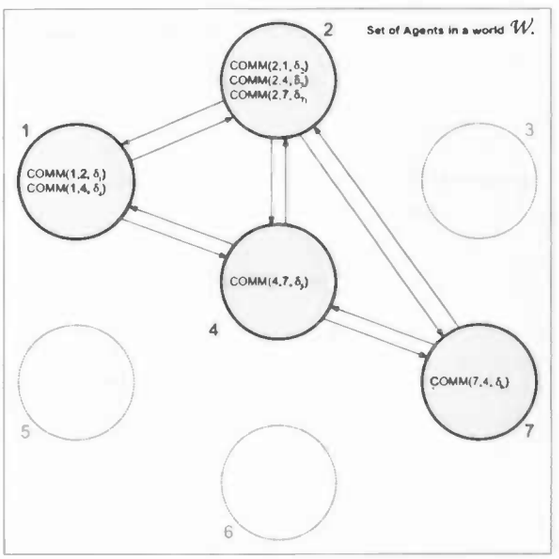

Figure 2.3: Agents communicating one-on-one during the action commitment process

of group G has taken on the belief BEL $(i, realize\,(<\psi_1, \ldots, \psi_n>, \psi))$. If this communication is successful then the common belief
C-BEL$_G$ $(realize\,(<\psi_1, \ldots, \psi_n>, \psi))$ has been established.

The next step in this plan generation process is that each subtask ψ_j is associated with one or more actions δ. For reasons of convenience it will be assumed that every subtask ψ_j is associated with only one action δ_j. For this process, one agent of the group takes on the role of initiating agent. Again this can be any of the agents from group G and will be denoted here as agent c. After each subtask ψ_j from $<\psi_1, \ldots, \psi_n>$ is associated with one action δ_j, every agent i from group G has to take on the belief that executing the actions $<\delta_1, \ldots, \delta_n>$ realizes the subtasks $<\psi_1, \ldots, \psi_n>$. This is formally notated as BEL$\big(i, \wedge_{j=1}^{n} means\text{-}for\,\big(\delta_j, \psi_j\big)\big)$ as defined in [4]. After this belief has been established, agent c communicates through one-on-group communication to all the agents of group G the fact that all the agents of group G have taken on this belief. Successful communication leads to the common belief C-BEL$_G$ $(\wedge_{i=1}^{n} means\text{-}for\,(\delta_i, \psi_i))$. Now that the overall goal ψ is divided into subtasks, and that these subtasks are associated with actions, these actions have to be allocated to the agents of group G.

Each action δ_j is allocated to one agent i from group G, resulting in pairs $< \delta_j, i >$. For every action that is allocated to one agent i, this agent socially commits itself to another agent k to perform this action, COMM (i, k, δ_j). See figure 2.3. After a successful establishment of COMM (i, k, δ_j) one of the agents i or k communicates this fact through one-on-one communication to the initiat-

ing and coordinating agent of the action allocation process. This initiating and coordinating agent can be any agent from group G and will be denoted here as agent d. At a certain point all the actions $<\delta_1, \ldots, \delta_n>$ will have been allocated to an agent with the corresponding commitment. If all these allocations have been communicated to agent d, then agent d communicates this fact through one-on-group communication to all agents of group G. This results in the situation that all the agents of group G collectively believe that all the actions $<\delta_1, \ldots, \delta_n>$ from plan P are allocated to an agent and that this agent socially committed itself to another agent to perform this action. This situation is formally known as a strong collective commitment SC-COMM$_{G,P}(\psi)$ as defined in [4]. This collective commitment was the goal of this stage and with it, this stage has reached its outcome situation. There are a lot of other kinds of collective commitments besides the strong collective commitment mentioned here [3]. For the understanding of the background of the algorithms presented in this thesis it is sufficient to speak only of the strong collective commitment.

2.5 Team action

The initial situation of this last stage of CPS is a collective commitment and in this particular case a strong collective commitment SC-COMM$_{G,P}(\psi)$. The agents who have socially committed themselves to perform an action δ_j can now execute this action. The process of agents executing their actions, is started by the same initiating agent d from the action allocation phase from the previous stage. The initiation of the action execution phase by agent d follows from its one-on-group communication of the fact that all actions $<\delta_1, \ldots, \delta_n>$ have been allocated. This is because the result of this communication is a collective commitment to the social plan P, which describes when the actions will have to be performed by which agent. During the action execution, communication takes place between the agents of group G. To end this stage successfully the following facts have to be communicated. When agent i from COMM(i, k, δ_j) has executed his action δ_j it communicates this to agent k. Agent i or agent k communicates the fact that action δ_j has been executed through one-on-one communication to the initiating agent d. As soon as agent d for all actions $<\delta_1, \ldots, \delta_n>$ has received the fact that they have been executed, agent d knows that the action execution phase has been successful. Agent d communicates this fact through one-on-group communication to all the agents of group G. Group G has now reached the outcome situation of the team action stage which means that they successfully completed the CPS process.

Chapter 3

Knowledge and communication

A knowledge-based algorithm for multi-agent communication is needed for the phases of CPS in which a collective intention, common belief, or a collective commitment is required. Collective intention, common belief, and collective commitment emerge after successful one-on-group communication as discussed in chapter 2. In this chapter a logical background of knowledge and common knowledge is given after which communication problems that arise in attaining (common) knowledge during communication are discussed. This chapter ends with a discussion on the creation of knowledge during one-on-group communication.

3.1 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 [8, 14]). 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 which has the same local state as processor R, the formula φ holds. In particular each processor knows its own local state; for the environment, there is no knowledge (or accessibility) relation. The knowledge relations are equivalence relations, obeying the well-known epistemic logic $S5_n^C$ (see [8]), including e.g. 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 also use abbreviations for referring to general knowledge at any finite depth. Inductively, $E_G^1\varphi$ stands for $E_G\varphi$ and $E_G^{k+1}\varphi$ is $E_G\varphi(E_G^k\varphi)$.

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. Time clearly obeys the axioms of the basic temporal logic K_t (see [9]), 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 axiom:
KT1. $K_i\Box\varphi \rightarrow \Box K_i\varphi, i = 1, ..., n$
This axiom holds for systems with perfect recall [10]. Halpern et al. present a complete axiomatization for knowledge and time [10], however in this thesis 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* $m' < m$
In the next table some formulas are given, together with their informal meanings that will be used in the rest of the thesis.

Formulas	**Descriptions**
$K_S\varphi$	Sender S knows φ
$K_{R_i}\varphi$	Receiver R_i knows φ
$E_G\varphi$	Every agent in group G knows φ (general knowledge)
$E_G^k\varphi$	Group G has depth k general knowledge of φ
$C_G\varphi$	It is common knowledge among group G that 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 and will always be true on this run

3.2 Gaining knowledge

What does it take to change the state of agent i from being ignorant about a fact φ to knowing φ, represented as $K_i\varphi$? To know φ an agent has to become aware of φ. How this happens depends on the sensory input mechanisms of the agents and the world they are operating in. An agent can become aware of a fact φ in a direct manner because one of its input mechanisms senses this fact, or an agent can become aware of a fact φ in a indirect manner because it can deduce this fact from other facts it already knows. The direct manner of attaining knowledge about a certain fact φ can be subdivided into an active and a passive process.

The active process is that an agent gathers a certain fact φ by itself from its environment and the passive process is that a certain fact φ is communicated to this agent by another agent. The active process of attaining knowledge is represented in the general algorithm from this thesis as an input tape storing a fact φ which will be read by an agent i. After reading this tape, agent i knows the fact φ stored on this tape, so $K_i\varphi$.

The passive process is represented by an agent i who knows a certain fact

φ and sends this fact φ to another agent r. Agent r knows the fact φ as soon as it receives the fact φ sent by agent i and thus $K_r\varphi$. This passive process of attaining knowledge is represented in the algorithm by the fact that agent r stores every fact φ it receives from agent i and writes every consecutive fact φ it has stored to an output tape. Agent i and agent r are respectively denoted as sender (S) and receiver (R) in the rest of the thesis. During the communication of facts from the sender to the receiver errors can occur. In the next section these errors and how a knowledge-based algorithm handles these errors will be discussed.

3.3 Transmission problems

Communication between two or more agents consists of transmitting messages via a medium. There are a lot of different kinds of media over which communication can take place. Which kind of medium is used depends mostly on the domain the agent is operating in. Via a medium, connections between agents can be established over which communication takes place. The point of interest here is the reliability of this communication. Reliability here means that when sender S sends a sequence of messages to receiver R, that receiver R receives this sequence of messages fully, without any changes, and in the right order. Halpern and Zuck called this problem the *sequence transmission* problem [11]. To overcome this problem a communication system has to satisfy the following three properties: *fairness*, *liveness*, and *safety*. The *fairness* property means that when a message is sent by S to R infinitely many times, this message arrives at least one time at R. The *liveness* property means that every message that is received by R is written by R on its output tape. The *safety* property states that every message written by R on its output tape is a prefix of the messages from the input tape. During the transmission of messages over a connection, errors can occur which jeopardize these properties for reliable communication. The protocol by which the communication takes place has to overcome these errors to assure reliable communication. The errors that can occur during communication between a sender S and a receiver R are the following.

1. **mutation**: S sends a message to R which is received in a different form by R;
2. **deletion**: a message is sent by S but is never received by R;
3. **duplication**: S sends a message to R and R receives this message more than one time;
4. **reordering**: S sends a sequence of messages which are received by R in a different order;
5. **delay**: S sends a message to R but this message arrives late at receiver R;
6. **insertion**: R receives a message from S which was never sent by S.

Not all these errors occur during every communication. It depends on the connection that is used to communicate which errors can occur. The combination of the protocol by which the agents are communicating and the connection that is used to communicate can be seen as a *communication system* (figure 3.1). This combination determines whether the communication is reliable. Reliability here means that the communication system satisfies the *fairness*, *liveness*, and *safety* properties. This brings us to the following definition.

Definition 1 ***Communication system.*** *A communication system consists of a connection in a communication medium between one or more agents which are communicating to each other according to a protocol that is known by all the participating agents. The combination of the properties of the connection and the features of the protocol determine the reliability of the communication system.*

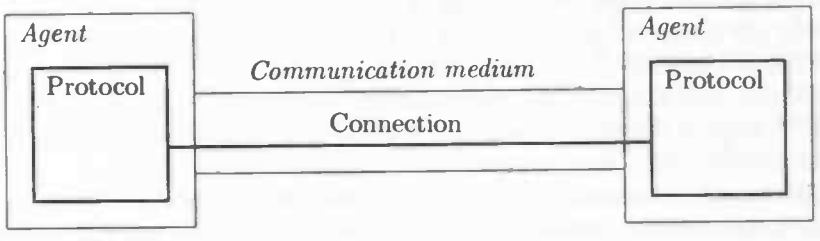

Figure 3.1: Communication system

Regardless which errors can occur in a system, if the connection over which the communication takes place does not satisfy the *fairness* condition then reliable communication is not possible. In this case it could happen that none of the messages sent by S arrive at R which means that there is no communication at all. So for the discussion on how the protocols in this thesis handle the above errors it is assumed that the *fairness* property holds for the connection over which the communication takes place. The protocol is responsible for assuring the *liveness* and *safety* properties by handling the errors that can occur at such a connection.

The distinctions between the different errors are not very strict. Dependent on the protocol and the connection, two different errors may appear to be one and the same problem for the protocol. Nevertheless, it will be shown how a knowledge-based protocol handles each of the above-mentioned errors should they be present in the used connection.

Mutation. If mutation errors can occur then the problem for the receiver is: how does it know whether the message it received is not mutated? The solution for this is the checksum as used in the TCP [17]. To handle mutation errors the TCP uses the 16-bit one's complement sum of the message as its checksum [2]. Because the checksum is a digital feature, the assumption that has to be made about the message and the way

this message is sent over the connection is that they have to be digital as well. The principle behind this mechanism is that the sender computes the checksum and sends it with the message to the receiver. When the receiver receives this message, it also computes the checksum of the message and compares it with the checksum sent by the sender. If these two checksums are the same, then the receiver knows that the message is not mutated. If the checksums are not the same, the receiver throws this message away and handles it as if it was not received at all.

The checksum from the TCP is a 16-bit representation of the message, which means that there are 65.535 different representations. A checksum where all the bits are 0 does not exist, that is why there are 65.535 different representations instead of 2^{16} (= 65.536) different ones. The set of possible different messages that can be sent by the sender is infinite, so there are messages which are different that will have the same 16-bit representation. Thus it is possible that a message gets mutated during transmission while the checksum of this mutated message is the same as the checksum of the original message. When the receiver encounters this situation, it will accept a mutated message as being correct. The probability for this to happen is about 1 to 65.535, which in practice is small enough to assure reliable communication.

Deletion. When messages can be deleted during transmission, then the sender never knows whether a message that it has sent actually is received by the receiver. The solution for this problem is very common and used in many communication protocols. As soon as the receiver receives a message from the sender it sends an acknowledgement of this reception to the sender. As long as the sender did not receive an acknowledgement from the receiver it keeps on sending this message to the receiver. The sender only starts sending the next message when it has received an acknowledgement of the previous message. The receiver has a similar problem when it sends an acknowledgement to the sender. Because this acknowledgement can be deleted during transmission, the receiver never knows if the sender received it. The solution is here the same as for the sender. The receiver keeps on sending this acknowledgement until it receives the next message from the sender.

Before a sender starts retransmitting a message it has to consider the time it takes for the receiver to process a message and send an acknowledgement back. If the sender resends messages before the receiver could send an acknowledgement, then these messages become redundant with the potential risk of congestion of the connection. This also counts for the receiver retransmitting an acknowledgement. For discrete systems that communicate in cycles, agents have to wait one or more cycles before retransmitting. Real-time systems have to set a timer which represents the time the other agents needs to be able to react, which is known as the Retransmission Time-Out (RTO) [18]. As soon as an agent sends a message or acknowledgement it sets the timer and only starts retransmitting if the

timer has expired.

This acknowledgement mechanism is applied in a bit different way in the general algorithm in this thesis because this algorithm makes use of a sliding, or sending, window analogous to the TCP [17], which makes it possible to send more messages at a time before receiving acknowledgements. The underlying principle is however the same for both mechanisms. The operation of a sliding window will be explained in section 3.5.

Duplication. This means that the same message is received more than once by the receiver. This problem can be caused by the fact that the sender did send the message more than once to the receiver or because somewhere during the transmission the message got duplicated. The first case can happen because of the retransmission mechanism handling deletion errors, in which case duplication is not really an error. In either case, the receiver must know when it receives a message whether this is the first time that it receives this message or that it already received this message before. If the receiver were ignorant about this fact, then it would write the same fact twice on its output tape. one-way of checking whether the receiver already received a certain message before is by comparing this message with all the messages it already received. This is not very effective.

The solution used by the algorithms is using sequence numbers. All the messages are given a sequence number and this sequence number is sent together with the message to the receiver. The receiver now only has to compare the sequence number of a message that it receives with the highest consecutive sequence number of the messages that it received so far. If the sequence number of the message is less than or equal to the highest consecutive sequence number, then the receiver knows that it is a duplication and that it doesn't have to write this message on its output tape. When the sequence number is greater than the highest consecutive sequence number, then the receiver knows that it is a new message. If a sending window is used, as is the case in the presented general algorithm, then the receiver also has to compare the sequence number of the received message with the sequence numbers of the non-consecutive messages that it already received.

A sender can send an infinite amount of messages to a receiver. This would mean that the set of sequence numbers has to be infinite as well. The bigger the sequence number gets, the more space it takes to store it in the message. In order to prevent the sequence number using all the space of a message, the sequence number has a maximum value after which it starts over again. In theory it can happen that a message from a previous cycle of sequence numbers gets interpreted as belonging to the current cycle of sequence numbers. The range that the sequence number goes through has to be big enough to prevent this error from happening. The TCP uses for the sequence number a 32 bit number which means that there are more than $4 * 10^9$ different sequence numbers, which in practice is a big enough number [17].

Reordering. Because the general algorithm presented in this thesis makes use of a sliding window (see section 3.5), the sender does not have to wait for an acknowledgement of a message before it can send the next message. With a sliding window, the sender can send a sequence of messages to the receiver at once. If the connection consists of more than one channel, then these messages don't have to go to the receiver via the same channel. The transmission speed of these channels can differ, with the effect that a message that was sent before another message can arrive at the receiver after that message. In order to prevent a faulty output tape, the receiver first must be aware that it received the messages out of order and accordingly has to reorder these messages back to the original sequence before it writes them down to its output tape. The solution for this problem provided by the algorithm is the same as for the duplication error, which is adding sequence numbers to the messages. Because each message has a sequence number, the receiver knows what should be the right order of the messages it received.

Delay. A connection can consist of more than one channel for the transport of messages from the sender to the receiver. When these channels have different transmission speeds, messages can get delayed. This can lead to reordering errors as described above. When a delay lasts longer than it takes for the retransmission mechanism to resend this message, then the delayed message has no value for the receiver any more. The receiver still has to process this message and look whether it is a new message and also the connection system has to apply resources to get this now meaningless message to the receiver.

If a communication system has to handle a lot of these delayed messages, its performance can slow down. In order to prevent this from happening, a maximum lifetime for the messages is used. Every message that is sent has a maximum lifetime. When this lifetime is expired and this message is still somewhere in the connection system, then this message will be deleted, thus becoming a deletion error. This maximum lifetime mechanism also prevents the receiver from interpreting a delayed message with a sequence number from a previous round as belonging to the current sequence of messages it is receiving. When the time it takes to go through one cycle of sequence numbers is longer than the maximum lifetime of message, then a message will be deleted before its sequence number comes up again.

Insertion. While receiving a sequence of messages from the sender the receiver can receive a message that was not sent by the sender. The receiver has to notice this fact, otherwise it would write a message to its output tape that wasn't sent by the sender. The protocols presented in this thesis handles this problem by adding to every message the source and destination address of this message. When the receiver receives a message, it checks whether it is the receiver mentioned in the destination field and the receiver checks whether the source is the same as the sender it is receiving messages from. In this way a misdelivered message will not end

up on the output tape of the receiver.

Now that the possible errors have been discussed as well as how the presented algorithms overcome them, some conclusions can be drawn. The duplication of a message is not really an error for the algorithm because it is a feature of the algorithm. A message that is delayed is not a problem in itself but can cause a reordering problem or, if the delay lasts long enough, a deletion error. Mutated message are thrown out and are not handled by the algorithm so a mutation becomes a deletion error. A message that is inserted will also not be processed by the algorithm and is just deleted. So the algorithm actually has to handle only deletion and reordering errors. The only concern of an agent that sends a message or an acknowledgement is whether it arrives, and the only concern of an agent that receives messages is what the right order is of these messages.

3.4 Knowledge creation within a group

The goal of one-on-group communication is that all the members of the group gain a certain level of knowledge about a fact φ sent by the sender, and that all the members gain a certain level of knowledge about the knowledge of the group of this fact φ. 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 [18, 11]. 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 φ. Considering the general form of a message this can be achieved in two ways. Analogously to the TCP [17], 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 in the data part of the message or in the header. The group to whom the sender is sending a certain fact φ is meta-information about this fact φ, so it is preferred to store group information in the header of a package instead of in the data part.

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 φ_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 φ_{j+1} until it has received acknowledgements about the package with the previous fact φ_j from *all* the receivers. The sender then knows that all the receivers know the fact φ_j, and thus $K_S E_G \varphi_j$. Every receiver knows that the sender works this way, so when a receiver R_i receives a package with the next fact φ_{j+1}, it knows that the sender knows that all other receivers did receive the previous

package and thus know the previous fact φ_j, so $K_{R_i}K_S E_G \varphi_j$. 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, $K_S E_G^k \varphi_j$ respectively $K_{R_i}K_S E_G^k \varphi_j$. 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. 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 [18, 11] had been used, the receiving agents would not have known that the facts φ_j they received were sent to other receivers as well. Each of the receivers would have known not more than that the group consists 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 φ_j and φ_{j+i}, is $K_{R_i}K_S E_{\{S,R_i\}} \varphi_j$, and not the much stronger $K_{R_i}K_S E_G \varphi_j$. So when the goal is to attain a certain depth of group knowledge, the algorithms from [18, 11] are not sufficient.

3.5 Sliding window and knowledge

In the above description of one-on-one and one-on-group communication, the sender waits before sending the next package until it has received acknowledgements of this package from all the receivers in the group. When the sender wants to send a large sequence of packages, this is not very efficient. For every new package the sender wants to send, it has to wait for an acknowledgement of the previous package. A more efficient way of sending packages is by using a *sliding window* as used in the TCP [17]. The way a sliding window works is that the sender sends a sequence of packages at once to the receiver. When the receiver receives this sequence of packages, it only sends an acknowledgement of the highest consecutive package received, indicating that it received all the packages up to this one.

The size of the sliding window is determined by the amount of packages that the receiver can process at once. The receiver communicates this size as an advice to the sender by storing this advice in the header of the packages it sends to the sender. If this size is for example five, then the sender selects from its input tape the first five items and sends these items as packages to the receiver. If all these packages arrive at once at the receiver, the receiver sends an acknowledgement of the last package to the sender. If not all the five packages arrive at once at the receiver, the receiver sends an acknowledgement of the highest consecutive package it did receive. If this highest consecutive received package was for example package three, then the sender resends the packages four and five to the receiver. The sender keeps on sending the unacknowledged packages from the window until it receives an acknowledgement of the last package in the window. As soon as the sender receives this acknowledgement it slides the

window completely to the next five facts on the input tape. Again the sender selects these five new facts from its input tape and sends them as packages to the receiver. This cycle is repeated until the sender has sent all the items from its input tape.

The size of the window is variable during this process. So if the amount of packages that the receiver can process at once changes, it sends another advice for the window-size. The sender decides what the actual window-size is and makes this clear to the receiver by storing this actual window-size in the header of the packages it sends to the receiver. The sliding window was designed for one-on-one communication. For the sliding window to work with the one-on-group communication algorithm, the following adjustment had to be made. On the receiver side, nothing changes and all the receivers in the group send their window-size advice in the headers of the packages to the sender. The sender receives more than one advice and has to make a choice. The goal of the sliding window is flow control and congestion control [17, 19]. In order to control congestion, the sender does not send more messages at once than the receiver can handle at once. Thus, in the one-on-group algorithm the sender has too take the smallest size advised by the receivers. Should the sender choose the advice of a bigger window-size, then the communication with receivers who advised for a smaller window can get congested, slowing down the whole one-on-group communication.

How does the sliding window effect the accumulation of knowledge when used during communication? First let's have a look at the one-on-one communication. The window is assumed to have size w. The sender reads from its input tape the first w items and thus knows these items, so $K_S\varphi_j$ for $j = 1, .., w$. The sender sends these items as packages to the receiver and continues sending these packages until it received an acknowledgement of the last package from the current window. As soon as the sender receives this acknowledgement, the sender knows that the receiver knows that the sender knows the items from the first window, $K_S K_{R_i} K_S \varphi_j$ for $j = 1, .., w$. The sender then slides the windows and starts sending packages with the items from the second window. The receiver is waiting for the packages with the items φ_j for $j = 1, .., w$ to be transmitted by the sender. The receiver receives all these packages at once or just some of them, and sends an acknowledgement of the highest consecutive package it received. If the receiver did not receive all the packages from the window, the receiver keeps on sending an acknowledgement of the highest consecutive package it received until it has received all the packages from the window. If the receiver has received all the packages from the current window, then it keeps on sending an acknowledgement of the last package from the window until it receives a package from the next window from the sender.

In this way, at every cycle the knowledge of the others' knowledge of the items from a window grows for items sent in previous windows. The difference with the one-on-one communication where no sliding window is used is that without a sliding window, the depth of knowledge grows with every successfully sent fact and with a sliding window, the depth of knowledge grows with every successful slide of the window. The effect of the sliding window on the accumu-

lation of knowledge with one-on-group communication is analogous to the effect for the one-on-one communication. The difference is that the sender now has to wait until it receives an acknowledgement of the last package from the window from *all* the receivers before it can slide the window and start sending packages from the new window. The depth of knowledge of items from previously windows grows with every successful slide of the window. The depth of knowledge about the members of the group grows accordingly and is equal to the depth of knowledge of items from the first window.

Chapter 4

General algorithm and epistemic proof

The packages from the knowledge-based algorithm for one-on-one communication from [18], have the following form $K_{R/S}(position, data, window\_size)$. S/R identifies the source of this package (Sender of Receiver), and $K_{R/S}$ stands for: the source of this package knows this package. The *position* field contains the index number of the data element, the *data* field contains the actual data, and the *window_size* field represents the size of the window. Since the one-on-group algorithms from this thesis are extensions of this algorithm, the same notation form for the packages will be used. The algorithm from [18] is for one-on-one communication and thus there is only one sender and one receiver, which makes it obvious to whom the package is sent. In the situation of one-on-group communication this is not the case, so a *destination* field is added to the header. Besides the destination, also the group to which the message is sent has to be added to the package to solve the knowledge problem mentioned in section 3.4. This is done in the *group* field. The solution for the mutation error from section 3.3 was the use of a checksum which deletes mutated packages, so a *checksum* field is also added to the header. The package used by the one-on-group algorithm now has the following form:

$K_{source}(destination, checksum, group, position, window\_size, data)$

Notice that the *window-size* field and the *data* field have switched position in comparison with the packages of the algorithm from [18]. The reason for this is to emphasize the fact that a package consists of a header and some data to be sent. In the package first all the elements from the header are mentioned and the last element stands for the data. S/R is changed into *source* to make the notation more general. An explanation of some notation that appears in the algorithm follows:

source = source port where this package is sent from $[S, R_i]$ (S sender of the

data, Ri are receiving agents {for i = 1 to n});
Ksource = the source who sends this package knows this package;
destination = destination port $[S, R_i]$ (S is sending agent, R_i are receiving agents {for i = 1 to n});
checksum = 16-bit one's complement sum of the package;
group = group receivers to which the message is sent $[-, R_G]$ (No value for this element means that the sender communicates one-on-one, R_G stands for the sender communicates to the whole group);
position = position of data from the input tape;
window_size = size of the sliding window (sender sends actual size, receiver sends advice for window_size);
data = data that has to be transmitted.

The algorithm is designed for communication in asynchronous systems. This means that the algorithm works in systems were there is no central clock and it works in systems were the communication does not proceed in synchronous steps. The reason for this is that this communication algorithm now can be used on internet based networks which are asynchronous. This design choice makes the algorithm event-driven. As a consequence, the algorithm for the sender as well as for the receiver consists of two parts. One part handles the reception of the packages and the other part handles the sending of the packages. The reception of messages is independent and works asynchronously to the sending process. Both processes affect the same local knowledge state of an agent. Though being independent the sending and receiving algorithm influence each other's behaviour through the local knowledge state of an agent.

In the next table, variables and functions as used in the general algorithm are explained.

Acknowledgement		
ack_Ri	:	Used by S. An acknowledged sequence number received from Ri
high_cons_ack_Ri	:	Used by S. Highest consecutive acknowledgement number received from Ri
high_cons_ack_RG	:	Used by S. Highest consecutive acknowledgement number received form all receivers from group G
seq_number	:	A sequence number
high_cons_seq	:	Used by Ri. Highest consecutive sequence number of the packages Ri received from the sender
Checksum		
calculate-Checksum(package)	:	Calculates the checksum of a package

Window		
window_size	:	The size of the sliding window
offset	:	Used by S. The offset of the current window
latest_advert_Ri	:	Used by S. Latest advertisement for the window size received from Ri
latest_advert_RG	:	Used by S. Lowest of the latest_advert_Ri from all the receivers
estimateOptimal-WindowSize	:	Function that is called by Ri to its system and that returns the best window_size for the current state the system is in

Timer		
time_out	:	The Retransmission Time-Out or RTO
timer	:	Used by R. Timer starts when an acknowledgement is sent to the Sender
timer(seq,Ri)	:	Used by S. Starts when package with sequence *seq* is sent to Receiver R_i

4.1 General algorithm

The algorithm consists of four parts. The sender as wel as the receiver have an algorithm that handles the incoming packages, and have an algorithm that handles the outgoing packages. The lines in bold face are the lines from the algorithm and the lines between brackets contain some comment on these algorithm lines.

Sender (incoming packages)

```
1  for (i = 1 to n)
   {For all agents who sender is sending to, ... }
2     high_cons_ack_Ri = 0
      {... initialize the highest consecutive acknowledge number.}
3  end
   {High_cons_ack_Ri's initialized}
4  high_cons_ack_RG = 0
   {Initialize the overall highest consecutive acknowledgement.}
5  while true do
   {Get ready for receiving acknowledgements from the receivers you're sending to, ... [19]}
6     when received K_Ri(S, checksum, −, seq, window_size, −) do
      {You have received a package. Prepare for processing, ... [18]}
7        if (checksum = calculateChecksum(K_Ri(S, −, −, seq, window_size, −)))
         {Check if the checksum of the package is correct, delete package if not. [17]}
8           latest_advert_Ri = window_size
            {The last advertisement you have received from Ri is this one.}
9           latest_advert_RG = min(latest_advert_Ri (for i = 1 to n))
            {The last RG advertisement is the lowest latest advertisement from all Ri's.}
10          if (seq > high_cons_ack_Ri) do
            {If this acknowledgement from Ri is higher than the highest consecutive
             acknowledgement received so far from Ri, ... [16]}
11             high_cons_ack_Ri = seq
```

```
                {This is the new highest consecutive acknowledgement from Ri.}
12              forall ack_Ri with (ack_Ri ≤ high_cons_ack_Ri) do
                {For all the packages up to the highest acknowledgment, ...}
13                  store K_S K_Ri(−, −, −, seq, −, −)
                    {... store the fact that you know that Ri knows it.}
14              end
                {Acknowledgement from Ri updated.}
15              high_cons_ack_RG = min(high_cons_ack_Ri (for i = 1 to n))
              {The new high_cons_ack_RG is equal to the lowest high_cons_ack_Ri for all i's.}
16            end
              {[10] ... high_cons_ack_Ri and high_cons_ack_RG updated with
               acknowledgement from Ri.}
17          end
            {... [7].}
18      end
        {[6] ... finished processing of incoming package.}
19  end
    {... [5].}
```

Sender (outgoing packages)

```
1   window_size = 4
    {Set initial window-size.}
2   time_out = 20
    {Retransmission Time-Out (RTO). Common value for TCP is 20 ms.}
3   offset = 0
    {Reading of a tape starts at position 0.}
4   while true do
    {Start reading and sending an infinite tape, ... [24]}
5       forall seq with (offset ≤ seq < offset+window_size) do
        {For all the packages in the current window, ...}
6           read(seq,alpha)
            {... read the values from the tape, ...}
7           store K_S(−, −, −, seq, −, alpha)
            {... and store this information in your knowledge base.}
8       end
        {Tape within window has been read. Facts stored.}
9       while (high_cons_ack_RG ≠ offset+window_size-1) do
        {While not all the packages in the window have been acknowledged
        from RG (all agents), ... [21]}
10          forall seq with (offset ≤ seq < offset+window_size) do
            {For all the packages in the current window, ...}
11              for (i = 1 to n) do
                {... and for all receiving agents, ...}
12                  if not K_S K_Ri(−, −, −, seq, −, −) do
                    {... check if package 'seq' has not been acknowledged yet by Ri, ...}
13                      if (timer(seq,Ri) ≥ time_out) do
                        {... and its retransmission time has expired, ...}
14                          checksum =
                              calculateChecksum(K_S(R_i, −, group, seq, window_size, alpha))
                            {... calculate the checksum of the package to be sent ...}
15                          send K_S(R_i, checksum, group, seq, window_size, alpha)
                            {... (re)send the package to Ri.}
16                          timer(seq,Ri) = 0
                            {Reset the timer.}
17                      end
                        {A package for which the retransmission time was expired, ...}
18                  end
                    {... and that was unacknowledged by Ri, has been resent.}
```

```
19            end
              {A package has been resent to all agents that didn't acknowledge it.}
20          end
            {All packages from the sending window have been resent to all the receiving agents
             that didn't acknowledge these packages.}
21        end
          {[9] ... all the packages in the window have been acknowledged by all
          receiving agents Ri.}
22        offset = offset + window-size
          {Move offset with the size of the current window.}
23        window_size = latest_advert_Rg
          {Set the window-size to the latest group advertisement.}
24      end
        {... [4].}
```

Receiver (incoming packages)

```
1   while true do
    {Get ready for receiving an infinite tape, ... [7]}
2     when received K_S(R_i, group, checksum, seq, window_size, alpha) do
      {You have received a package (from S). Prepare for processing, ... [6]}
3       if(checksum = calculateChecksum(K_S(R_i, -, group, seq, window_size, alpha)))
        {Check if the checksum of the package is correct, delete package if not.}
4           store K_{R_i} K_S(-, -, group, seq, window_size, alpha)
            {Store the received package.}
5       end
        {... [3]}
6     end
      {[2] ... finished processing incoming package.}
7   end
    {... [1].}
```

Receiver (outgoing packages)

```
1   when K_{R_i} K_S(-, -, -, 0, -, -)
    {Wait until the first message is received.}
2   high_cons_seq = 0
    {Initiate the highest consecutive sequence at 0.}
3   time_out = 20
    {Set the retransmission Time-Out (RTO). Common value for TCP is 20 ms.}
4   timer = 0
    {Reset timer.}
5   while true do
    {Get ready to acknowledge incoming packages, ... [15]}
6     while not K_{R_i}(-, -, -, high_cons_seq + 1, -, -) do
      {Still not received package with sequence number 'high_cons_seq+1', ...}
7       if (timer ≥ time_out)
        {... and it is time to (re)transmit acknowledgement,
         Prepare for (re)transmitting, ...[12]}
8         window_size = estimateOptimalWindowSize()
          {Estimate the best window-size for the state the buffer of Ri is in.}
9         checksum =
            calculateChecksum(K_{R_i}(S, -, -, high_cons_seq, window_size, -))
          {Calculate the checksum of the package to be sent.}
10          send K_{R_i}(S, checksum, -, high_cons_seq, window_size, -)
          {Send acknowledgement.}
12          timer = 0
          {Reset the timer.}
13      end
```

```
        {[7] ... you've just (re)transmitted an acknowledgement package.}
14    end
      {You've received message high_cons_seq+1.}
15    high_cons_seq = high_cons_seq +1
      {You now know the next message. Increment high_cons_seq.}
16  end
    {... [5].}
```

4.2 Epistemic analysis and proof

In this section a proof is given for the gaining of knowledge as described in section 3.4. For the readability of the proof the form of the package is shortened to $Ksource(position, data)$. We assume that the group stays unchanged and for the destination we assume that the sender S sends to a receiver R_i and vice versa, so the *destination* field and the *group* field are left out. Further it is assumed that no mutation errors occur so the *checksum* field is also left out. Because the sliding window and the possibility of changing the window-size only effects the flow control, the *window-size* is left out as well.

Definition 2 *The following abbreviations are used in the proof:*

$K_{R_i}(p, \alpha)$: *"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 receiver of group G knows that the p-th data segment is α";*
$E_G(p, -)$: *"Every receiver of group G knows the value of the p-th data segment".*

Theorem 1 *Let $\mathcal{R}$ be any set of runs consistent with the knowledge-based algorithm from section 4.1 where:*

- *the environment allows for deletion and reordering errors, but no other kinds;*
- *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* $K_{R_i}K_S\left(j + \Sigma_{m=1}^{k} w_m, \alpha\right) \rightarrow \Box K_{R_i}K_S\left(E_G K_S\right)^k (j, \alpha)$.
[Back_i]: S *stores* $K_S K_{R_i}\left(j + \Sigma_{m=1}^{k} w_m, -\right) \rightarrow \Box K_S K_{R_i} K_S\left(E_G K_S\right)^k (j, -)$.
[Back_G]: S *stores* $K_S E_G\left(j + \Sigma_{m=1}^{k} w_m, -\right) \rightarrow \Box K_S\left(E_G K_S\right)^{k+1} (j, -)$.

In the proof below we use the following general principle from temporal logic, see section 3.1:

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

From the assumptions of the theorem, we can derive some consequences that we will regularly use in the proof:

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 from section 3.1.

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 $(k = 0)$:

$$R_i \text{ stores } K_{R_i}K_S(j,\alpha) \rightarrow \Box K_{R_i}K_S(j,\alpha) \tag{4.1}$$

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

$$\text{if } R_i \text{ sends } K_{R_i}(j,-) \text{ then } P(R_i \text{ stores } K_S(j,\alpha)) \tag{4.2}$$

$$\text{so } P\Box K_{R_i}K_S(j,\alpha), \text{ and } \Box K_{R_i}K_S(j,\alpha)$$

S only stores acknowledgements if it also received it from R_i, thus it knows that R_i has sent it in the past.

$$\text{If } S \text{ stores } K_SK_{R_i}(j,-) \text{ then } K_SP(R_i \text{ sends } K_{R_i}(j,-)) \text{ and } .. \tag{4.3}$$

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

$$K_SP(\Box K_{R_i}K_S(j,-)), \text{ and } K_S\Box K_{R_i}K_S(j,-), \text{ so } \Box K_SK_{R_i}K_S(j,-) \tag{4.4}$$

If (3) and (4) are put together, then we have the **Back_i**-part of the theorem for all i from the first window $(k = 0)$.
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 if it did receive those. 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.

$$\text{If } S \text{ stores } K_SE_G(j,-) \text{ then } K_SP(R_{i<i=1..n>} \text{ sends } K_{R_i}(j,-)) \tag{4.5}$$

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

$$K_S P(\Box E_G K_S(j,-)), \text{ and } K_S \Box E_G K_S(j,-), \text{ so } \Box K_S E_G K_S(j,-) \quad (4.6)$$

If (5) and (6) are put together, then we have the **Back_G**-part of the theorem for all j from the first window ($k = 0$).

What knowledge will emerge if the window slides? This will be 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 slides the window if it has received from all the receivers R_i an acknowledgement for the last package from the actual window.

$$S \text{ sends } \left(j + \Sigma_{m=1}^{k} w_m, \alpha\right) \rightarrow P\left(S \text{ stores } K_S E_G\left(j + \Sigma_{m=1}^{k-1}, -\right)\right) \quad (4.7)$$

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

$$S \text{ sends } K_S\left(j + \Sigma_{m=1}^{k} w_m, \alpha\right) \rightarrow \Box K_S\left(E_G K_S\right)^k (j,-) \quad (4.8)$$

R_i knows this fact. So if R_i receives a package from S with position mark $j + \Sigma_{m=1}^{k}$, 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:

$$R_i \text{ stores } K_{R_i} K_S\left(j + \Sigma_{m=1}^{k} w_m, \alpha\right) \rightarrow \Box K_{R_i} K_S\left(E_G K_S\right)^k (j,-) \quad (4.9)$$

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

[Back_i]:
R_i only sends an acknowledgement for the $j + \Sigma_{m=1}^{k} w_m$-th data element if it did store $K_{R_i} K_S\left(j + \Sigma_{m=1}^{k} w_m, -\right)$ in the past. With A, now the following can be derived:

$$R_i \text{ sends } K_{R_i}\left(j + \Sigma_{m=1}^{k} w_m, -\right) \rightarrow \Box K_{R_i} K_S\left(E_G K_S\right)^k (j,-) \quad (4.10)$$

S knows this fact. So if S receives an acknowledgement from R_i for the $j + \Sigma_{m=1}^{k} w_m$-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:

$$S \text{ stores } K_S K_{R_i}\left(j + \Sigma_{m=1}^{k} w_m, -\right) \rightarrow \Box K_S K_{R_i} K_S\left(E_G K_S\right)^k (j,-) \quad (4.11)$$

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 will have received an acknowledgement for the $j + \Sigma_{m=1}^{k} w_m$-th data segment from all R_i. Thus,

$$S \text{ stores } K_S E_G \left(j + \Sigma_{m=1}^{k} w_m, -\right)$$

With A and B, it can now be concluded that:

$$S \text{ stores } K_S E_G \left(j + \Sigma_{j=m}^{k} w_m, -\right) \rightarrow \Box K_S \left(E_G K_S\right)^{k+1} (j, -) \quad (4.12)$$

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

Chapter 5

CPS & one-on-group algorithm

In the previous sections we have seen that it is possible to attain a certain level of group knowledge while communicating one-on-group. The reason for developing a one-on-group communication algorithm for MAS are the needs of cooperative problem solving (CPS) as discussed in chapter 2. So let us have a look at how the general algorithm from section 4.1 can work for the CPS process. Communication within the CPS process consists of alternating one-on-one communication with one-on-group communication, depending at with phase of one of the four stages the CPS process is. The sender in the one-on-group communication is the initiating agent of the current phase of one of the CPS stages. This initiating agent can be the same agent throughout the whole process of CPS or there can be different initiating agents. In what way do both cases affect the gaining of group knowledge? And what are the implementation requirements for a CPS specific algorithm to assure the gaining of knowledge as discussed in chapter 3?

5.1 Gaining of knowledge throughout CPS

In between two one-on-group communication processes, the agents communicate one-on-one, and vice versa. For the one-on-one communication it can be the initiating agent communicating independently one-on-one with all the members of a potential team or it can be two agents communicating about a commitment to perform an action and communicating the outcome to a coordinating agent. After a successful one-on-group communication process, follows a one-on-one communication process. And after a successful one-on-one communication process, follows a one-on-group communication process. In fact, the transition from one to the other communication process can be seen as a notification that the previous communication process was successful.

During the team formation process for example, the initiating agent only communicates the fact that the agents of a potential group have taken on the

intention $\text{INT}(i, \psi \wedge \text{M-INT}_G(\psi))$, if this is true. So as soon as the members of a potential group receive the first one-on-group message communicated by the initiator, the members of this group know that the previous one-on-one communication process was successful. The transition from an one-on-group to an one-on-one communication process works similarly if the initiating agent stays the same. For example, an initiating agent only starts the plan formation process by communicating one-on-one to the members of the team if this initiating agent received an acknowledgement of the last one-on-group communication message from *all* the members of this team. So at the moment that the members of the group receive a plan formation one-on-one communication message from the initiating agent, they know that the one-on-group communication was successful.

So for the gaining of group knowledge, the transition from an one-on-one to an one-on-group communication process, and from an one-on-group to an one-on-one communication process is equal to the gaining of group knowledge at one cycle within one-on-group communication. All of this holds nder the condition that the initiator is the same agent during the whole process. Thus with each transition the depth of knowledge among the members of the group grows from k to $k + 1$.

During the transition from one-on-one communication to one-on-group communication, the initiator always stays the same agent. As discussed in chapter 2, this initiator is the only agent whose group knowledge is sufficient to start communicating something one-on-group. During the transition from a one-on-group communication process to a one-on-one communication process, the initiator can be any other agent from the group which makes the situation a bit more complex. The new initiating agent of the one-on-one communication process can start his role as initiator as soon as he received the last one-on-group message from the initiator of the one-on-group communication process.

It can be the case that one or more members of the group did not yet receive the last one-on-group message. So when the members of the group receive the first one-on-one message from the new initiator they don't know whether all the members did receive the last one-on-group message as is the case in the situation where the initiating agent stays the same. This means that the depth of knowledge among the members of the group doesn't grow during this transition. The new initiating agent from the one-on-one communication process only starts sending one-on-group messages about the fact established during the one-on-one communication process if this process was successful. For the one-on-one communication process to be successful, all the agents from the group must have received the last one-on-group message from the previous initiating agent. So when the agents of the group receive the first one-on-group message from the new initiating agent these agents know that all the agents received the last one-on-group message from the previous one-on-group communication process, and that all the one-on-one communication processes between the new initiator and the other receivers of the group were successful. So with this transition the depth of knowledge among the members of the group grows from k to $k + 1$.

The difference in gaining knowledge between the situation where the initiating agent changes and the situation where the initiating agent stays the same is that in the last situation the knowledge among the members of the group grows with every transition and that in the first situation the knowledge of the members only grows during the transition from one-on-one to one-on-group communication.

5.2 Adjusting the algorithm for CPS, Problems

In the previous section it is discussed that within the CPS process the group knowledge accumulates while going through the stages of CPS. For the general algorithm presented in section 4.1 it was proved that the group knowledge grows while a sender is communicating an arbitrary sequence of data one-on-group. The question is now, how this general algorithm has to be modified in such a way it can be used in the CPS process. A modified algorithm must be able to handle the transition from one-on-one to one-on-group communication and the transition from one-on-group to one-on-one communication with a possible change of the initiating agent. By 'handling' it is meant that the communication throughout the process is a stream of accumulating messages, thus assuring the gaining of knowledge proved in section 4.2. Let us take the general algorithm from section 4.1 and look where problems might be encountered while used in the different situations of a CPS process. The messages from the general algorithm are of the form:

$K_{source}(destination, checksum, group, position, window\_size, data)$.

When this message form is compared with the form of the messages used in the proof, $K_{source}(position, data)$, it is clear that in the latter case a lot of fields are left out. This implies that they are not important for the knowledge gaining. What is important is to determine which fields from the header are involved in the knowledge gaining process as described by theorem 1,. Informally, this theorem states that when an agent receives successfully k messages after any arbitrary message j, its knowledge of the group knowing this message j is gained to the power of k. The *destination* field is only used by the connection in the communication medium to determine to which agent this message has to be delivered and by the receiving agent which determines by this value whether it is supposed to receive this message. Thus the destination field is involved in message delivery and not in knowledge gaining. The *checksum* field is used by the receiving agent of a package to determine whether any mutation error has occurred. Being a check for mutation errors, the checksum is not involved in knowledge gaining. The *window_size* field is part of the sliding window mechanism which deals with flow and congestion control. The sliding window affects the knowledge as stated in theorem 1, but is not directly involved in the knowledge gaining process.

The *group* field is used by the receiving agent of the current package to determine to which other agents this message is sent. This group information

is involved in knowledge gaining but only in a way that a receiver can deduce knowledge about the knowledge of other agents. It does not contribute to the accumulating of knowledge or give problems in the knowledge gaining process during the changing of the initiator. The *source* field has a similar function as the group field. The receiver of a message determines by this field who the sender is of this message, deducing knowledge about the knowledge of this sender.

The last field is the *position* field. This field determines what the index position is on the input tape of the data element in the package. In the algorithm of section 4.1, the sender starts sending messages from an input tape. Every data segment from the tape has a sequence number which is sent together with the data segment by the sender. On reception of a message, a receiver acknowledges to the sender that it received this message by sending a message to the sender containing the same sequence number. This mechanism works well for the case where one sender is sending a predefined sequence of data to a group of receivers, which in fact is a one-way transport of data. The communication process from CPS is not a one-way transport of data but a dialogue communication process where the next message is not predefined but depends on what the answers are from the receivers. If we want to use the algorithm from section 4.1 for this dialogue type of communication with its interleaving one-on-one and one-on-group communication, a single index is not sufficient because some problems will be encountered.

It is very unlikely that during the independent one-on-one communication process between the sender and the receivers, the same amount of messages will be used. So for every sender-receiver communication within a one-on-one communication process a separate index is needed. This works for the situation where the initiator stays the same agent. 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 S_3 uses to communicate with R_1 starts at 100 and the index S_3 uses to communicate with R_2 starts at 200. Let us work out an example. S_3 sends four messages to R_1 which are received and answered by R_1. This answer 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 that follows the agents are identified by the numbers 1,2 and 3. If an agent acts as a sender or receiver, this is denoted by S1 or R1 respectively. The agents exchange messages. The arrow, `->` indicates the direction of the 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.

```
 Agent3              Agent1
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
7. S3 -(103,_,data)-> R1
8. S3 <-(104,_,answ)- R1
```

This brings the index for the next message to be sent to R_1 to 104. S_3 communicates two messages with R_2, which are answered by R_2, as follows

```
  Agent3              Agent2
1. S3 -(200,_,data)-> R2
2. S3 <-(200,_,answ)- R2
3. S3 -(201,_,data)-> R2
4. S3 <-(201,_,answ)- R2
```

This brings 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 communicate the outcome, S_3 has to communicate two messages one-on-group which are answered by R_1 and R_2:

```
  Agent1              Agent3              Agent2
1. R1 <-(104,G,data)- S3 -(202,G,data)-> R2
2. R1 -(104,_,answ)-> S3 <-(202,_,answ)- R2
3. R1 <-(105,G,data)- S3 -(203,G,data)-> R2
4. R1 -(105,_,answ)-> S3 <-(203,_,answ)- R2
```

After this successful one-on-group communication, S_3 enters the next stage where it has to communicate one-on-one again with the other agents from G. The index for the next message to R_1 is 106, and the index for the next message to R_2 is 204.

Introducing a separate index for each sender-receiver pair in the communication solves the problem of the unequal amount of messages sent during the one-on-one communication phase. Does this solution also work for the situation where the initiator changes after the one-on-group communication? The previous example ended 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, and the previous initiator S_3 will be denoted again as R_3. S_2 sends three messages to R_1 which are received and answered by R_1. Because S_2 and R_1 did not communicate to each other before, S_2 sets a new index to communicate with R_1 that starts at 300, as follows.

```
  Agent2              Agent1
1. S2 -(300,_,data)-> R1
2. S2 <-(300,_,answ)- R1
3. S2 -(301,_,data)-> R1
4. S2 <-(301,_,answ)- R1
5. S2 -(302,_,data)-> R1
6. S2 <-(302,_,answ)- R1
```

This brings the index for the next message to be sent from S_2 to R_1 at 303. S_2 also communicates one-on-one to R_3. The last communication between S_2 and R_3 was the message (302,_,answ) being sent from R_3 to S_2. Now, S_2 wants to send some data to R_3. Which index does it have to use? Agent S_2 did not yet communicate to R_3 in the setting of S_2 being the initiator. 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 being 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. Or S_2 can use the next index number, 204. It seems there are three options for agent S_2 to use the index for communicating one-on-one with R_3. For each of the three options worked out next, the last two communication lines of the previous one-on-group communication are taken as a starting point.

Option 1, S_2 sets new index.

```
  Agent1                Agent3                Agent2
1. R1 <-(105,G,data)- S3 -(203,G,data)-> R2
2. R1 -(105,_,answ)-> S3 <-(203,_,answ)- R2
3.                    R3 <-(400,_,data)- S2
4.                    R3 -(400,_,answ)-> S2
5.                    R3 <-(401,_,data)- S2
6.                    R3 -(401,_,answ)-> S2
```

Option 2, S_2 reuses the last index number.

```
  Agent1                Agent3                Agent2
1. R1 <-(105,G,data)- S3 -(203,G,data)-> R2
2. R1 -(105,_,answ)-> S3 <-(203,_,answ)- R2
3.                    R3 <-(203,_,data)- S2
4.                    R3 -(204,_,answ)-> S2
5.                    R3 <-(204,_,data)- S2
6.                    R3 -(205,_,answ)-> S2
```

Option 3, S_2 uses the next index number.

```
  Agent1                Agent3                Agent2
1. R1 <-(105,G,data)- S3 -(203,G,data)-> R2
2. R1 -(105,_,answ)-> S3 <-(203,_,answ)- R2
3.                    R3 <-(204,_,data)- S2
4.                    R3 -(204,_,answ)-> S2
5.                    R3 <-(205,_,data)- S2
6.                    R3 -(205,_,answ)-> S2
```

All the above options show some anomalies in the index numbering with respect to being an accumulating stream of packages. For the first option, there are two different consecutive communication streams between agent 2 and 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 is prone to communication errors and thus a situation that should be avoided. Besides being prone to errors, two parallel communication processes break the stream of accumulating messages necessary for the gaining of knowledge as proved in chapter 4. For the second option there are two anomalies. The first one is that in the one-on-one communication the receiver is the agent who 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 [17]. If R_3 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 three it is possible that agent 3 sends a next message (204,_,data) to agent 2 and receives from agent 2 a message (204,_,data) instead of (204,_,answ). Both agents have then sent a data message with index 204 and have also received a data message while both agents expected an answer message. This is a situation that should be avoided.

5.3 Adjusting the algorithm for CPS, Solutions

How can these problems be solved? TCP makes use of two indices per connection [21]. One index is configured by the sender and the other index is configured by the receiver. Thus for every message that is sent between the sender and receiver, a sequence number is sent as well as an acknowledgement of the last consecutive sequence number that is received. Could this two index system solve the index numbering problems? First let us look at a one-on-one communication process ending with a one-on-group communication using two indices per agent pair. Agent S_3 communicates four messages one-on-one to agent R_1, and communicates two messages with agent R_2. The first message sent by the sender to an agent only contains the sequence number of the sender. When the receiver receives this messages, it initiates its own sequence number and answers with a package containing this sequence number together with the acknowledged sequence number from the sender. Thus after two messages the sender and receiver know each other's sequence numbers.

```
 Agent1                  Agent3                    Agent2
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 mechanism of two indices works when the initiator changes. Lines 5 and 6 from the previous communication schema are used as starting point, and agent 3 becomes the sender. The first option with the one index mechanism was that S_2 set 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 the possibility of two parallel communication processes between both agents. Further it avoids breaking up the stream of accumulating messages which is needed for the gaining of knowledge proved in section 4.2. There are now two options left for S_2 when using the two index number mechanism. The first one is that it reuses the last index number and the other one is that it uses the next index number. Worked out, these options look as follows.

Option 1, S_2 reuses the last index number.

```
 Agent1                    Agent3                    Agent2
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
6.                        R3 -(303,402,_,answ)-> S2
```

Option 2, S_2 uses the next index number.

```
 Agent1                    Agent3                    Agent2
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
6.                        R3 -(303,403,_,answ)-> S2
```

For the first option, where S_2 reuses the last index number, the anomaly of the receiver increasing the index (as was the case in the situation with one index) does not occur here. However, the second anomaly still exists. Agent 2 still sends two messages with the same index numbers. If the first one was a message with an answer instead of an acknowledgement, then agent 2 cannot send another message with the same index numbers containing data for the new one-on-one communication. Option 2 looks a bit strange. Agent 2 sends two messages with the same acknowledgement number but increases its own sequence number. Again a similar problem can arise as with the single index number 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 won't know how to proceed so this is a situation that should be avoided.

Using a two index mechanism solves some of the problems that arise while the initiator changes but not all the problems. The problems that are left have one and the same cause. When another agent becomes the initiator it is not general knowledge that there is a new initiator. 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 the problems between these two agents as discussed above but also leads to problems for the other agents in the group which still act as receivers. These agents start getting one-on-one communication messages about the next stage from different agents acting as initiator. It might be clear that this is not a workable situation. To solve this problem, the algorithm has to provide a mechanism that prevents that more then one agent acts as an initiator.

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 that more than one agent acts as an initiator is that if any other agent wants to act as an initiator, this agent notifies the current initiator of this fact. Instead of just acknowledging the last one-on-group message from the current initiator the new initiator sends a request with this acknowledgement. Before the current initiator starts communicating one-on-one for the next stage, it now knows that there is another agent that wants to act as an initiator. The current initiator now can decide whether it continues itself as an initiator or whether it lets the other agent act as an initiator. If the current initiator decides to continue as an initiator, it continues communication one-on-one concerning the next stage. As soon as the 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 an initiator. If the current initiator decides that the other agent can act as the initiator it sends a message one-on-one to this agent confirming that it is the initiator for the next stage. After the new initiator receives this message, it knows that it is the initiator for the next stage and starts communicating messages one-on-one concerning the next stage. The agents are cooperative as stated in chapter 2 about CPS, so if the other agent has better resources for being the new initiator, the current initiator shall transfer the role of initiator to the other agent.

It can also be the case that there is more than one agent which notifies the current initiator that it wants to act as the initiator for the next stage. The current initiator makes a decision about which agent will get the role of initiator for the next stage. If the current agent decides to continue to act as an initiator, then it starts communicating facts one-on-one about the next stage. If the current initiator decides that another agent can act as an initiator, it sends a confirmation message to this other agent after which this agent starts communicating facts one-on-one about the next stage. For all the agents that announced themselves as potential initiator, but did not get this role handed to them by the current agent, will know this as soon as they receive a one-on-one message concerning the next stage from another agent in the group. As noted before, the agents are cooperative. The current initiator will choose the new initiator based on which agent has the best resources. Let us work three examples. In the first two examples, the current initiator and another agent both want to

act as initiator. In the first example the initiator changes and in the second example the initiator stays the same. The third example shows two other agents announcing themselves as potential initiators, after which the current initiator decides which agent will be the initiator for the next stage. The initiator is the same agent as the sender denoted in the schemes as S_i for the time it acts as an initiator. The fact that an agent announces itself as being a potential agent 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, it sends a message containing *answ* into the data field.

Example 1, S_2 becomes the new initiator.

```
  Agent1                    Agent3                    Agent2
1. R1 <-(102,201,G,data)- S3 -(301,400,G,data)-> R2
2. R1 -(202,102,_,answ)-> S3 <-(401,301,_,init)- R2
3.                         S3 -(302,401,_,answ)-> R2

  Agent3                    Agent2                    Agent1
4. R3 <-(402,302,_,data)- S2 -(500,_,_,data)  -> R1
5. R3 -(303,402,_,answ)-> S2 <-(600,500,_,answ)- R1
6. R3 <-(405,303,_,data)- S2 -(501,600,_,data)-> R1
```

Example 2, S_3 stays the initiator after init request from S_2.

```
  Agent1                    Agent3                    Agent2
1. R1 <-(102,201,G,data)- S3 -(301,400,G,data)-> R2
2. R1 -(202,102,_,answ)-> S3 <-(401,301,_,init)- R2
3. R1 <-(103,202,_,data)- R3 -(302,401,_,data)-> S2
4. R1 -(203,103,_,answ)-> R3 <-(402,302,_,answ)- S2
5. R1 <-(104,203,_,data)- R3 -(303,402,_,data)-> S2
6. R1 -(204,104,_,answ)-> R3 <-(405,303,_,answ)- S2
```

Example 3, S_2 becomes the new initiator after init request from S_2 and S_3.

```
  Agent1                    Agent3                    Agent2
1. R1 <-(102,201,G,data)- S3 -(301,400,G,data)-> R2
2. R1 -(202,102,_,init)-> S3 <-(401,301,_,init)- R2

  Agent3                    Agent2                    Agent1
3. R3 -(302,401,_,answ)-> S2
4. R3 <-(402,302,_,data)- S2 -(500,_,_,data)  -> R1
5. R3 -(303,402,_,answ)-> S2 <-(600,500,_,answ)- R1
6. R3 <-(405,303,_,data)- S2 -(501,600,_,data)-> R1
```

In the above three examples no anomalies in the index numbering are present. The combination of the two index mechanism together with the mechanism that regulates the change of the initiator handles the problems that could occur when the initiator changes during the CPS process.

Chapter 6

CPS specific algorithm and epistemic proof

The packages from the general algorithm from section 4.1 have the following form: $K_{source}(destination, checksum, group, position, window\_size, data)$. As discussed in section 5.2, an algorithm for CPS needs an index mechanism consisting of two indices and does not need a sliding window. The *window_size* field becomes the second index field. The first index contains the sequence number from the agent who is sending the package, and the second index field contains an acknowledgement of the sequence of the package this agent is reacting to. These fields will be called respectively the *sequence* field and the *acknowledgement* field. The package used by the CPS one-on-group algorithm now has the following form:

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

Here follows an description of the fields from the package from the CPS algorithm.
source = source port where this package is sent from $[S, R_i]$;
Ksource = the source who sends this package knows this package;
destination = destination port of package $[S, R_i]$;
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 at;
data = data that has to be transmitted.

In the next table, variables and functions as used in the CPS algorithm are explained.

Acknowledgement		
ack_Ri	:	Used by S. An acknowledged sequence number received from Ri
seqSRi	:	Used by S. Sequence number of packages S is sending to Ri
seqRi	:	Used by S. Sequence number of packages S is receiving from Ri
seqRi	:	Used by Ri. Sequence number of packages Ri is sending to S
seqS	:	Used by Ri. Sequence number of packages Ri is receiving from S
Data		
compose(data)	:	Used by S and Ri. Agent makes up the data it wants to send
Timer		
time_out	:	The Retransmission Time-Out or RTO
timer	:	Used by R. Timer starts when an package is sent to the Sender
timer(seqSRi)	:	Used by S. Starts when package with sequence $seqSR_i$ is sent to Receiver R_i
Checksum		
calculate-Checksum(package)	:	Calculates the checksum of a package

6.1 CPS algorithm

The algorithm consists of four parts. The sender as wel as the receiver have an algorithm that handles the incoming packages, and have an algorithm that handles the outgoing packages. The lines in bold face are the lines from the algorithm and the lines between brackets contain some comment on these algorithm lines.

Sender (incoming packages)

1 **for (i = 1 to n)**
{*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, ... [13]*}
5 **when received** $K_{R_i}(S, checksum, -, seqR_i, seqSR_i, data)$ **do**
{*You have received a package. Prepare for processing, ... [12]*}
6 **if (checksum = calculateChecksum(**$K_{R_i}(S, -, -, seqR_i, seqSR_i, data)$**))**
{*Check if the checksum of the package is correct, delete package if not. [11]*}
7 **if (seqSRi = ack_Ri + 1) do**
{*If this acknowledgement from Ri is equal to the next ack_Ri, ... [10]*}
8 **ack_Ri = seqSRi**

```
            {... this is the new current acknowledgement from Ri, ...}
9           store K_S K_Ri(S, −, −, seqR_i, seqSR_i, data)
            {... store the fact that you know that Ri knows it.}
10        end
          {[7] ... acknowledgement from Ri.}
11      end
        {... [6].}
12    end
      {[5] ... finished processing of incoming package.}
13  end
    {... [4].}
```

Sender (outgoing packages)

```
1   time_out = 20
    {Retransmission Time-Out (RTO). Common value for TCP is 20 ms.}
2   for (i = 1 to n) do
    {For all receiving agents.}
3     if not seqSR_i do
      {If S did not communicate to Ri before}
4       seqSRi = x
        {Initiate own sequence number for Ri at x}
5     end
      {seqSRi initiated.}
6   end
    {seqSRi for all receiving agents initiated.}
7   while true do
    {Start sending sequence of messages, ... [25]}
8     compose(data)
      {... ,make up the data for this package, ...}
9     store K_S(−, −, G, −, −, data)
      {... and store this information in your knowledge base.}
10    while (∃ ack_Ri ≠ seqSRi) do
      {While not all receivers have acknowledged the package with sequence seqSRi ...}
11      for (i = 1 to n) do
        {... and for all receiving agents, ...}
12        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          if (timer(seqSRi) ≥ time_out) do
            {... and its retransmission time has expired, ...}
14            checksum = calculateChecksum(K_S(R_i, −, G, seqSR_i, seqR_i, data))
              {... calculate the checksum of the package to be sent ...}
15            send K_S(R_i, checksum, G, seqSR_i, seqR_i, data)
              {... (re)send the package to Ri.}
16            timer(seqSRi) = 0
              {Reset the timer.}
17          end
            {A package for which the retransmission time was expired, ...}
18        end
          {... and that was unacknowledged by Ri, has been resent.}
19      end
        {A package has been resent to all agents that didn't acknowledge it.}
20    end
      {[9] ... all agents Ri have acknowledged the package with sequence number seqSRi.}
21    for (i = 1 to n) do
      {For all receiving agents, ...}
22      seqRi = seqRi + 1
        {Sequence number of next message from Ri is known. Increment seqRi.}
23      seqSRi = seqSRi + 1
```

```
        {Increment own sequence number for Ri.}
24    end
      {Sequence numbers for and from Ri updated.}
25  end
    {... [7].}
```

Receiver (incoming packages)

```
1   while true do
    {Get ready for receiving sequence of messages, ... [7]}
2     when received K_S(R_i, G, checksum, seqS, seqR_i, data) do
      {You have received a package (from S). Prepare for processing, ... [6]}
3       if(checksum = calculateChecksum(K_S(R_i, −, G, seqS, seqR_i, data)))
        {Check if the checksum of the package is correct, delete package if not.}
4         store K_{R_i} K_S(−, −, G, seqS, seqR_i, data)
          {Store the received package.}
5       end
        {... [3]}
6     end
      {[2] ... finished processing incoming package.}
7   end
    {... [1].}
```

Receiver (outgoing packages)

```
1   when K_{R_i} K_S(R_i, −, G, x, ∅, data)
    {The first message is received.}
2   seqS = x
    {The first sequence number from S is x.}
3   seqRi = y
    {Initiate own sequence number at y.}
4   time_out = 20
    {Set the retransmission Time-Out (RTO). Common value for TCP is 20 ms.}
5   timer = 20
    {Reset timer.}
6   while true do
    {Get ready to acknowledge incoming packages, ... [15]}
7     compose(data)
      {... ,make up the data for this package, ...}
8     while not K_{R_i} K_S(Ri, −, G, seqS + 1, seqR_i, data) do
      {Still not received package with 'seqS+1' (and 'seqRi'), ...}
9       if (timer ≥ time_out)
        {... and it is time to (re)transmit acknowledgement. Prepare for
        (re)transmitting, ...[12]}
10        checksum = calculateChecksum(K_{R_i}(S, −, −, seqR_i, seqS, data))
          {Calculate the checksum of the package to be sent.}
11        send K_{R_i}(S, checksum, −, seqR_i, seqS, data)
          {... (re)send acknowledgement.}
12        timer = 0
          {Reset the timer.}
13      end
        {[7] ... you've just (re)transmitted an acknowledgement package.}
14    end
      {You've received message seqS+1 wiht acknowledgement seqRi}
15    seqS = seqS+1
      {You know the sequence number of the next message. Increment seqS.}
16    seqRi = seqRi+1
      {Increment own sequence number, seqRi.}
17  end
    {... [6].}
```

6.2 Epistemic analysis and proof

In this section a proof is given for the gaining of knowledge as described in section 5.1. 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 for the destination we assume that the sender S sends to a receiver R_i and vice versa, so the *destination* field and the *group* field are left out. Further it is assumed that no mutation errors occur so the checksum field is also left out. Theorem 2 only states the gaining knowledge about the data sent by the sender. In the general algorithm, only the sender sends data and the receiver just acknowledges these data. In the CPS algorithm the receiver can also sent data instead of only acknowledgements. So every package sent is in fact an acknowledgement. The gaining of knowledge of data sent by the receiver is similar to the gaining of knowledge of data sent by the sender. From the perspective of the receiver, the receiver functions as the sender of the algorithm with the only restriction that it does not communicate one-on-group. For the proof we only need the sequence number; the acknowledgement number is left out. Theorem 2 and the proof are now analogous to the theorem and the proof of the windowless knowledge-based one-on-group algorithm from van Baars and Verbrugge in [20].

Definition 3 *The following abbreviations are used in the proof:*

$K_{R_i}(p, \alpha)$: *"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 receiver of group G knows that the p-th data segment is α";*
$E_G(p, -)$: *"Every receiver of group G knows the value of the p-th data segment".*

Theorem 2 *Let $\mathcal{R}$ be any set of runs consistent with the knowledge-based algorithm from section 6.1 where:*

- *the environment allows for deletion and reordering errors, but no other kinds;*
- *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 the same general principle from temporal logic as presented in section 4.2.

Proof
We prove *theorem 2* by induction on k.

First we look at the situation for $\mathbf{k = 0}$.
From B follows the **Forth**-part for $(k = 0)$ namely

$$R_i \text{ stores } K_{R_i} K_S(j,\alpha) \rightarrow \Box K_{R_i} K_S(j,\alpha). \tag{6.1}$$

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

$$\text{if } R_i \text{ sends } K_{R_i}(j,-) \text{ then } P(R_i \text{ stores } K_S(j,\alpha)), \tag{6.2}$$

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

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

$$\text{If } S \text{ stores } K_S K_{R_i}(j,-) \text{ then } K_S P(R_i \text{ sends } K_{R_i}(j,-)) \ldots \tag{6.3}$$

With A, C and the fact proven at (6.2) it can now be derived that:

$$K_S P(\Box K_{R_i} K_S(j,-)), \text{ and } K_S \Box K_{R_i} K_S(j,-), \text{ so } \Box K_S K_{R_i} K_S(j,-). \tag{6.4}$$

If (6.3) and (6.4) are put together, 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 if it did receive those. If S has received acknowledgements of a certain package from R_G where $G = \{1,\ldots,n\}$ then S knows that $R_{i<i=1..n>}$ have sent these acknowledgements in the past.

$$\text{If } S \text{ stores } K_S E_G(j,-) \text{ then } K_S P(R_{i<i=1..n>} \text{ sends } K_{R_i}(j,-)) \ldots \tag{6.5}$$

With A, C and the fact proven at (6.2) it can now be deduced that:

$$K_S P(\Box E_G K_S(j,-)), \text{ and } K_S \Box E_G K_S(j,-), \text{ so } \Box K_S E_G K_S(j,-). \tag{6.6}$$

If (6.5) and (6.6) are put together, 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 will emerge for $k \neq 0$? This will be 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 $(j+(k-1))$.

$$S \text{ sends } K_S(j+k,\alpha) \rightarrow P(S \text{ stores } K_S E_G(j+(k-1),-)). \quad (6.7)$$

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

$$S \text{ sends } K_S(j+k,\alpha) \rightarrow \Box K_S(E_G K_S)^k(j,-). \quad (6.8)$$

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 (6.8) together with A and B, the following can be derived:

$$R_i \text{ stores } K_{R_i} K_S(j+k,\alpha) \rightarrow \Box K_{R_i} K_S(E_G K_S)^k(j,-). \quad (6.9)$$

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 did store $K_{R_i} K_S(j+k,-)$ in the past. With A, now the following can be derived:

$$R_i \text{ sends } K_{R_i}(j+k,-) \rightarrow \Box K_{R_i} K_S(E_G K_S)^k(j,-). \quad (6.10)$$

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:

$$S \text{ stores } K_S K_{R_i}(j+k,-) \rightarrow \Box K_S K_{R_i} K_S(E_G K_S)^k(j,-). \quad (6.11)$$

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 will have received an acknowledgement for the $(j+k)$-th data segment from all R_i. Thus,

$$S \text{ stores } K_S E_G(j+k,-).$$

With A and B it can now be concluded that:

$$S \text{ stores } K_S E_G(j+k,-) \rightarrow \Box K_S(E_G K_S)^{k+1}(j,-). \quad (6.12)$$

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

Chapter 7

Design specifications for implementation

The third research question was, what are the possibilities for implementing the CPS algorithm in the internet architecture. Now that an algorithm is defined for communication involved in the CPS process, the implementation possibilities for such an algorithm can be explored. The main design specification is that such an algorithm can be used in the standard internet architecture. A protocol fitted in the internet architecture makes it possible to create MAS environments with agents that are located around the world without having to build a new network. In section 5.2 it has been discussed in what way the general algorithm of section 4.1 has to be modified to ensure the necessary reliable knowledge-based communication for CPS. The result of this discussion is the CPS algorithm from section 6.1. An implementation of an algorithm for CPS communication has to provide the properties of this CPS algorithm embedded in the model for reliable communication as stated in definition 3.1 from section 3.3. Before the feasibility of implementing the CPS algorithm in the internet protocol architecture will be discussed, first a brief overview of the TCP/IP network architecture follows. This overview is mainly based on [19, 2, 12].

7.1 TCP/IP network architecture

The TCP/IP network architecture is a layered model consisting of a stack of five independent protocol layers. See figure 7.1. The lowest layer is the hardware layer. The function of this layer is to transmit raw bits over a communication channel. There are a lot of different communication channels such as the telephone line, optic fibre, satellite connection, and many more. Every different communication channel has its own protocol which deals with the physical properties of the connection and with how bits are represented. On top of the hardware layer lies the network interface layer. During the sending process, this layer accepts internet datagrams from the internet layer and transmits these internet datagrams as bits over the communication channel. During the receiv-

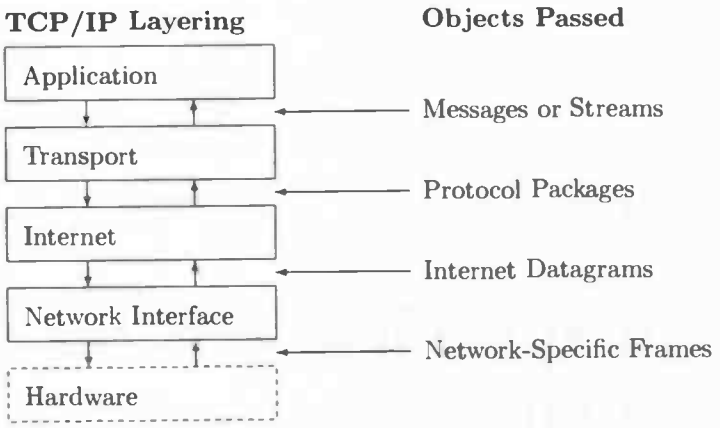

Figure 7.1: TCP/IP Network layer architecture

ing process the network interface protocol receives bits from the communication channel and delivers these bits as an internet datagram to the internet layer.

The protocol that runs at the internet layer is the internet protocol (IP). This protocol is the heart of the internet architecture. This protocol sends and receives internet datagrams which are the main packages the internet traffic consists of. On top of the internet layer lies the transport layer. The protocols at this layer send transport protocol datagrams to and receive transport protocol datagrams from the internet layer. At the transport layer different protocols can operate such as the transport control protocol (TCP), user datagram protocol (UDP), and more. Although there are more protocols at this layer than just the TCP, the internet architecture is in general referred to as TCP/IP architecture because the TCP is the most common protocol used at this layer. The highest protocol layer is the application layer. At this layer run applications which send data to remote applications. Depending on the requirements of an application with respect to reliability, the application chooses a suitable protocol from the transport layer. The application delivers its data to this protocol after which this data is processed down through the stack of protocols to a remote machine where this data is processed up through the stack of protocols to the destination application. Both machines can be connected directly or can be connected by a network. If both agents are situated in different interconnected networks, these networks are connected to each other by a router. A router is a special computer that is willing to transfer packages from one network to another network and consists of at least the lowest three layers of the TCP/IP architecture. Figure 7.2 shows a schematic overview of data sent from one agent to another agent connected by a router.

The IP at the internet layer is the basis of the TCP/IP network architecture. As can be seen in figure 7.2, the internet datagram is the highest conceptual object being transmitted over the network. Every datagram from a protocol from a higher level than the internet level is transported to the same higher

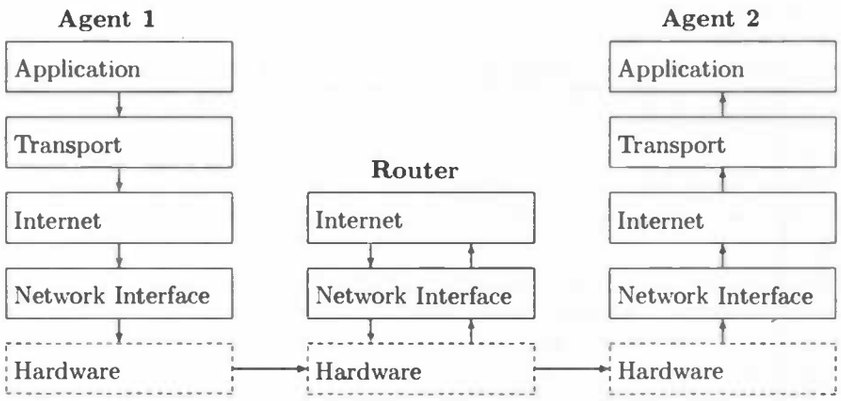

Figure 7.2: TCP/IP Data flow

level protocol at a remote computer embedded in an internet datagram. To prevent rebuilding the internet architecture, a live version of the CPS algorithm from section 6.1 should be implemented at a layer above the internet layer. With respect to definition 1 from section 3.3 of a communication system, this means that the connection from this definition includes at least the IP. Before the possibility of an implementation of the CPS algorithm can be explored, it should be clear what the properties of the internet layer are with respect to reliability.

7.2 Internet layer

Figure 7.3 shows a schematic view of an internet datagram. In this figure the internet datagram is represented by lines which are 32 bits long. Every field in the datagram consists of a certain amount of bits. The space occupied by a field corresponds to the amount of bits this field consists of. The length of the *options* field is variable so the amount of bits occupied by this field is also variable. The header must consist of a multiple of 32 bits. If the *options* field does not consist of a multiple of 32 bits, then accordingly the header also does not. If this problem occurs the *padding* fills up the option field with zero-bits to make it a multiple of 32 bits.

As discussed before, the internet layer is the basis for an implementation of the CPS algorithm. So what properties does this layer provide with respect to reliability. First let us compare the fields from the header of a CPS package with the fields of the internet datagram header. A CPS algorithm package has the following form:

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

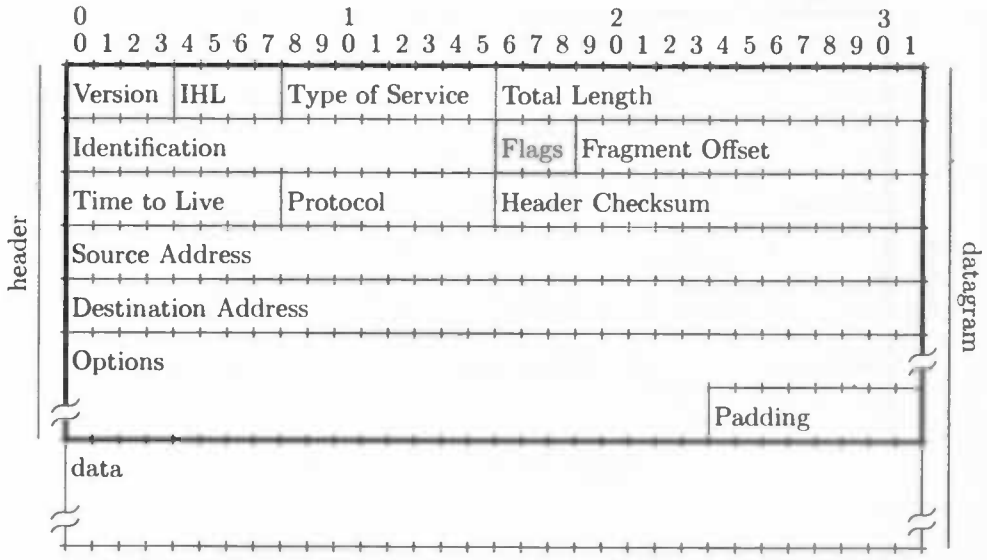

Figure 7.3: Internet Datagram [16, 2]

This package will be referred to as CPS package and consists of a header and of a data part. The header of this CPS package will be referred to as CPS header and the data part will be referred to as CPS data.

In both headers, a source and destination field are present. In the case of the CPS header these are the *source* and *destination* fields and represent the names of the sending and receiving agents. The *source address* and the *destination field* address from the internet datagram header contain respectively the IP numbers of the sending and receiving computers. Computers connected to an internet are identified by an IP number. An agent that runs as an application on a computer can be located by the IP number of its host computer. Just a name such as 'agent 2' or 'receiver 5' as a source address will not work at the internet level. In both headers a checksum is present.

The *checksum* field from the CPS header contains the checksum of the CPS package. This is the checksum which handles mutation errors as discussed in section 3.3. The checksum in the internet datagram header is the *header checksum* field and contains only the checksum of the internet datagram header and does not handle mutation errors occurring in the data that is sent. Thus the checksum mechanism from the CPS algorithm has to be implemented at a higher level. There is no counterpart for the *group* field from the CPS algorithm in the internet datagram. This makes sense because a group field is presented in this thesis as the solution for the knowledge-based one-on-group communication problem. The group knowledge mechanism also has to be solved at a higher level. For the *sequence* and the *acknowledgement* field from the CPS header also no counterparts are present in the internet datagram header. The

sequence and acknowledgement index mechanisms have to be solved at a higher level as well. The *time to live* field from the internet datagram header indicates the maximum time this datagram is allowed to live while being transmitted over am internet. The function of this field is to prevent the problem of delayed messages becoming redundant as discussed in section 3.3. Because the *time to live* field is already present in the internet datagram we don't have to account for it in the implementation of the CPS algorithm. The other fields from the internet datagram header are not of interest for the discussion here and will not be discussed.

From the transmission problems as discussed in section 3.3 only the delay problem is completely solved by the IP at the internet layer. The insertion error is partially solved by the IP. This problem is solved for the insertion of internet datagrams. When at the internet level an internet datagram arrives, the IP verifies by the destination address from the header whether it is supposed to receive this datagram. Because the other transmission problems are not solved at the internet layer they will have to be solved at a higher layer.

7.3 Transport layer

The conclusions of this chapter so far are that some of the transmission problems have to be solved at the transport layer or at the application layer and that a live version of the CPS algorithm should be an implementation at the transport layer or at the application layer. The transmission problems which still have to be solved are insertion, mutation, deletion, duplication and reordering errors. As discussed in section 3.3 the mechanisms that can handle these transmission errors are the checksum for mutation errors, destination verification for insertion errors, an index mechanism for duplication and reordering errors, and an acknowledgement mechanism for deletion errors. Besides these transmission problems the problem of group knowledge has to be solved as well. The solution for this problem is including group information. So the mechanisms that still have to be implemented for a working CPS algorithm are the following:

- checksum mechanism
- destination verification
- index mechanism
- acknowledgement mechanism
- group information

There are two possibilities for an implementation of the CPS algorithm. The first one is that the CPS algorithm is implemented as a transport control protocol at the transport layer, and the second possibility is that the CPS algorithm is implemented at the application level while it uses an existing protocol from the transport layer. Let us consider the most common protocol that exists at the transport layer to find out what its properties are. This protocol is the

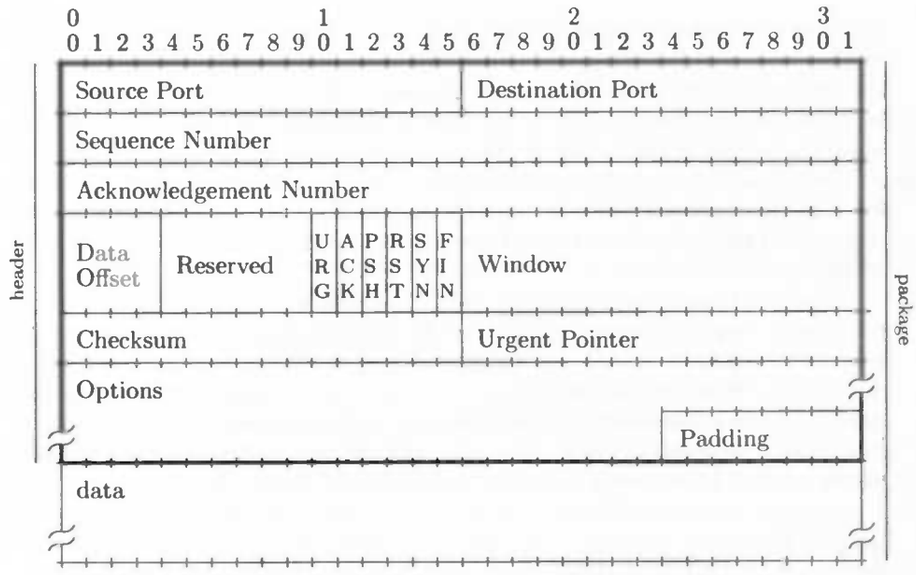

Figure 7.4: TCP datagram [17, 2]

TCP and formed the basis for the knowledge-based algorithm for the internet transmission control protocol from [18] which in turn was the basis of the general algorithm out of which the CPS algorithm has emerged, as presented respectively in section 4.1 and 6.1. The TCP is a reliable connection-oriented host-to-host communication service [17]. This means that it is a communication service for reliable communication between two hosts and that prior to the actual communication a connection between these hosts has to be established. What are the properties of this protocol?

A schematic view of a TCP datagram can be found in figure 7.4. This figure represents the TCP datagram in a similar way as the internet datagram is presented in figure 7.3. One of the fields from the TCP header is the *checksum* field. This field contains the checksum of the TCP datagram. The *data* field from this datagram contains data which an application wants to sent to a remote application. The *data* field is part of the TCP datagram so the *checksum* field handles possible mutation errors of the data as discussed in section 3.3. The *source port* and the *destination port* field are used by the TCP to identify a connection between two hosts. If a TCP datagram arrives at the TCP it uses the values of these fields to verify whether it is supposed to receive this datagram. The *source port* and the *destination port* field together with the TCP solve the part of the insertion problem that was not solved at the internet layer. The *sequence number* field contains a number which indicates what the position is of this datagram in the sequence of datagrams to be sent. With the *acknowledgement number* field the sender of this datagram acknowledges a datagram it received by putting in this field the sequence number of the

datagram it expects to receive next. Notice that this acknowledgement works slightly differently from the acknowledgement mechanism as discussed in section 5.2 where the acknowledgement contains the sequence number of the received package instead of the sequence number of the next datagram that is expected. The principle and the proceeding of both mechanisms are however the same. The other fields from the TCP datagram are not of interest for the discussion here and will not be discussed.

It seems that the checksum mechanism, destination verification, index mechanism and the acknowledgement mechanism are foreseen by the TCP header fields just discussed. It seems that we just need to add a group information field to the TCP header and we have a protocol datagram that satisfies the properties needed for an implementation of the CPS algorithm. Unfortunately this is not the case. The TCP is a transport protocol whereas the CPS algorithm is a dialogue algorithm. The TCP is used by an application that wants to send some data to another application. The application hands the data over to the TCP together with some information so that the TCP knows what the destination is for this data. If this data is larger than the maximum size of the data part, the TCP breaks the data up into smaller data segments and sends these subdivided data segments as a sequence to the TCP of the destination agent. Every subdivided data segment gets a TCP header and is transported to the TCP at the remote computer on which the receiving application runs. The remote TCP puts the subdivided data segments back to the original data and delivers it to the destination application. The application has no direct access to the fields from the TCP datagram. These fields are only used by the TCP. This means that the *sequence number* field and the *acknowledgement number* field have no meaning for the agent running as an application. In order to keep track of the order and amount of messages this agent has communicated it has to set up an index which administrates the sequence numbers of these messages. The situation now is that if an agent wants to send some data to another agent it stores this data together with the next sequence number in its knowledge base and sends this data to the TCP, after which the TCP transports this data to the TCP of the receiving agent.

The TCP is a reliable transport protocol. If the delivery of the data from the agent fails, the TCP will notify the sending agent of this failure. So instead of getting an acknowledgement the agent has to assume that the data is delivered as long as it does not receive a delivery failure notice. In the first situation, the agent gains *knowledge* about the delivery status and in the second situation, the agent gains *belief* about the delivery status. Especially in the situation where an agent has to communicate one-on-group, believing that a message is received is not sufficient. A solution could be that the receiving agent sends data via the TCP in which it acknowledges the reception of the data sent by the sender. This situation starts to look like an implementation of the CPS algorithm at the application level while it makes use of the TCP at the transport layer. This is not very efficient because now for every message that a sending agent sends to a receiving agent and in return for every answer or acknowledgement that the receiving agent sends to the sending agent, the TCP's from these agent have to

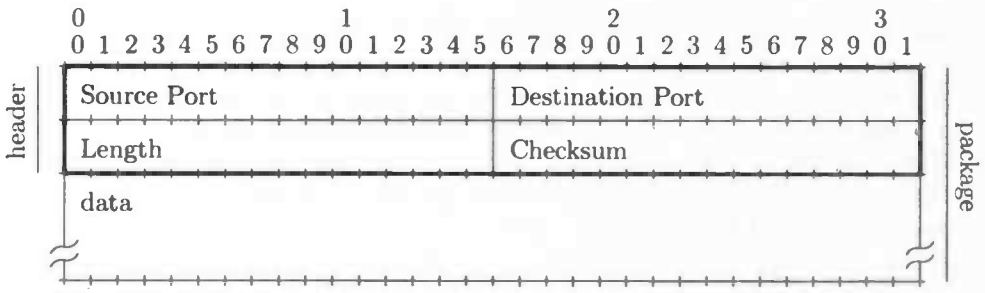

Figure 7.5: UDP datagram [2];(p.177)

establish a connection, transport the message, and close the connection. This means a lot of overhead in the communication. Besides this growing overhead there are now two mechanisms providing reliability operating at the same time. It can thus be concluded that the TCP is not a suitable protocol to be used by an application level implementation of the CPS algorithm, and the TCP itself is not a good model for a transport layer implementation of the CPS algorithm.

7.4 Application layer

The implementation of the CPS algorithm tends to move to an implementation at the application layer. The TCP is not a suitable protocol to be used by such an implementation. So, the question is whether there is another protocol at the transport control layer that can be used by an application layer implementation of the CPS algorithm. A protocol which is a lot simpler than the TCP is the user datagram protocol (UDP). In contrast with the TCP, the UDP is an unreliable connectionless host-to-host message protocol. The UDP provides a mechanism for applications to send messages to other applications with a minimum of protocol mechanism [15]. A schematic overview of a UDP datagram is presented in figure 7.5. The UDP header contains only four fields. If we know what the properties of this protocol are with respect to reliability it can be determined which mechanisms will have to be solved by an application level implementation of the CPS algorithm. The first two fields from the UDP header are the *source port* and the *destination port*. The *destination port* field determines the destination of the message within the destination computer. The *source port* determines the destination of the source within the computer where this message was sent from. When a UDP datagram arrives at the transport control layer the UDP can determine by the values of these fields whether it is supposed to receive this datagram before it delivers the data from the *data* field to the agent. This destination verification mechanism solves the part of the insertion problem that was not solved at the internet layer. The *checksum* field contains the checksum of the UPD datagram. The *data* field contains the data which an application wants to sent to a remote application. The *data* field is part of the UDP datagram, so the *checksum* field solves the mutation problem of

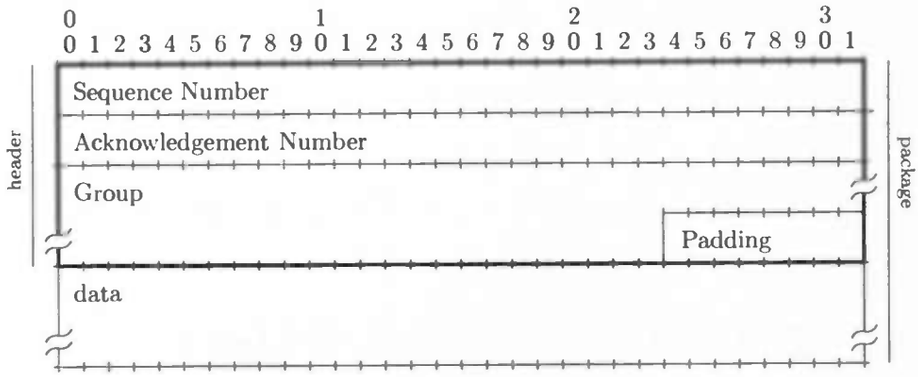

Figure 7.6: CPS datagram

the data as discussed in section 3.3. The *length* field contains a count of octets the datagram exists of but is not of interest for the discussion here. From the mechanisms that had to be solved for a workable CPS algorithm at the layers higher than the internet layer, the UDP provides the checksum mechanism and the destination verification mechanism. This means that an application level implementation of a CPS algorithm still has to provide an index mechanism, acknowledgement mechanism, and a group information mechanism, under the assumption that it makes use of the UDP at the transport control layer.

In the CPS algorithm the group information mechanism consisted of a *group* field that was added to the protocol package. In the case of one-on-group communication this field represents all the agents this message is sent to. The solution for the index mechanism is the use of two indices as discussed in section 5.2. One index keeps track of the sequence numbers of the messages sent by an agent and the other index keeps track of the sequence numbers of the messages received by an agent. In the protocol package from the CPS algorithm these indices are stored in the *sequence number* field and in the *acknowledgement number* field of the CPS header. The acknowledgement mechanism uses the *sequence number* field and the *acknowledgement number* field to acknowledge messages received by an agent. The three CPS header fields just discussed together with the data an agent wants to send to another agent form the CPS datagram of an application level implementation of the CPS algorithm. A schematic overview of the CPS datagram is presented in figure 7.6. The CPS algorithm delivers the CPS datagram to the UDP together with some information so that the UDP knows what the destination is of this CPS datagram. The protocol part of the CPS algorithm implementation is an implementation analogue to the CPS algorithm from section 6.1, with one exception. Because the checksum mechanism is accounted for at the UDP, this mechanism can be left out of the CPS implementation.

Chapter 8

Discussion and conclusion

In the overview of CPS in chapter 2, it became clear that for a successful process of collective problem solving, reliable one-on-one and one-on-group communication is needed. I referred to different one-on-one communication algorithms which formed the basis for the general one-on-group communication algorithm as presented in section 4.1. This algorithm allows true one-on-group communication, creating general knowledge up to any desired level about the identity of the group and the announcement sent to the group by an initiating sender. The first research question was if it is possible to design a knowledge-based algorithm for multi-agent communication that can handle the one-on-group sequence transmission problem. With the development of the general algorithm from section 4.1, this research question is positively answered.

Because the CPS communication process consists of more than just broadcasting a data tape to a group of receivers, the general algorithm had to be adjusted to the CPS algorithm as presented in section 6.1. This CPS algorithm can handle the initiator changes and allows true one-on-group communication, creating general knowledge up to any desired level about the identity of the group and the announcement sent to the group by different initiating senders. The second research question was if it is possible to design a knowledge-based algorithm for CPS communication. With the development of the CPS algorithm from section 6.1, this research question is also positively answered.

A related paper is [7] where a procedure is presented that, under some strong assumptions about the communication channels, trust among group members and temporary persistence of some relevant beliefs (e.g. the group should be aware of the procedure), establishes a common belief C-BEL$_G(\varphi)$. The idea is essentially that one initiator first broadcasts the message φ to all agents in the group, based on a standard low-level communication protocol such as TCP, ensuring that it knows at a certain point that E-BEL$_G(\varphi)$; then the initiator broadcasts the message that C-BEL$_G(\varphi)$ to all of them. Typically, such strong assumptions are only true in very fixed multi-agent systems, such as participants in a rescue operation who work in a fixed team according to a commonly known fixed procedure [7]. The two one-on-group procedures presented in this thesis do not establish common beliefs but only fixed levels of group knowledge,

but can work in open environments because the prerequisites are much weaker than those in [7]. This is in line with the argument in [6] that developers of multi-agent systems can decide beforehand, according to organization structure, goal and environment, which level of team-awareness of relevant propositions is appropriate for a given application.

As future work in this direction, it would be interesting to design a logic exactly suited to communication protocols such as TCP and the two one-to-many protocols given here, in a similar fashion as the sound and complete system TDL developed by Lomuscio and Woźna in [13] for authentication protocols. 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.

Research question three asked the question what the possibilities are of implementing the CPS algorithm in the internet architecture. For an implementation of the CPS algorithm to be of practical use it has to fit in the standard internet architecture. As discussed in chapter 7, such an implementation should at least make use of the IP at the internet layer of the TCP/IP architecture. Chapter 7.3 showed that the TCP at the transport layer is not a useful protocol for an implementation of the CPS algorithm. This chapter further showed that some of the mechanisms needed by the CPS algorithm have to be implemented at the application level. Availability of these mechanism at the transport layer leads to a redundant implementation. In section 7.4 it became clear that the UDP is a very useful protocol to be used by an application level implementation of the CPS algorithm.

As future work in the practical direction, an implementation should follow at the application level which makes use of the UDP protocol at the transport level. This implementation then should be tested in different real world MAS environments. These tests should back up the proof from this thesis about reliability and knowledge gaining.

Bibliography

[1] F. Dignum, B. Dunin-Kęplicz, and R. Verbrugge. Creating collective intention through dialogue. *Logic Journal of the IGPL*, 9(2):289–303, 2001.

[2] D. E. Douglas. *Internetworking with TCP/IP, Volume 1: Principles, Protocols and Architectures.* Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2006.

[3] B. Dunin-Kęplicz and R. Verbrugge. Collective intentions. *Fundamenta Informaticae*, 51(3):271–295, 2002.

[4] B. Dunin-Kęplicz and R. Verbrugge. Dialogue in teamwork. In J. M. Fonseca et al., editor, *Proceedings of The 10th ISPE International Conference on Concurrent Engineering: Research and Applications*, pages 121–128, Rotterdam, 2003. A.A. Balkema Publishers.

[5] B. Dunin-Kęplicz and R. Verbrugge. A tuning machine for collective commitments. In *Proceedings of The First International Workshop on Formal Approaches to Multi-Agent Systems*, pages 99 – 116, 2003.

[6] B. Dunin-Kęplicz and R. Verbrugge. A tuning machine for cooperative problem solving. *Fundamenta Informaticae*, 63:283–307, 2004.

[7] B. Dunin-Kęplicz and R. Verbrugge. Creating common beliefs in rescue situations. In B. Dunin-Keplicz, A. Jankowski, A. Skowron, and M. Szczuka, editors, *Proceedings of Monitoring, Security and RescueTechniques in Multiagent Systems (MSRAS)*, Advances in Soft Computing, pages 69–84, Berlin, 2005. Springer.

[8] R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge.* MIT Press, Cambridge (MA), 1995.

[9] R. Goldblatt. *Logics of Time and Computation.* Number 7 in CSLI Lecture Notes. Center for Studies in Language and Information, Palo Alto (CA), 1992.

[10] J. Halpern, R. van der Meyden, and M. Vardi. Complete axiomatizations for reasoning about knowledge and time. *SIAM Journal on Computing*, 33(3):674–703, 2004.

[11] J. Y. Halpern and L. D. Zuck. 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 Sistributed Computing*, pages 269–280, 1987.

[12] J. F. Kurose and K. W. Ross. *Computer Networking, A Top-Down Approach Featuring the Internet.* Addison Wesley, 2003.

[13] A. Lomuscio and B. Woźna. A complete and decidable security-specialised logic and its application to the TESLA protocol. In P. Stone and G. Weiss, editors, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 145–152. ACM Press, 2006.

[14] J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Schience.* Cambridge University Press, Cambridge, 1995.

[15] J. Postel. User datagram protocol (UDP). Technical Report RFC 768, Internet Society, 1980. ftp://ftp.rfc-editor.org/in-notes/rfc768.txt.

[16] J. Postel. Internet protocol (IP). Technical Report RFC 791, Internet Society, September 1981. ftp://ftp.rfc-editor.org/in-notes/rfc791.txt.

[17] J. Postel. Transmission control protocol (TCP). Technical Report RFC 793, Internet Society, September 1981. ftp://ftp.rfc-editor.org/in-notes/rfc793.txt.

[18] F. Stulp and R. Verbrugge. A knowledge-based algorithm for the internet protocol TCP. *Bulletin of Economic Research*, 54(1):69–94, 2002.

[19] A. S. Tanenbaum. *Computer Networks.* Prentice Hall Professional Technical Reference, 2003.

[20] E. van Baars and R. Verbrugge. Knowledge-based algorithm for multi-agent communication. In G. Bonanno, W. van der Hoek, and M. Wooldridge, editors, *Proceedings of the 7th Conference on Logic and the Foundations of Game and Decision*, pages 227 – 236, 2006.

[21] M. Wooldridge and N. R. Jennings. The cooperative problem-solving process. *Journal of Logic and Computation*, 9(4):563–592, 1999.

Appendix A

CPS process

The packages used in the following graphical representation have the following form: $K_{source}(destination, group, data)$. *Source* represents the sender, *destination* represents the destination of the package, *group* represents the group this package is sent to, and *data* is the data to be sent.

1. Potential recognition
    - Initial situation, figure A.1
    - In progress, figure A.2
    - Outcome situation, figure A.3
2. Team formation
    - Initial situation, figure A.4
    - In progress, figure A.5
    - Outcome situation, figure A.6
3. Plan formation
    - Initial situation, figure A.7
    - In progress (1), figure A.8
    - In progress (2), figure A.9
    - In progress (3), figure A.10
    - In progress (4), figure A.11
    - In progress (5), figure A.12
    - Outcome situation, figure A.13
4. Team action
    - Initial situation, figure A.14
    - In progress, figure A.15
    - Outcome situation, figure A.16

Level 1: Potential recognition *(initial situation)*
Initial situation: Agent-4, initiator, has a goal φ that it wants to be achieved, (GOAL(4,ψ)).
Outcome: Potential for cooperation that initiator agent-4 sees with respect to ψ, denoted as PotC(4,ψ). PotC(4,ψ) means that φ is the goal of agent-4 and there are one or more groups G_i such that agent-4 believes that G_i can collectively achieve ψ. The index i denotes the different potential groups.

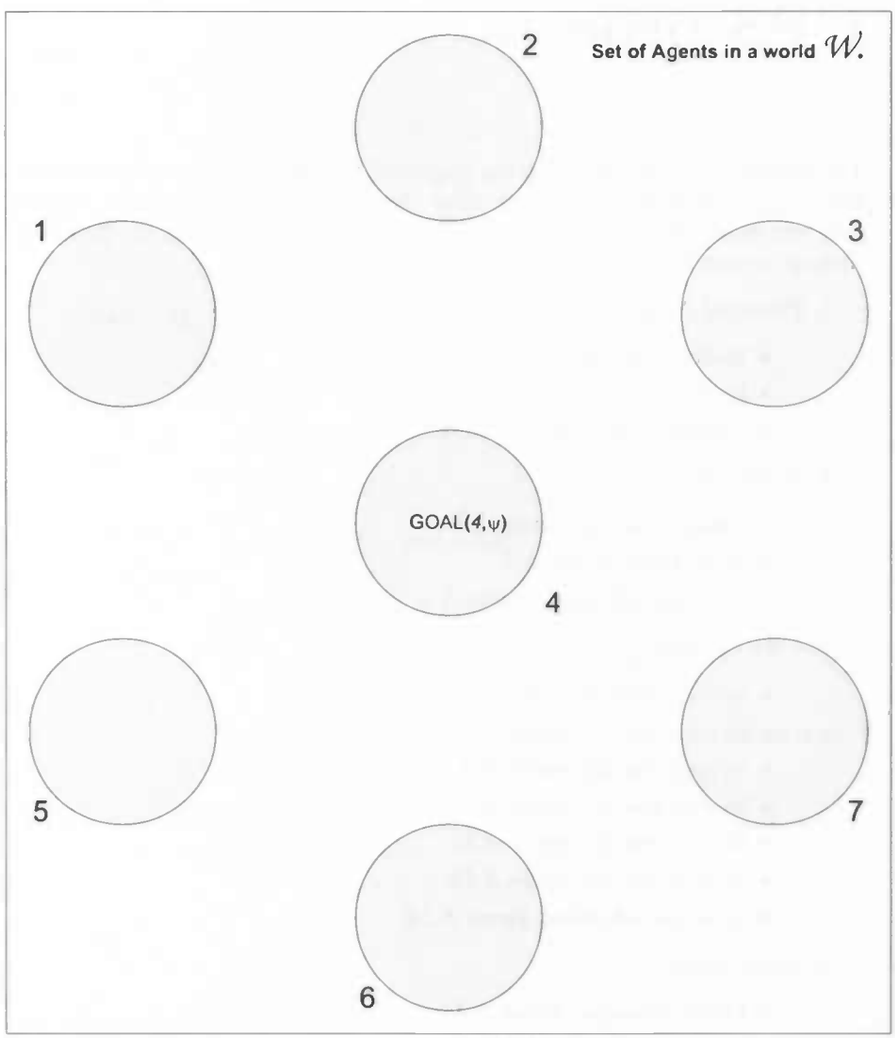

Figure A.1: Potential recognition, initial situation

Level 1: Potential recognition *(in progress)*
Initiating agent-4 inquires what the *abilities, opportunities, willingness* and *commitment strategies* of the different agents are with respect to the goal φ. He does this through *one on one* communication.

Dialogue types: *Information seeking.*
Speech acts: *ASS(ert), REQ(uest).*
ψ(a,_,w,c) means: with respect to ψ this agent is able and willing to cooperate in achieving this goal and has commitment strategie c with respect to this goal
ψ(a,o,w,c) means: with respect to ψ this agent is able, has the opportunities and is willing cooperate in achieving this goal and has commitment strategie c with respect to this goal

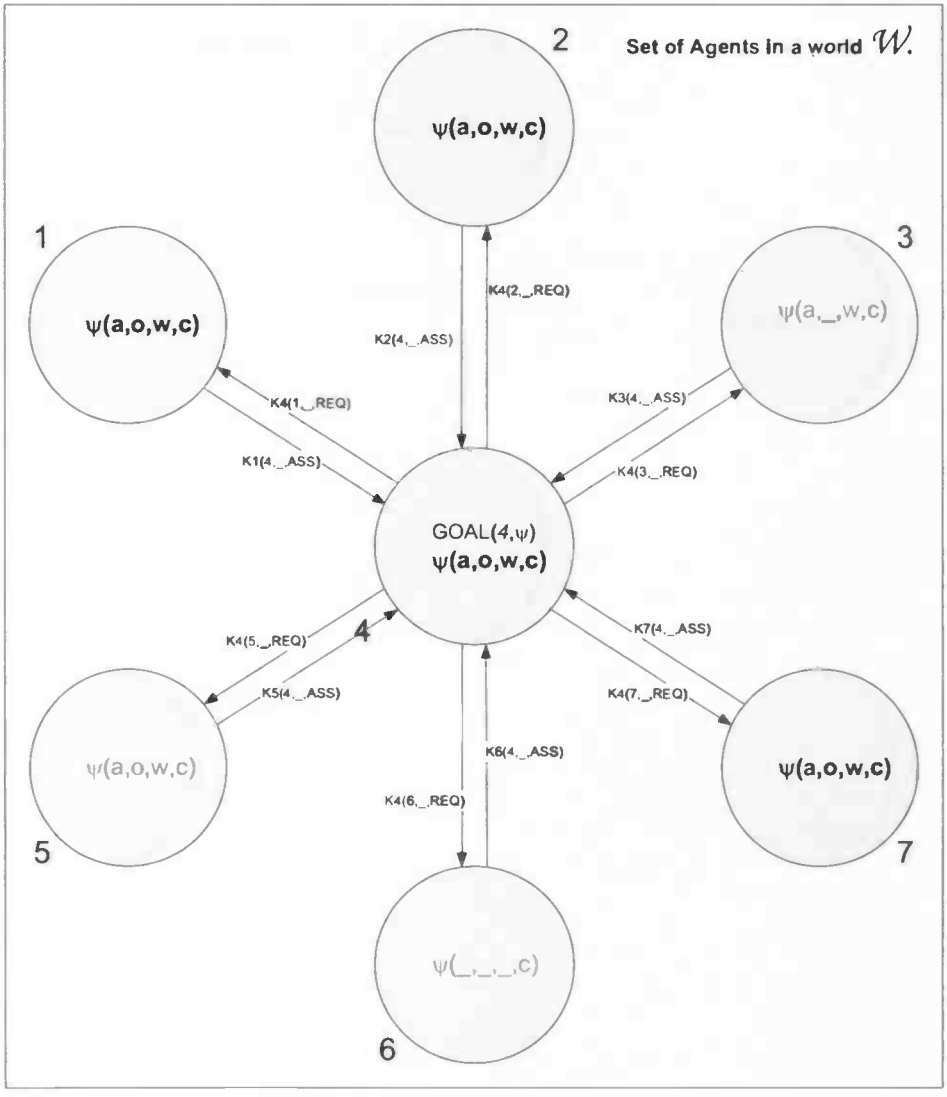

Figure A.2: Potential recognition, in progress

Level 1: Potential recognition *(outcome situation)*
Outcome: Initiating agent-4 sees potential for cooperation (PotC(4,ψ)) with respect to goal ψ and with one ore more groups Gi. Visualised here is group G1 = [1,2,4,7]

Agent-4 can transmit the outcome per potential group to each of the agents that are member of this group for wich agent-4 sees potential. He does this through *one on one* communication. This is an optional step for this stage.

Dialogue types: *end fase of Information seeking.*
Speech acts: *ASS(ert).*

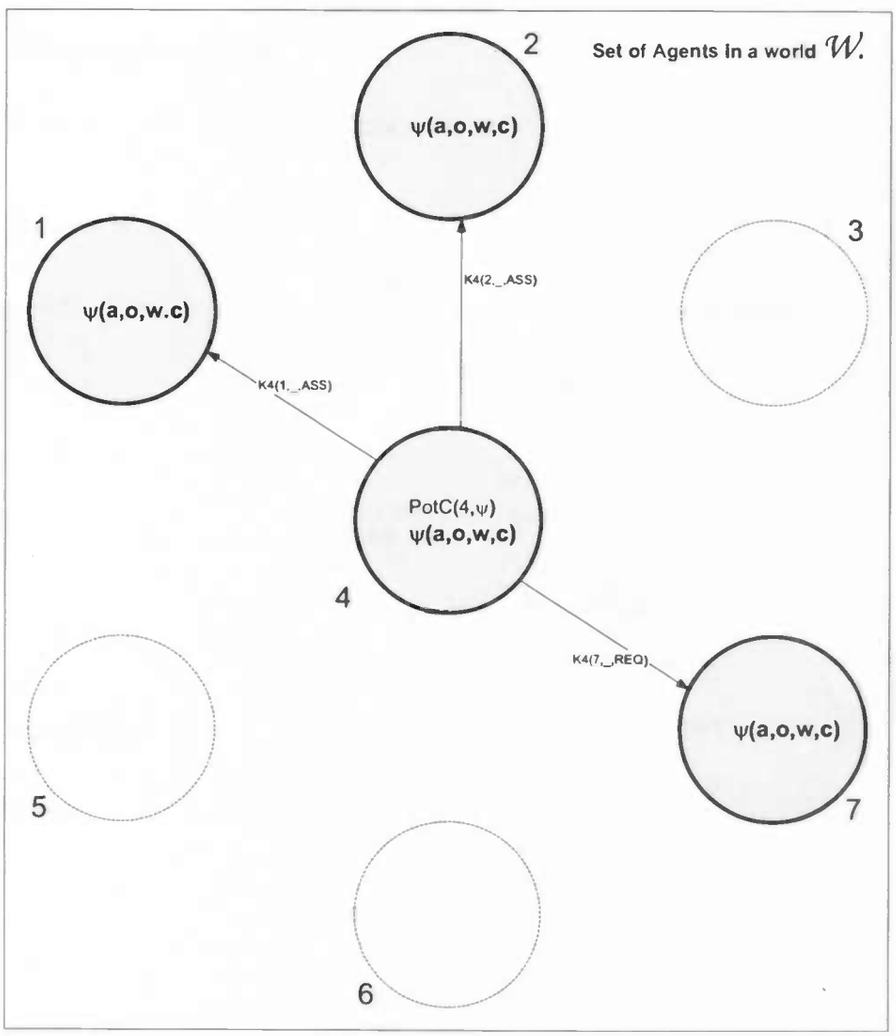

Figure A.3: Potential recognition, outcome situation

Level 2: Team formation *(Initial situation)*
Initial situation: Initiating agent-4 has a sequence of potential groups for achieving goal ψ.
Visualised here is group G1 = [1,2,4,7]
Outcome: Initiating agent-4 has found one group Gi which has a collective intention C-INT$_G$(ψ) to achieve ψ.

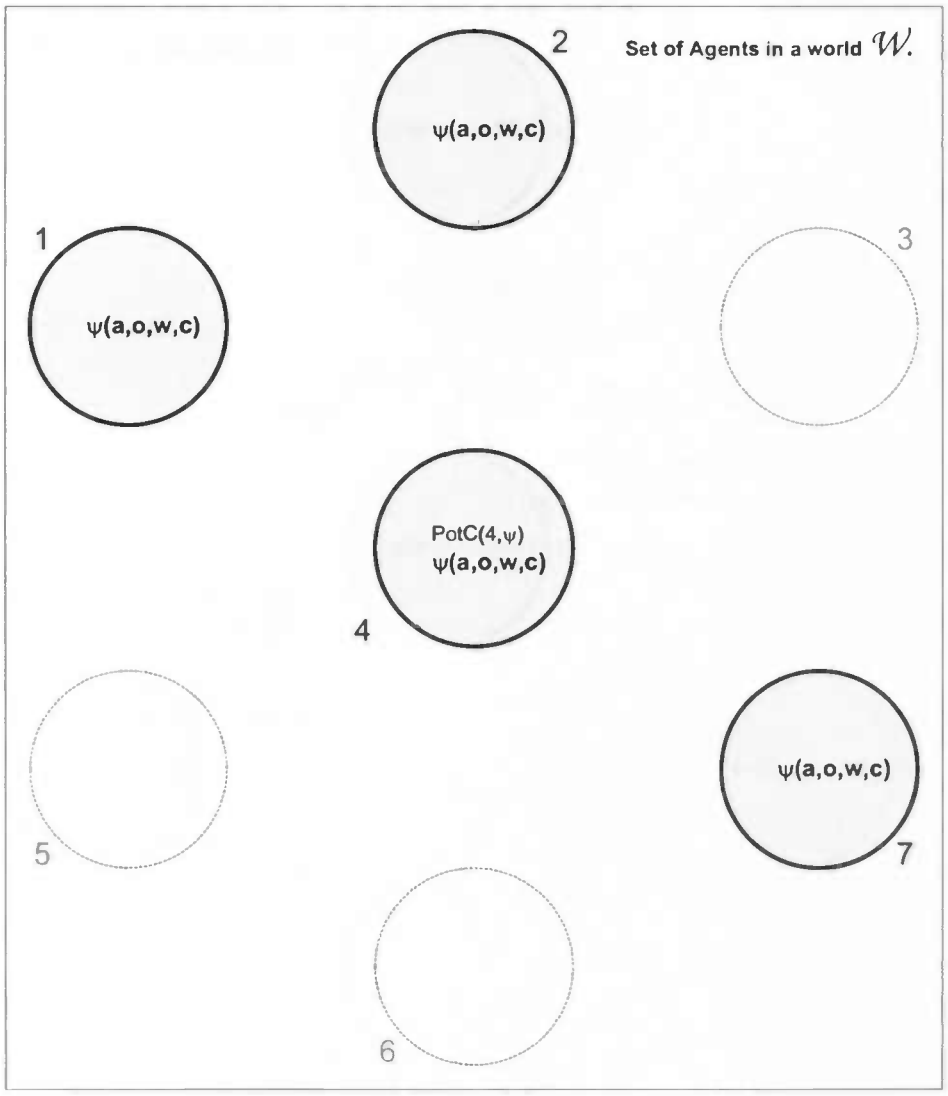

Figure A.4: Team formation, initial situation

Level 2: Team formation *(In progress)*
Initiating agent-4 tries for all potential groups to persuade the members to take on an individual intention towards φ and the intention that there be a mutual intention among that group (INT(i,ψ ∧ M-INT$_{G1}$(ψ))). He does this through *one on one* communication.

Dialogue types: *Persuasion.*
Speech acts: *ASS(ert), REQ(uest), CHALL(ange), CONC(ede).*

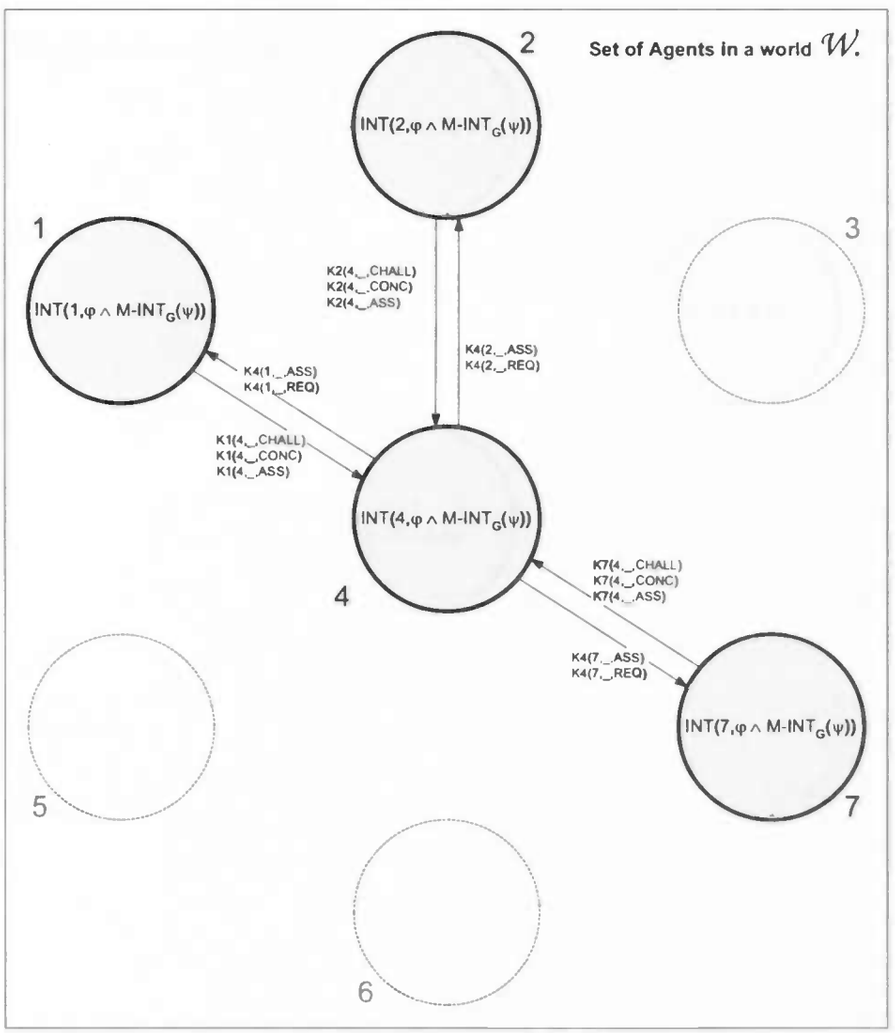

Figure A.5: Team formation, in progress

Level 2: Team formation *(Outcome situation)*
Outcome: Initiating agent-4 has found group G1 where all the members have the intention to achieve ψ and have the intention that there be a mutual intention among G1 (INT(i,$\psi \wedge$ M-INT$_{G1}(\psi)$)). The collective intention C-INT$_G(\psi)$ emerges from the fact that intitiating agent-4 comunicates through *one one group* communication that all the members in G1 did take on INT(i,$\psi \wedge$ M-INT$_{G1}(\psi)$).

Dialogue types: *end fase of Persuasion.*
Speech acts: *ASS(ert).*

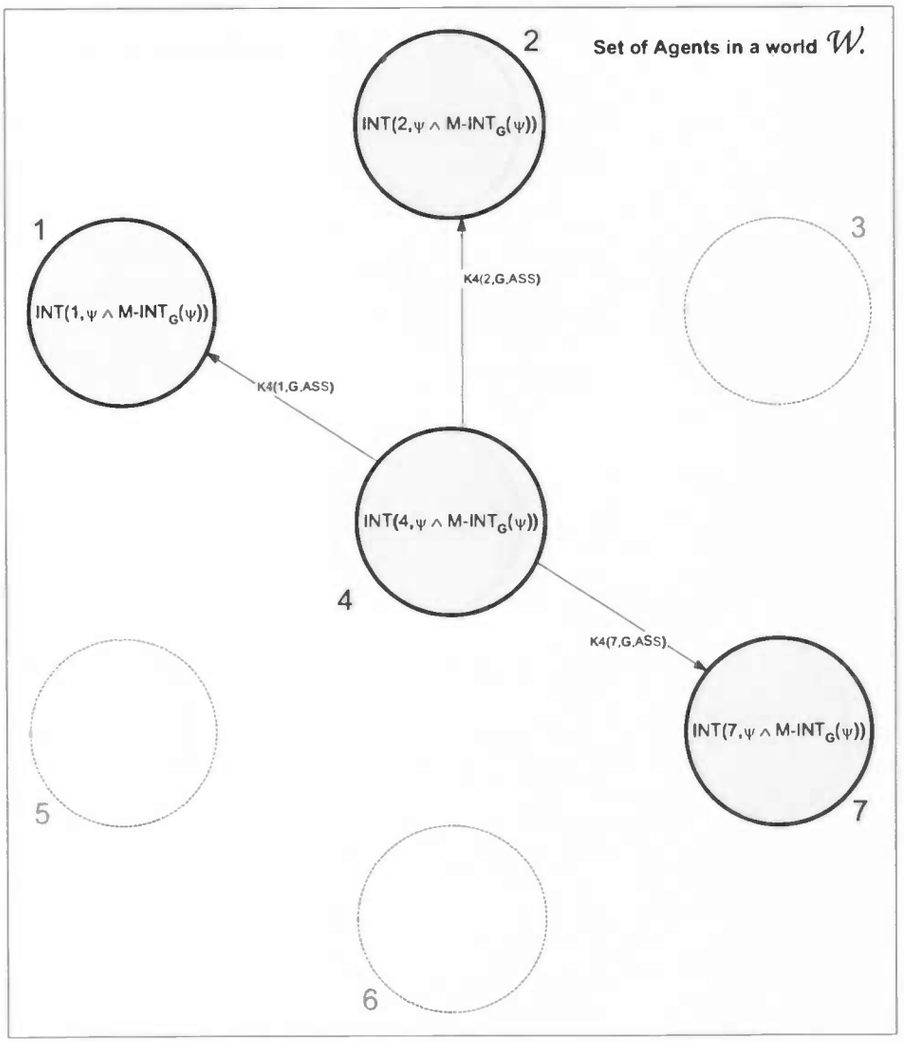

Figure A.6: Team formation, outcome situation

Level 3: Plan formation *(Initial situation)*
Intitial situation: Group G1 together with the collective intention to achieve ψ, ($C\text{-}INT_G(\psi)$).
Outcome: A collective commitment of group G1 based on a plan P. There are al lot of different kinds of collective commitments. A common kind is the strong collective commitment $SC\text{-}COMM_{G1,P}(\psi)$ wich will be used here .

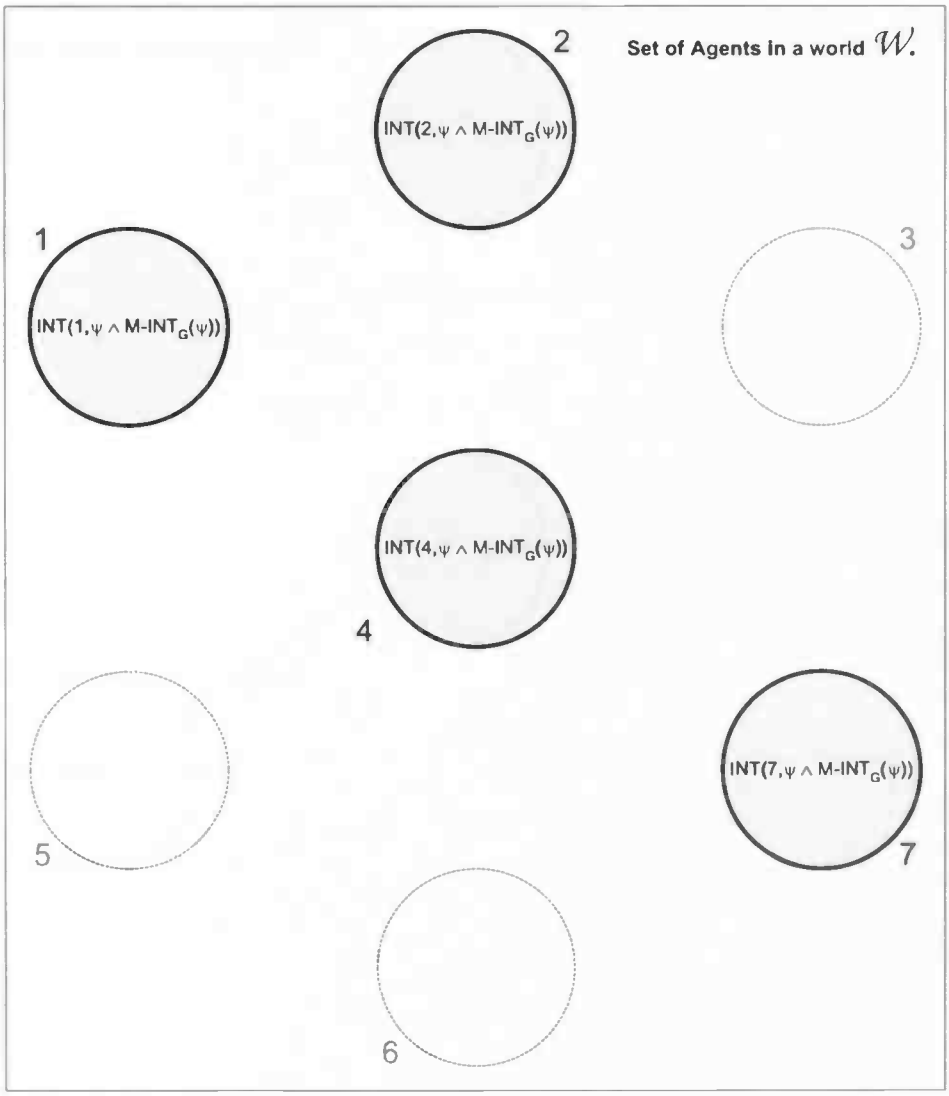

Figure A.7: Plan formation, initial situation

Level 3: Plan formation *(In progress – Task division(1))*
Any agent (here we use agent-4) from group G1 can act as initiating agent who starts generating a plan by an adequate task division of ψ into a sequence of subtasks $(\psi_1, \psi_2, \ldots, \psi_n)$. These subtasks are compared with the individual capabilities and opportunities that the agents of group G1 have. All the agents from group G1 have to take on the belief that these subtasks $(\psi_1, \psi_2, \ldots, \psi_n)$ lead to the main goal ψ, $BEL_i(\text{realize}(<\psi_1, \psi_2, \psi_3, \psi_4, \psi_5>, \psi))$.

Dialogue types: *Deliberation(persuasion(inquiry)),Deliberation(persuasion(Information seeking))* .
Speech acts: *point of research, (ASS(ert), REQ(uest), CHALL(ange), CONC(ede), ...).*

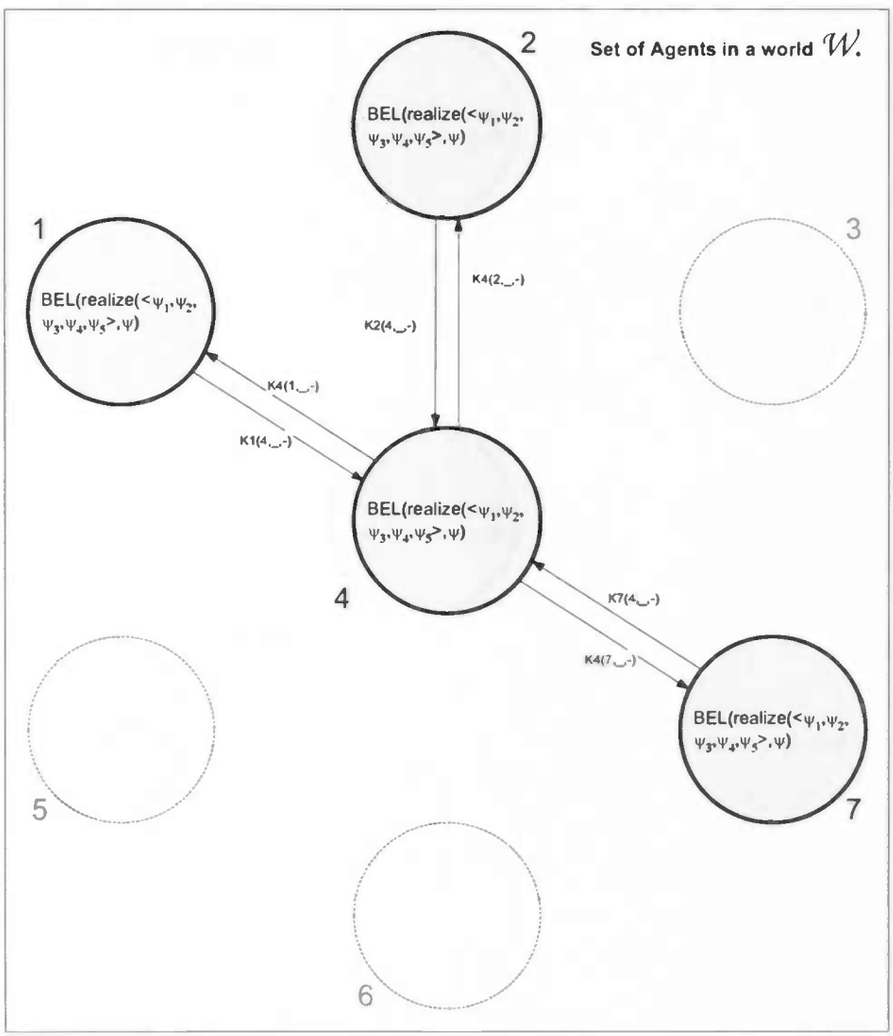

Figure A.8: Plan formation, in progress (1)

Level 3: Plan formation *(In progress – Task division(2))*
All agents of group **G1** believe that the sequence of subtasks ($\psi_1,\psi_2,\psi_3,\psi_4,\psi_5$) lead to the main goal ψ, BEL(realize(<$\psi_1,\psi_2,\psi_3,\psi_4,\psi_5$>,$\psi$). Initiating agent-4 communicates this fact to all agents of group **G1** through *one on group* communication which leads to the collective belief that the subtasks ($\psi_1,\psi_2,\psi_3,\psi_4,\psi_5$) lead to the main goal ψ, (C-BEL$_G$(realize(<$\psi_1,\psi_2,\psi_3,\psi_4,\psi_5$>,$\psi$)).

Dialogue types: *end fase of Deliberation(Persuasion) dialogue.*
Speech acts: *ASS(ert).*

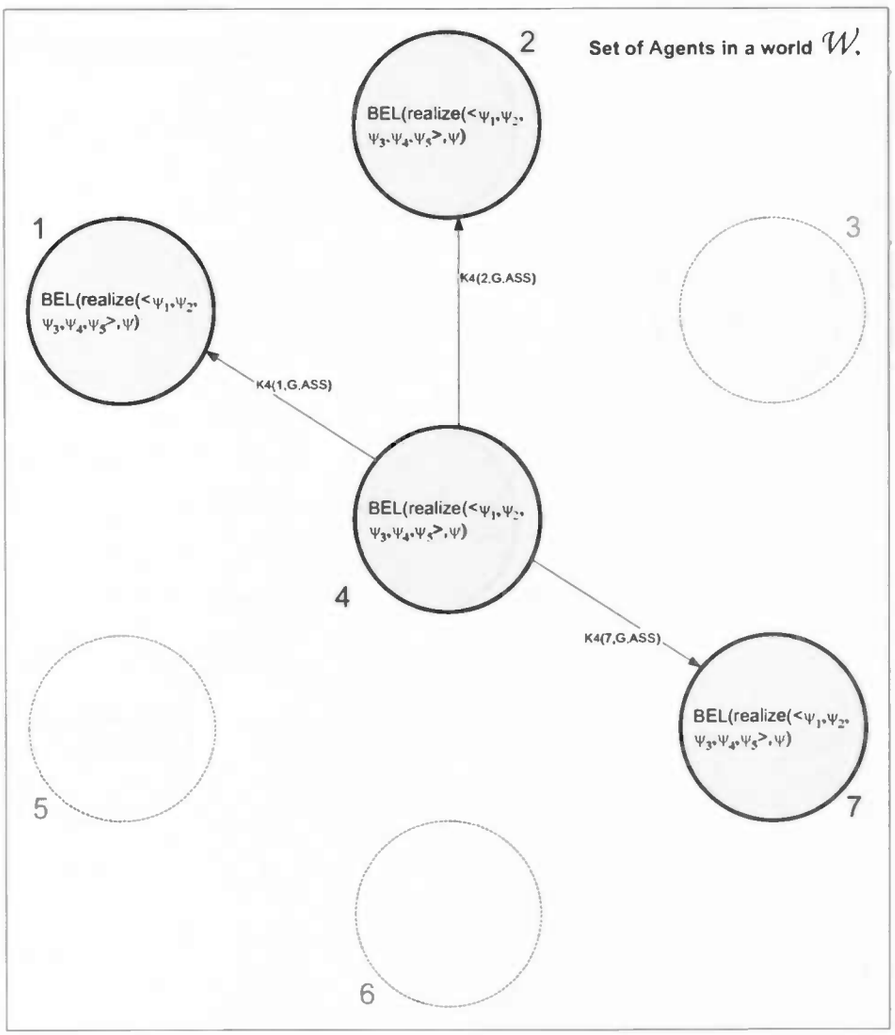

Figure A.9: Plan formation, in progress (2)

Level 3: Plan formation *(In progress – Means-end analysis(1))*
Any agent (here we use agent-4) from group G1 can act as initiating agent to associate actions to the subtasks. Each subtask $(\psi_1,\psi_2,\ldots,\psi_n)$ is associated with with one action δ_i. A sybtask can also be associated to more than one action but for reasons of convenience here only one action per subtask is presented. All the agents from group G1 have to take on the belief that executing the actions $(\delta_1,\delta_2,\ldots,\delta_n)$ realize the subtasks $(\psi_1,\psi_2,\ldots,\psi_n)$, $BEL_i(\wedge^n_{i=1}$ means-for$(\delta_i,\psi_i))$. The initiating agent-4 communicates to the members through *one on one* communication

Dialogue types: *Deliberation(Inquiry).*
Speech acts: *point of research* ***(REQ(uest), ASS(ert),*** *...).*

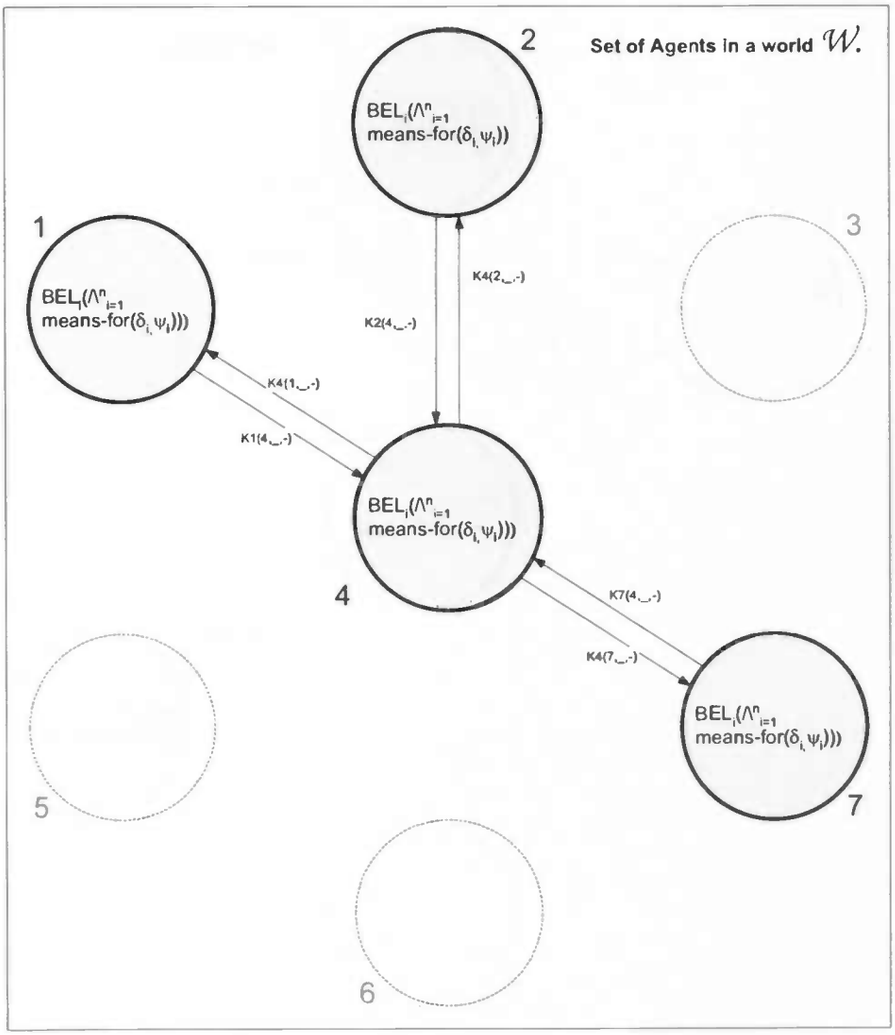

Figure A.10: Plan formation, in progress (3)

Level 3: Plan formation *(In progress – Means-end analysis(2))*
All the agents from group **G1** belief that executing the actions $(\delta_1,\delta_2,\ldots,\delta_n)$ realize the subtasks $(\psi_1,\psi_2,\ldots,\psi_n)$, $BEL_i(\wedge^n_{i=1}$ means-for$(\delta_i,\psi_i))$. Initiating agent-4 communicates this fact to all agents of group **G1** through *one on group* communication which leads to the collective belief that executing the actions $(\delta_1,\delta_2,\ldots,\delta_n)$ realize the subtasks $(\psi_1,\psi_2,\ldots,\psi_n)$, C-$BEL_G(\wedge^n_{i=1}$ means-for$(\delta_i,\psi_i))$.

Dialogue types: *end fase of Deliberation(Inquiry) dialogue.*
Speech acts: *ASS(ert).*

Figure A.11: Plan formation, in progress (4)

Level 3: Plan formation *(In progress – Action allocation)*
Any agent from G1 can act as initiating agent to coordinate the allocation process of the actions. Each of the actions $(\delta_1, \ldots, \delta_n)$ are allocated to an agent *j*, which results in pairs $<\delta, j>$. Agent *j* socially commits itself to another agent *k* to perform this action resulting in COMM(*j*,*k*,δ). Either agent *j* or *k* communicates a succesfull outcome to the initiating agent of this stage. In this case there are 7 subtasks with 7 actions associated with them.

Dialogue types: *Deliberation(Inquiry(Negotiation)), Deliberation(Negotiation(Inquiry, Information Seeking)).*
Speech acts: *point of research, (ASS(ert), REQ(uest), CHALL(ange), CONC(ede), ...).*

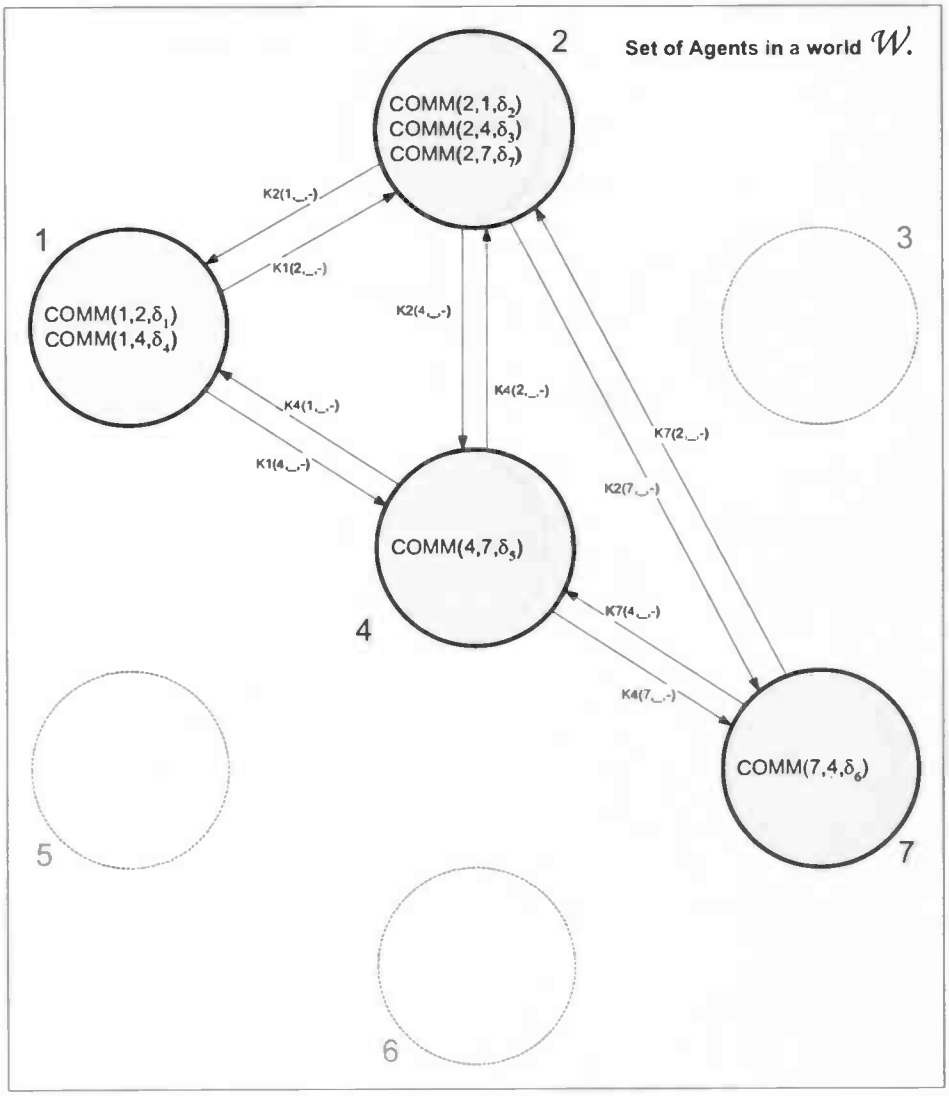

Figure A.12: Plan formation, in progress (5)

Level 3: Plan formation *(Outcome situation)*
Outcome: All agents of the group G1 have the right beliefs, intentions and commitments to achieve the overall goal ψ, $COMM_{G,P}(\psi)$. The initiating agent-4 communicates this to the group through *one on group* communication through which the desired collective commitment emerges, C-$COMM_{G,P}(\psi)$.

Dialogue types: *end fase of overall dialogue type Deliberation.*
Speech acts: *ASS(ert).*

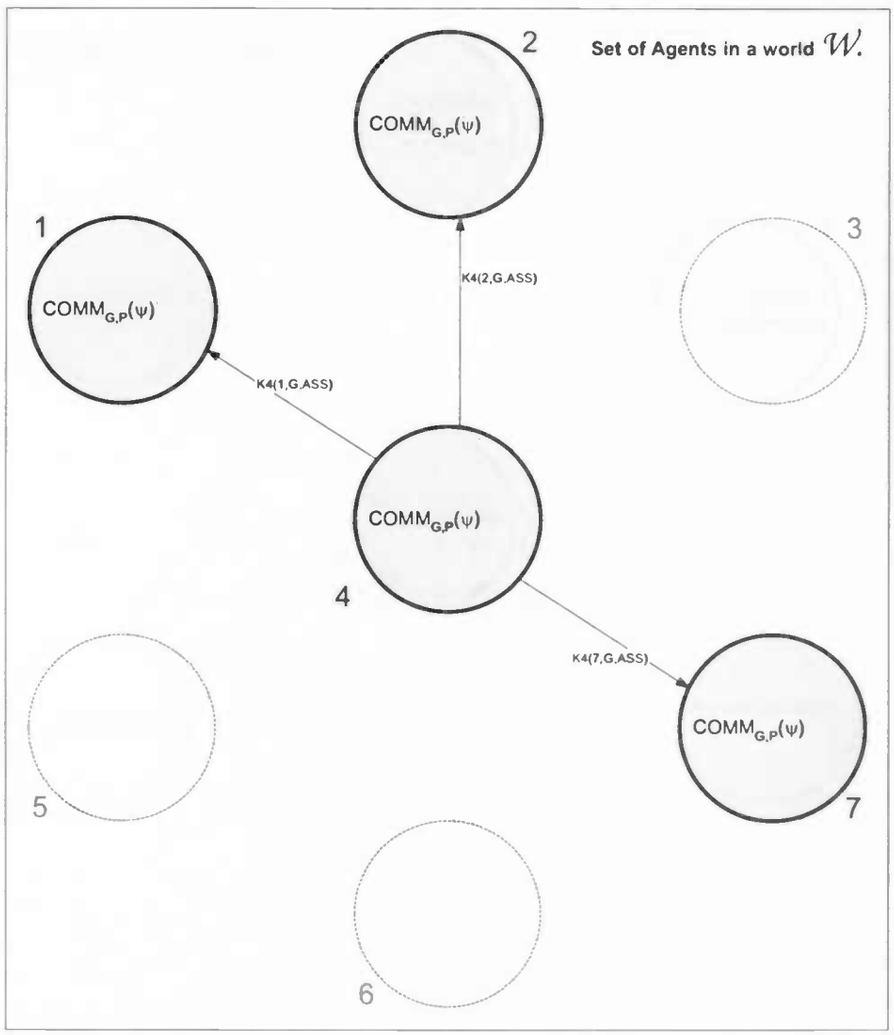

Figure A.13: Plan formation, outcome situation

Level 4: Team Action *(Initial situation)*
Intitial situation: A collective commitment of group G1 based on a plan P. There are al lot of different kinds of collective commitments. A common kind is the strong collective commitment SC-$COMM_{G1,P}(\psi)$ wich will be used here .
Outcome: The actions $(\delta_1,\delta_2, \ldots ,\delta_n)$ associated to the subtasks $(\psi_1,\psi_2, \ldots ,\psi_n)$ have been carried out by the agents of group G1 who were committed to do them. By the succes of these actions the overall goal ψ has been achieved.

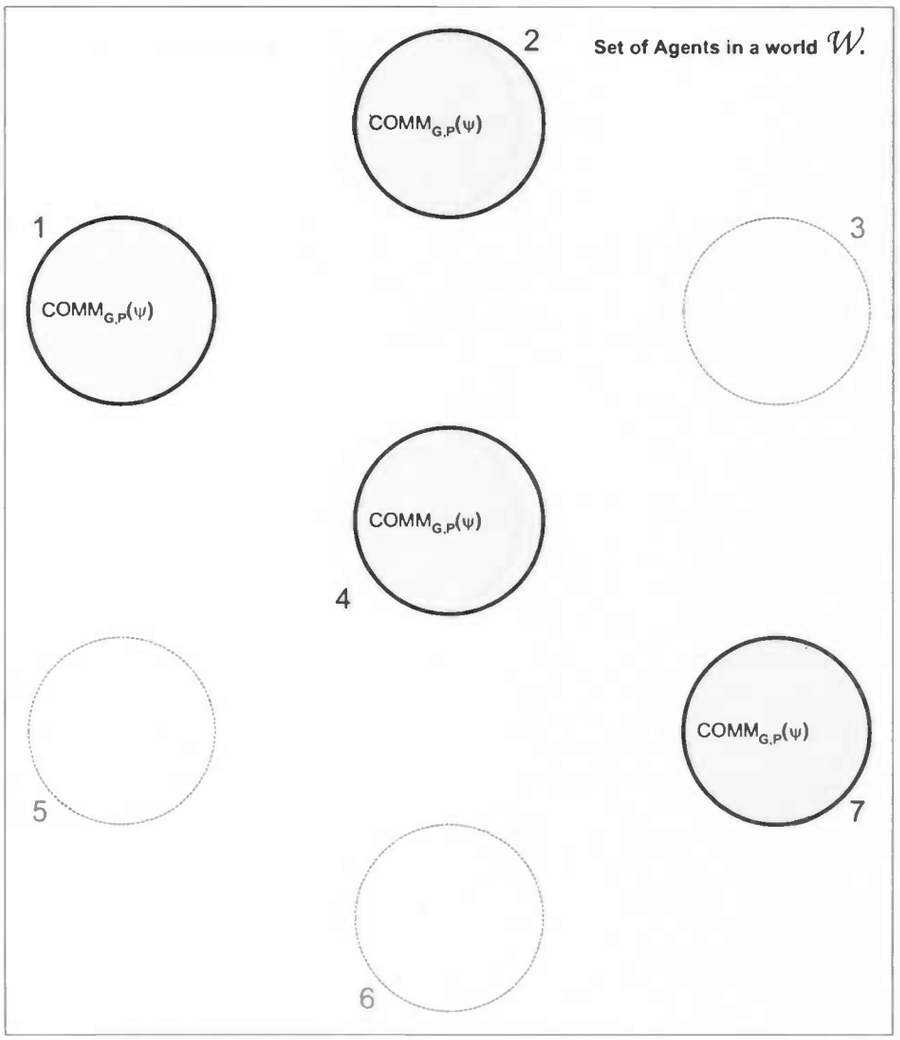

Figure A.14: Team action, initial situation

Level 4: Team Action *(In progress)*
The actions ($\delta_1, \delta_2, \ldots, \delta_n$) associated to the subtasks ($\psi_1, \psi_2, \ldots, \psi_n$) are being carried out by the agents who did commit themself to execute these actions. Agents can communicate to each other through *one on one* als well *one on group* communication.

Dialogue types: *All types possible, depending on the domain and the collective commitment.*
Speech acts: *Dialogue type dependent.*

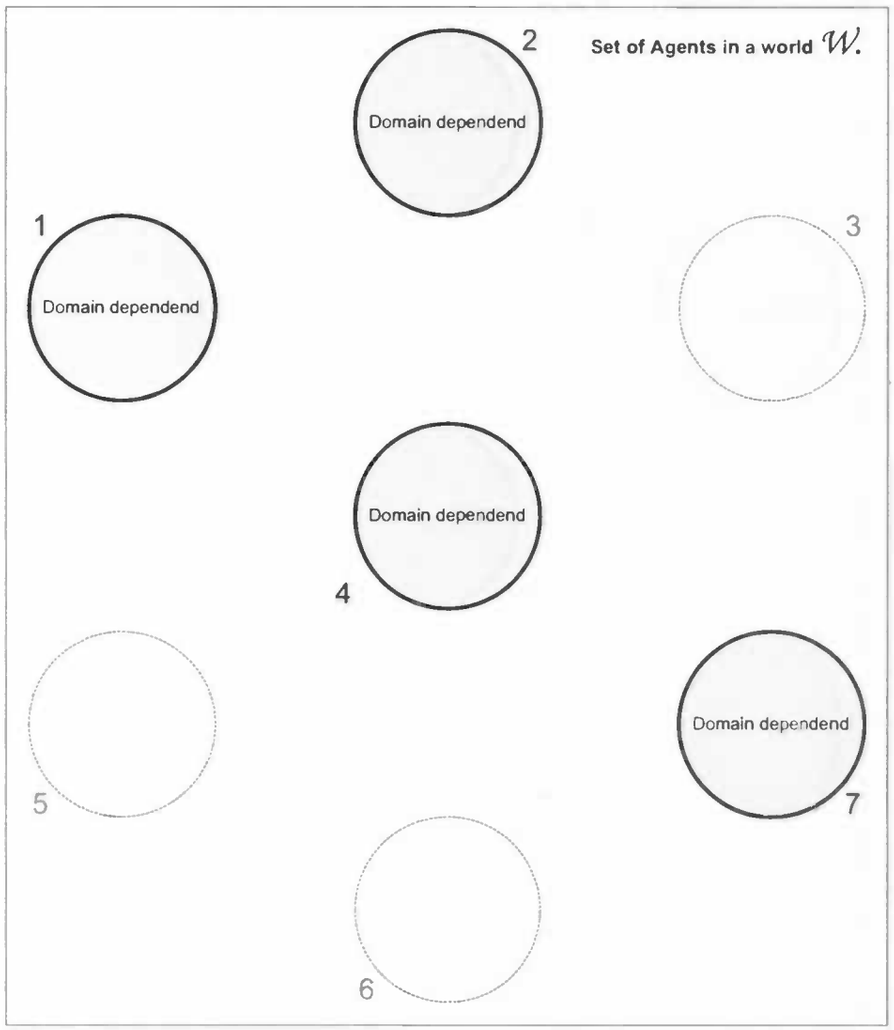

Figure A.15: Team action, in progress

Level 4: Team Action *(Outcome)*
Outcome: The actions (δ_1,δ_2, . . ,δ_n) associated to the subtasks (ψ_1,ψ_2, . . ,ψ_n) have been carried out by the agents of group G1 who were committed to do them. By the succes of these actions the overall goal ψ has been achieved.

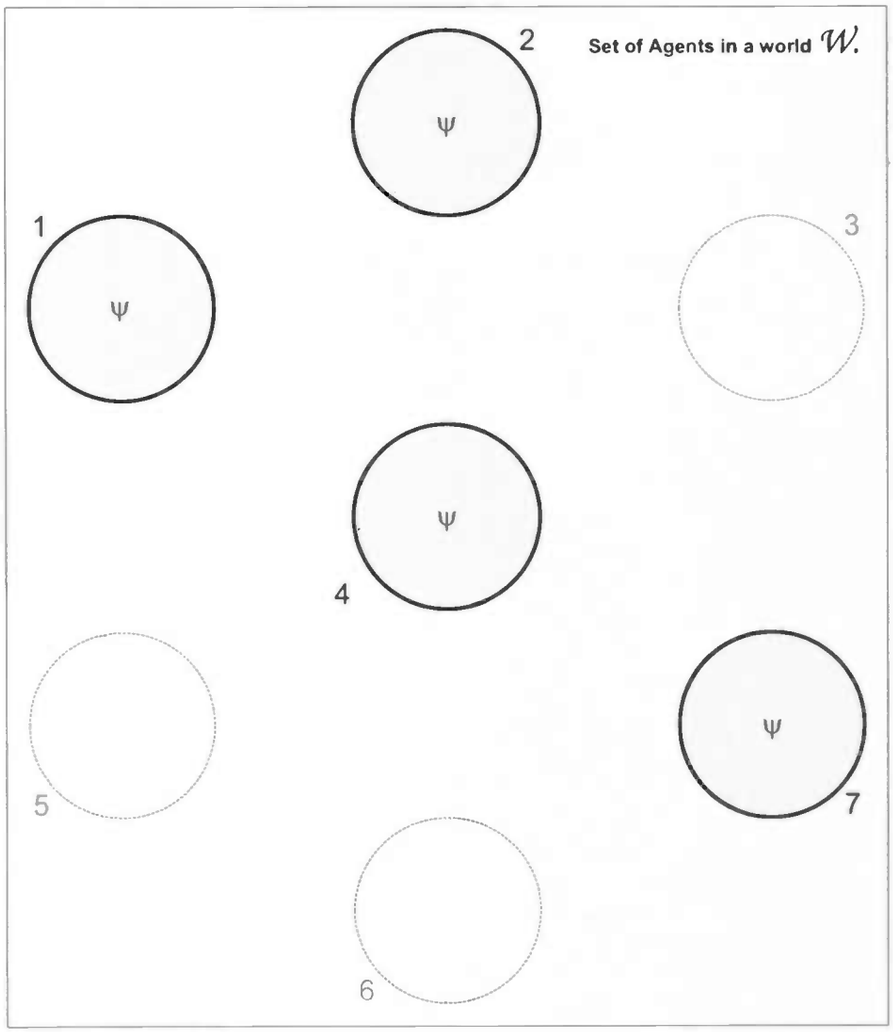

Figure A.16: Team action, outcome situation

Knowledge-based Algorithm for Multi-agent Communication {Master Thesis}

E. van Baars, egon@vanbaars.com
Student number: s0981254

August 29, 2006

Internal advisor: dr. R. Verbrugge
[Artificial Intelligence, University of Groningen]
Referee and external advisor: ir. S. Achterop
[Computing Science, University of Groningen]
External advisor: H. Paas
[Computing Science, University of Groningen]

Institute of Artificial Intelligence,
University of Groningen,
Grote Kruisstraat 2/1, 9712 TS Groningen, The Netherlands