TR-0821

A Scheme for Cooperative Systems Communication

by

Stephen T.C. Wong

December, 1992

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Scheme for Cooperative Systems Communication

Stephen T. C. Wong*

Institute for New Generation Computer Technology (ICOT), Japan

Abstract

The purpose of this paper is to present the design principles and features of a novel communication scheme that has been used to support **cooperative problem solving** within a network of knowledge-based systems. In presenting this account, we attempt to answer a set of critical yet unsettled questions of **cooperative systems communication** such as when and how a knowledge-based system knows which message to send, what the conditions of success are for a message, and how individual systems cooperate with each other for different purposes. To achieve this, we first propose two key design principles: (1) the **loose coupling** of communication issues and knowledge representation issues, and (2) the notion of **communicative acts**. We then work these ideas into the communication scheme COSMO, whose key features include: knowledge handlers, an operation model, organizational roles, message types, communication strategies and protocols.

1 Introduction

With the steady progress of research in computer systems over the past decade, it is now clear that there are many classes of complex problems which cannot be solved in isolation. Technological advances in communication networks, however, have opened up many new vistas for cooperative interaction across several computational systems or processes for solving such problems. Broadly speaking, a cooperative problem solving (CPS) system refers to several loosely connected and potentially heterogeneous agents that cooperate together to solve problems that require their combined expertise and resources.

This paper is concerned with a particular class of CPS systems known as cooperative knowledge-based systems (CKBS). In this context, a nodal knowledge-based system is

*This work is based on the author's research done at the NSF sponsored Engineering Research Center: Advanced Technology for Large Structural Systems (ATLSS) Center at Lehigh University, PA, USA. The author would like to thank John Wilson, Donald Hillman, Fabian Donze, and Toru Ishida for their helpful comments. The writing of this paper is also supported by research award No. 191101 from the Science and Technology Agency of Japan and grant No. INT-9123128 from National Science Foundation of the USA.

an autonomous agent, i.e., it asserts control over its own (local) resources. CKBS is rooted in distributed Artificial Intelligence, and recent surveys on the activities of this proliferating field can be found in [18, 23, 17]. However, owing to its potential for wide-ranging applications, CKBS also attracts researchers from many non-AI areas such as those dealing with databases, computer supported coordinated work, and management information systems [15, 12, 14, 30, 41, 8].

Communication is a critical component of cooperative problem solving systems. It is a process that involves two or more agents sending messages to one another over a network. However, simply passing data or calling procedures among a group of agents does not make that group cooperative. Cooperative computation requires schemes of communication that are more sophisticated than conventional ones in order to address the following issues:

(I1) **Heterogeneous agents**: Cooperating agents often differ in internal structure, information content, and inference ability. Usually, they are preexisting systems for single applications and are integrated to handle more complex applications. This causes several problems. First, the specifications encoded in messages are often not in a compatible form and thus are not mutually understandable. Second, consistent viewpoints and knowledge among agents are difficult to achieve due to discrepancies in knowledge and inference ability.

(I2) **Knowledge interchange**: The world explored by a cooperating agent includes the complex and changing phenomena of other agents, such as their behavior, beliefs, intents, and so forth. For example, an agent needs to negotiate with others to resolve conflicts due to inconsistencies in their knowledge. Or, an agent may want to persuade another party to put a high priority on the tasks requested of it. Thus, cooperating agents must be able to communicate such relevant knowledge rather than just pure data.

(I3) **Localized control**: A CKBS agent typically retains its **autonomy** to serve its existing users. That is, it asserts full control over local information and processing, and may temporarily relinquish part of the control for cooperative purposes. The autonomy means that the behavior of such agents stems from neither a centralized scheduling scheme nor predefined interaction. The agents must possess significant expertise in developing their own strategies of communication dynamically, without the benefit of a global perspective. In contrast, such strategies are generally static in conventional distributed systems.

(I4) **Organizational structure**: To refrain from overloading the computing resource for communication, agents must avoid excessive communication and keep the interactions between them under control. Often, a group of agents achieves this with the aid of an organizational structure which defines their roles, behavior expectation, and authority relations [34, 24]. This kind of structure also enables us to incorporate many effective human organization techniques into computer systems.

Furthermore, it is worth noting that computational speed-up may be a favorable side-effect, but is not the primary objective of cooperative problem solving systems. This sets apart CPS projects from other projects that solely aim to increase the computation speed through parallel or distributed processing of AI programs [6, 27, 60].

The purpose of this paper is to present the key design principles and features of COSMO – a communication scheme to support orderly and rational interaction among cooperating agents. This scheme originated from the development of Building Design Network (BDN) in the ATLSS Engineering Research Center at Lehigh University [3, 54]. BDN is a distributed knowledge-based system prototype in which several agents with different construction expertise and responsibilities (and possibly their users) cooperate together to obtain the preliminary design of structural buildings. Agents of this application reside in separate UNIX workstations and use TCP as the communication backbone. The knowledge bases are written predominately in an extended Prolog language [55, 56]. Aside from practical applications, we present this account in an attempt to offer new insight into the challenging issues of communication, I1 to I4, as stated earlier in this section.

This paper is organized as follows. Section 2 introduces two important principles of designing communication schemes: the loose coupling of communication and knowledge representation issues and the notion of communicative acts. The subsequent sections work these principles into a working scheme, COSMO. Specifically, Section 3 discusses four key components of the scheme: the operational model, organizational roles, message types, and communication and computation steps. Section 4 presents a set of strategies which are built on these components for general control in communication. Section 5 describes some of the communication protocols devised to convene intentions that are more complex than single communicative acts. Furthermore, Section 6 discusses related work by others in inter-agent communication. The final section concludes this phase of research and suggests possible future work.

2 Guiding principles

This section discusses two basic principles of designing communication schemes in cooperative problem solving systems. The first principle is to **loosely couple** the study of communication issues with the study of the representation issues of individual agents. The second principle is to bring in the notion of **communicative acts** for coordinating actions and exchanging knowledge among agents.

Knowledge representation and inter-agent communication are hard problems. It is better to deal with them individually rather than together in the early stage of system development. The first principle proposes the use of a **global language** of communication between heterogeneous agents. The use of a global language makes fewer assumptions (thus it enables loose coupling) about how knowledge is represented in individual agents (issue I1). In other words, at the communication level, all agents should view knowledge residing in one another as being represented in a single, uniform scheme. Another advantage is that it enables the integration of preexisting, autonomous knowledge-based systems for broader applications (issue I3). This also protects an organization's investment in local knowledge base management software, application programs, and user training.

The notion of a global language is a major approach in designing heterogeneous information systems. The ability to translate existing code into a desired format is a critical key to making it shareable. The global schema approach, for example, was the first to be used in multidatabase design and continues to be a popular choice in that field. The global schema is another layer, above the local schemas, that provides global data independence. The choice of a global schema would depend on the multidatabase designer [30]. Recently, the Interlingua Working Group of the Knowledge Sharing Initiative, headed by Richard Fikes and Mike Genesereth, took up an ambitious project to develop a common language, the Knowledge Interchange Format (KIF), along with a set of translators to map into and out of the language from existing knowledge representation languages [36]. KIF is supposed to a superset of all agent languages.

The practicality of using a global language or schema is mainly a matter of the tradeoff between writing translation code and limiting participation. If the developers are willing to write enough translation code into the translators (considering development costs and execution efficiency), the cooperative system can accept a wide variety of heterogeneous knowledge bases. Another consideration is the difference in expressive power between the global language and local languages. If the effort of writing code to get around the discrepancy between a local language and the global language becomes a burden, that knowledge base is unlikely to be integrated into the cooperative system. (On the other hand, if the Interlingua group succeeds, this discrepancy problem would then vanish.) Note, the coding of a translator must also take care of the impedance mismatch of encoded knowledge, e.g., differences in terms, names, and data format [30, 52].

To support a global communication language, an agent, thus, should contain a **knowledge handler** for coding and decoding messages, in addition to a reasoning program (i.e., knowledge base and inference engine). In COSMO, such a knowledge handler is an independent computation process and normally runs as a background process, unless the local user explicitly invokes it to see its content. A handler also performs other crucial functions, such as checking against possible errors and keeping track of messages that are pending for reply. We will return to this point in Section 4.

This coding model of communication, that is, the use of knowledge handlers to facilitate the interface among heterogeneous agents, is still not flexible enough for the interchange of knowledge (issue I2). Cooperating agents must be able to communicate in a more verbalized and regulated way, and to convene various intentions during cooperation. In this respect, the **speech acts theory** [44, 45], a thriving branch in the philosophy of language and linguistics, offers some new insight. We describe the part of the theory that influences the design of our communication scheme below.

Speech acts theory states that the primitive units of human communication are speech acts of a certain type called **illocutionary acts**. Some examples of these are statements, questions, commands, promises, and apologies. Whenever a speaker utters a sentence in an appropriate context with certain intentions, he or she performs one or more illocutionary acts. In general, an illocutionary act consists of an illocutionary force F and a propositional content P. For example, the two utterances "You will leave the party" and "Leave the party!" have the same propositional content, namely that you will leave the party; but the first of these has the illocutionary force of a prediction and the second has the illocutionary force of an order. On the other hand, the two utterances "Are you going to the beach?" and "When will you see Sandie again?" both have the illocutionary force of questions but have different propositional contents.

A special class of sentences which express elementary illocutionary acts of form F(P) are the performative sentences. These sentences consist of a performative verb used in the first person present tense of the indicative mood with an appropriate complement clause, i.e., a propositional content. In uttering a performative sentence one performs the illocutionary act with the illocutionary force named by the performative verb by way of representing oneself as performing that act. Some examples (with the performative verbs italicized) are: "I *promise* that I will do it tomorrow", "I *order* you to report the schedule to the project manager", "I *offer* that x is a better choice than y."

Not all illocutionary acts are of the simple F(P) form. More complex cases are called **complex illocutionary acts** which are composed of simple acts using connectives such as "and" and "but". Thus, in a certain context, by uttering "I will go to her house, but will she be there?" a speaker both makes an assertion and asks a question, i.e. the form is $F_1(P_1) \wedge F_2(P_2)$, and the successful performance of this complex act is a function of the successful performance of its constituents.

Now, for distributed systems computation, a basic unit of communication among agents is the transfer of a message from one agent (the sender) to another (the receiver). The purpose of communication is to provide the receiver with some information or to have the receiver take certain actions. Inspired by the speech acts theory, such a unit is called a communicative act.

The second design principle that we put forward is: a communication act is analogous to an **elementary** performative act in human communication; its message type τ expresses an illocutionary force F, named by a performative verb; and its message content p expresses a complement clause P, which is a specification of the sender intended to be computed by the receiver. Such a specification, as stated earlier, is represented in a format understandable by all agents. Note that we have not considered the notion of complex performative acts here.

Meanwhile, a message type τ has two basic functions. In the first place, it is an index for a receiver to select a procedure to compute the message content and to determine the type of the response act. Thus, the meaning of the performative verb that a type denotes is defined by the operation of the procedure associated with it. To maintain the consistency of its performative meaning, each of the message types defined in a cooperative system must indicate the same procedure across all agents.

Secondly, a message type has a name similar to the performative verb that it denotes. For instance, a communication act of type *order* would tell a receiver to do something without the option of refusal. This act assumes that the sender has a higher authority or priority than the receiver. An act of type *ask* would request the receiver to answer a question in the form which has already been determined by the propositional content of that question, e.g., a receiver either affirms or denies a yes-no question. We believe that the use of such a logical naming convention to represent the concepts and structures inherent in a communication scheme impacts on the very thinking that goes into constructing that scheme. This is similar to the historical fact that high-level languages, such as Pascal and C, have a profound influence on a programmer's ability to conceive efficient algorithms.

The simple illustration in Figure 1 may help us to better understand the operation of a communication scheme based on the aforementioned ideas. Let us suppose that there are two agents with a sequential processing ability, and their language of communication has two message types: *ask* and *answer*. Let us suppose further that there is an entity x in the real world which is denoted as p in one reasoning program, as p' in the other, and as p" in the common language. We show the thread of control in a query process between the two agents in Figure 1. (Italics are used to denote message types in this paper.) The operational steps shown in Figure 1 are described in the following:

1. The local reasoning program invokes a query about x (represented as p? in the figure).
2. The local handler encodes the query into an outgoing message, *ask*(p"), where *ask* is called a message type.
3. The remote handler decodes the message, identifies its type, transforms its content p" into p', and passes such information to the remote program.
4. The remote program identifies an internal operation t, and executes t to compute p'.
5. The result of computation r' is sent to the remote handler.
6. The remote handler composes a response message *answer*(r"), where r" denotes the same thing in the real world as r', and *answer* is a response type of query.
7. The local handler converts r" into r, and sends it together with the message type, *answer*, to update the local program.

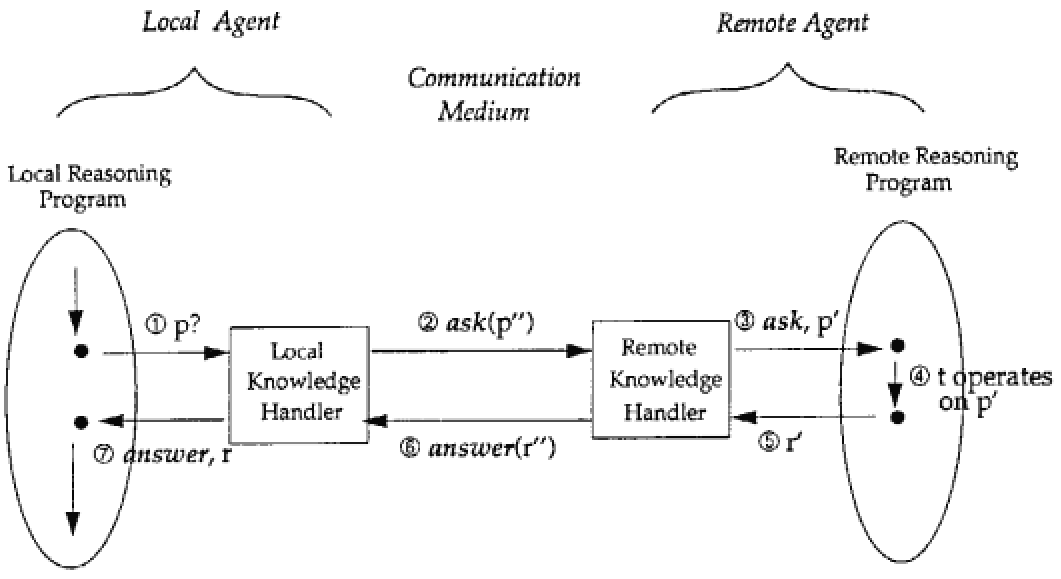

Figure 1: The thread of control between two agents in a simply query

These two principles, though promising, have yet to be worked out in detail for practical applications. A closer look at the above example reveals many unsettled questions such as the following:

(Q1) When and how should a cooperating agent know which act to perform (issue I3)?
(Q2) What are the conditions of success of a communicative act (issue I2)?
(Q3) What is the relation between organizational roles and communicative acts (issue I4)?
(Q4) How does an act relate to other acts (issues I2 and I3)?
(Q5) Can more complex intentions be convened by a few primitive acts, and if so, how (issues I1 and I2)?

In the subsequent sections, we incrementally present the key features of COSMO. This scheme formalizes and extends the principles discussed, and, thus, provides some answers to these challenging questions.

3 Basic Components of COSMO

The purpose of this section is to introduce the basic components of our scheme. These components are the building blocks for the strategies and methods of communication to be presented in Sections 4 and 5.

3.1 Operational model

In COSMO, we consider two major kinds of communicative acts: those used to initiate actions and those used to respond to these actions. For an agent to know when to initiate an act A (see question Q1) and whether that act is successful (question Q2), we use the following **operational model** of communicative acts:

$$\{\text{precondition}\}\text{: A: }\{\text{postcondition}\}.$$

In this model, {precondition} denotes a set of internal constraints which must be satisfied before A can be performed, and {postcondition} denotes a set of conditions which must be met in order to consider that A is successful. Such a model is similar to the representational formalism used in most plan recognition algorithms, and which originated in the earlier research on AI planning in STRIPS [20] and NOAH [43]. The representation formalism specifies that a precondition needs to be true to carry out the planning operation (a communication act in our case) and an effect that holds once the operation is accomplished. The model of COSMO extends this formalism into distributed systems communication and enriches it with the concepts of the degree of strength and the classes of communication acts.

For {precondition}, usually, it is the reasoning program of the agent which decides the act that is performed at a particular stage of operation. However, sometimes a reasoning program can have several alternative actions, and deciding which one to perform becomes a problem (again, a problem related to Q1). For example, suppose that an agent wants to know something about p, but does not know which agent it should ask and is reluctant to broadcast the request in order to avoid excess communication. Its knowledge handler would then assign every feasible alternative a number and select the alternative with the highest number. Such a number is called a **utility** value. In the next section, we describe certain heuristics for setting utility values in the knowledge handler for general communication purposes.

Furthermore, the other constraints in {precondition} include:

- **Honesty**: An act must be compatible with the internal state or knowledge of the sender. For example, an agent will not ask a question when it already has the answer, or it will not make a statement that it knows is false. This concerns the truth condition of an act and is noted as an honesty condition for cooperative systems communication (refer to Q1).
- **Propositional Content:** This constraint also relates to Q1. It states that every message type may be subject to a "propositional content condition" other than honesty. The recognition of propositional content as an aspect of communicative acts echoes Austin's notion [5] that every genuine speech act has a **locutionary** component, not

just those that vouch for the truth of the representation as required in the honesty constraint. We will illustrate this point with examples in Section 3.3.

- **Role**: A CKBS in this scheme is also an organized information system. Thus, the message type must satisfy a specific role constraint between the sender and the receiver in the organization (see question Q3). The role constraint relates to the priority of the message as well as dedicating the communication behavior of the two agents. We discuss this constraint in the next subsection.
- **Accessibility**: The sender must be able to access the receiver. This accessibility requires not only the support of a reliable communication network but also the use of knowledge handlers to ensure compatibility of the two agents at the software level (question Q2).

To terminate an act A properly, its {postcondition} requires that the sender must receive a message that either is a direct response of defined types to A or is a control message indicating certain communication problems. We also present certain functions of control messages in Section 4.

However, the scope of this operational model is limited in the sense that it manages isolated communicative acts. It neither tells us how to relate one act with many other acts in a discourse among agents (question Q4), nor does it enable agents to communicate intentions that are more complex than simple communicative acts (question Q5). Before addressing these hard questions in Section 5, we first need to introduce three other basic components of COSMO: the organization roles, the message types, and the communication and computation steps.

3.2 Organizational roles

Organization hierarchies are used in cooperative problem solving systems to: (1) establish the problems to be solved; (2) segment the problems into separate activities to be performed by different agents; and (3) coordinate activities and tasks among agents so that overall solutions are achieved. Depending on the application, (1) and (2) may be pre-determined or may be jointly decided by the agents in the course of communication.

Generally, there is a set of admissible roles in such a cooperative system, and each agent in the system is assigned one of these roles. The function of the roles is to indicate the position of that agent in the hierarchy and to determine what reasoning strategies to use. To compare the ranking differences of agents, this proposed scheme assigns a number to every role. For example, for two agents, a of role_a and b of role_b, agent a ranks higher than b if and only if (iff) $v(\text{role}_a) > v(\text{role}_b)$, where $v(\text{role}_x)$ denotes the role value of an agent x.

In the current architecture of COSMO, the information of the organizational structure is encoded in a **role table**, which contains information about all agents' roles and their role values in the organization at a particular stage of operation. Since agents' reasoning programs and their knowledge handlers access the information of the role table frequently, we allocate a copy of the role table in every computational process for efficiency. Whenever the underlying organizational structure is changed, all copies of role tables in the cooperation system will be updated immediately. As will be shown later, this synchronization of role tables is necessary for the proper operation of the scheme. This arrangement also assumes that the organizational hierarchy of particular applications is reasonably stable. Otherwise, the frequent broadcast of updated role tables would overload the communication network.

The ranking difference between any two people in an organization affects their interactive behavior, such as decision making and communication. Any cooperative computing system which claims to exhibit certain human problem-solving abilities should exhibit such adaptive behavior. One way to accomplish this, as is implemented in our prototypes, is to have every agent partition its set of problem-solving strategies into several classes. An agent selects a particular class of strategies based on the ranking difference between itself and the would-be receiver. As an example, suppose that the set of strategies of agent a is $S_a = s_1, \cdots, s_n$, where the subscripts are the indices of individual strategies. Then, agent a chooses s_i in S_a to interpret an incoming message from agent b when $v(role_a) - v(role_b) = i$.

In this way, one can say that an agent has several classes of problem-solving strategies in its reasoning program and switches among them according to which agent it communicates with. The subsection below provides an example on the use of organizational roles to decide the types of communicative acts.

3.3 Message types

In COSMO, agents use two disjoint sets of message types: one set contains types that are strictly used in communicative acts for initiating actions, and the other contains types that are strictly used in response to the former acts. Hence, these response types are a postcondition of the acts of the first types. We call the messages in the first set, **action messages**, and those in the second set, **response messages**.

COSMO classifies message types according to the performative intents or purposes of their associated acts. It further distinguishes the types of a class to indicate the role relationship between the sender and the receiver. BDN application, for example, has three classes of message types: inquiring, informing, and complaining (see Figure 2). Inquiry messages have two uses. To inquire is either to query for information or to request some action. Informative messages are both assertive and directive. To inform is either to give out information or to instruct someone to do something. The content of an informative message must be in grounded form, that is, without variables, while there is no such restriction for inquiry messages (the propositional content constraint in Section 3.1). Complaint messages are used to express one's dissatisfaction. To express dissatisfaction with a state of affairs

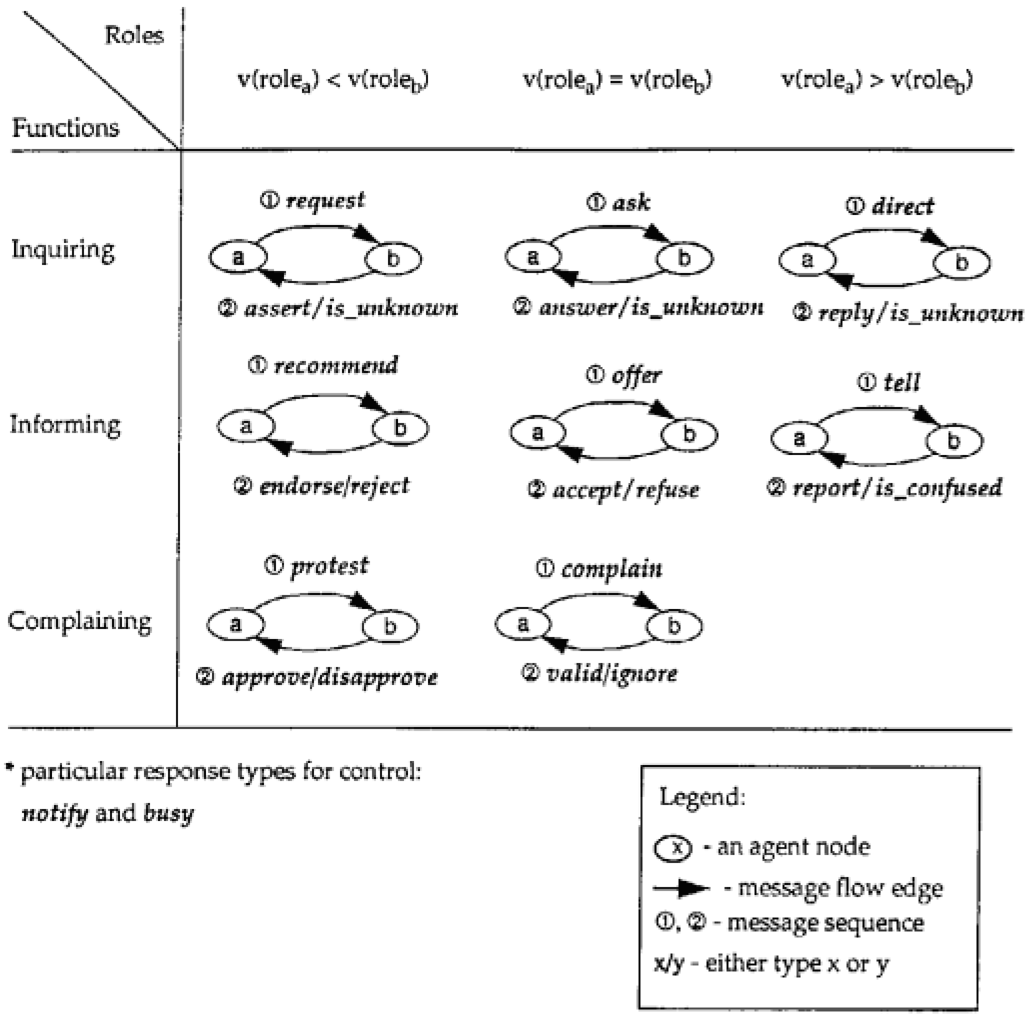

Figure 2: Communicative acts currently implemented in COSMO for BDN.

commits the sender to presuppose both the existence of that state of affairs and that this particular state is bad for the sender. Complaining acts are used to initiate negotiations among agents (the propositional content constraint of complaint acts, see Sections 5.2 and 5.3).

Different communicative acts can sometimes achieve the same performative function with greater or lesser degrees of strength, e.g. suggesting that the receiver abort the task is weaker than ordering it to abort the task [45]. Following the discussion in Section 3.2, we have a role relation between the two communicating agents that dictates this degree of strength in COSMO. Here, we, in turn, use appropriate message types to indicate role relations explicitly. Figure 2 illustrates this point with a set of communicative acts that are currently implemented for BDN application.

In this section, we briefly comment on the general usage of these message types. Let degree(τ) represent the characteristic degree of strength of an act of action type τ. For inquiring acts, as an example, we have degree(*direct*) > degree(*ask*) > degree(*request*), as indicated by the role relations of the sender and the receiver. Such a comparision of message types has many implications in cooperative systems communication. For instance, an act of *request* or *ask* type lets the receiver know that the sender is either of the same rank or a lower rank. Thus, the receiver can grant or refuse the inquiry by returning messages with either an assert or answer type. However, in a *direct* act, such a refusal is precluded as it is coming from a higher authority (see Figure 2). Or, to resolve the ordering conflict of messages, when *direct*(p) and *ask*(p') arrive at the same time from two remote agents, the local agent would execute the former message first.

If an inquiry could not be understood, that is, it is not computable, then the receiver would simply send back an *is_unknown* message. Similarly, for informative acts, an agent can accept or refuse an act of type *offer* or *recommend* but must follow the instruction of a *tell* act. When the content of an informative message is not computable, a response act of type *is_confused* will be sent instead.

Complaining acts are used by the sender of a lower or an equal rank to seek the approval of a receiver for starting negotiation. COSMO does not define any complaining act for a sender of higher authority; such a sender can simply tell lower ranking receivers to start negotiation right away. However, to maintain the mode of cooperation (assuming that this mode can achieve better results than the others), we must avoid abuse of the positional power in communication. In Section 5, we discuss some way to deal with such problems. Moreover, types *notify* and *busy* can replace any of the response types for specific control purposes. Section 4 provides some examples on the use of these two types.

In addition, the set of message types listed in Figure 2 is a **minimum set**, in the sense that the intention or communicative function of any message type cannot be replaced by any other or any combination of other types. On the other hand, a more complex kind of intention, such as bargaining, to be discussed in Section 5, may be convened by several protocols composed of these types. Note that the current classification of message types in COSMO covers only a subset of speech acts in Searle's five illocutionary categories [45].

3.4 Communication and Computation Steps

A communicative act of one agent may spawn many other acts among agents; for instance, agent *a* asks *b* about p, *b* then asks *c* about p, and so on. A process of communication among agents is a sequence of communication and computation steps. A communication step encodes the specification of a sender into a message and sends it out to a receiver, i.e., performs a communicative act. A computation step changes or updates an agent's reasoning program, and when necessary, decodes an incoming message. Refer to Figure 1, for an example, Step 1 is a computation step in which the local agent desires to know about x;

Steps 2 and 6 are communication steps including questions and answers; and Steps 3-5 can be grouped into one computation step involving the decoding of an action message and internal computation of the remote program.

A sequence of these steps terminates properly when the first sender of that sequence receives a response message to its initial act. To keep track of multiple sequences occurring simultaneously in a cooperative system, the first message of any sequence contains a unique label which will be included in all subsequent messages of the same sequence.

In particular, we are interested in those sequences with precisely defined steps. We call such a sequence a **protocol**. Protocols are regulated means to convene complex intentions in terms of a few elementary communicative acts. They enforce an order on the way that co-operating agents interact with one another. An agent which wants to initiate communication of a specific aim would select a particular protocol. The agent would then communicate that it is using this protocol to the other agents. This would then allow the intended receivers to select corresponding acts and to ignore irrelevant messages. In this way, the protocol also avoids excessive communication and keeps discussion among agents under control.

We use the following format to describe communication and computation steps. For simplicity, we do not express coding operations explicitly. We also assume that every step in COSMO is an indivisible operation in the sense that it runs from the beginning to the end without interruption.

A communication step whereby agent a invokes an act of type τ is expressed as: $a \rightarrow b \mid \tau$, p, l; where b is the receiver, the content p is a specification of a common format, and l is a unique identifier of the sequence. A computation step of agent a is written as, $a \mid \pi(\mathrm{p})$, where π is an internal operation on the content p of an act in the previous step. The global language used in our prototypes is of a logical form such that p can be a term, a list, a predicate, or a complex sentence, i.e., predicates connected by logical operators such as $\wedge$ (and), $\vee$ (or), and $\supset$ (implication). The transformation operations of such a handler also include the parsing of a message content into its atomic components for evaluation in the reasoning program, and the composing of evaluated results into a proper logical form for reply. The parsing and composing operations are described in Chapter 4 of [54]. For clarity in this presentation, we ignore the case where p is a complex sentence.

Figure 3 describes a typical login protocol between a remote host h and local computer c, of equal rank [58]. In this figure, f in Step 4 is a one-way function; i.e., given y, it is not feasible to find an x such that f(x) = y.

4 Communication Strategies of Handlers

This section presents some of the general heuristics used by knowledge handlers to manage message transactions. A strategy, in this context, refers to a decision-making procedure used

```
(1) $c \rightarrow h$  |  ask, login, $l$
(2) $h \rightarrow c$  |  answer, "okay, give me your password", $l$
(3) $c \rightarrow h$  |  offer, password, $l$
(4) $h$                |  compute y = f(password)
                       |  retrieve record($c$, f(password$_c$))
                          from user database
                       |  y = f(password$_c$) then accept;
                          otherwise refuse.
                       ...
```

Figure 3: A simple login protocol.

to observe and evaluate its environment and – in response to it – prescribe some immediate action. Some of the strategies simply make use of message types to convene their purposes. However, when several options are available, a knowledge handler would then apply an appropriate utility function to select the preferred act. A utility function would consider the general factors of the environment such as message types, organizational roles, and statistics of communicative acts. This is demonstrated in the communication strategies presented in this section.

These communication strategies are largely domain-independent, that is, weak methods in AI. They are operated by **default** when the reasoning programs lack domain knowledge to drive the communication process. Otherwise, system developers should supersede these strategies with more effective, but domain-specific, strategies from the chosen application. This is analogical to the case where a programmer would rather tailor an efficient algorithm to execute important programming tasks than use the general algorithms provided by the underlying operating system.

4.1 Which agent to notify

The handler of every agent records all messages into its **discourse table**. This table also includes other information such as time of transactions and the status of messages, i.e., whether a message is pending or has been responded to. An agent updates its discourse table whenever a message is sent or received. There are two ways in which a discourse table can be deleted. First, the local agent or user can issue a deletion command to the local knowledge handler. Second, an agent can remove the previous table entries from its system memory at the starting session of every new global problem that is unrelated to the previous one. The deleted entries are automatically stored in database files on the hard disk.

The information in a discourse table is frequently accessed for many purposes during the

(1) $a \rightarrow b$	*ask*, p(x), l_1
(2) b	don't know p(x)
(3) $b \rightarrow c$	*ask*, p(x), l_2
(4) c	don't know p(x)
(5) $c \rightarrow a$	*ask*, p(x), l_3
(6) a	find p(x) in discourse table pending the response of b
	notify c
(7) $a \rightarrow c$	*notify*, circular(p(x)), l_3 (circular is a predicate with p(x) treated as a term)
...	

Figure 4: An example of a circular inquiry.

communication process. One reason is to notify the occurrence of redundant acts in inquiry. A **redundant act** refers to an agent asking the same pending question to another agent twice. A **circular act** refers to some other agent asking the same question for which the local agent is also awaiting an answer.

Figure 4 shows a circular inquiry involving three agents of equal role (see also Figure 5 for a graphical illustration). In Step 1 of Figure 4, agent a asks b about a predicate p(x). But, in Step 2, b could not match that predicate with the set of clauses in its knowledge base. In Step 3, b then asks c about p(x). The latter does not know the answer either and asks a instead (Steps 4 and 5). The knowledge handler of a decodes the message from c in Step 6 and finds that it is still waiting for an answer to the same query. Thus, in Step 7, a notifies c about the circularity found.

In a closed computer system, agent a will automatically write off c in its future query about p(x). A cooperative knowledge-based system, however, is an open system. Agent c may know about p(x) through other agents or its user later on. An extension of this strategy in COSMO is that a still considers c in the future query, but will lower the priority of that agent in its list of possible candidates. This brings us to the next strategy.

4.2 Which agent to query

One of the purposes of the interaction between distributed agents is to query one another for missing information. Sometimes an agent may not know who has the answer to its question, and being afraid of overloading system resources, may be reluctant to broadcast the question

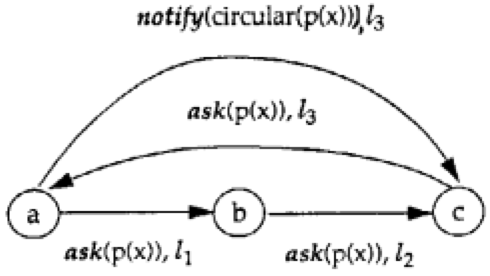

Figure 5: A graphical illustration of the circular communication in Figure 4.

to all other agents, or a large subset of them, in the network. In this case, it uses the following strategy in its handler to guess the most likely candidate for inquiry.

Let the local agent be denoted by a and a set of possible candidates to query be $Q_a = \{a_1, \cdots, a_n\}$. Normally, the local reasoning program is expected to have some domain knowledge to limit the number of agents in Q_a, e.g., the relevancy of an agent's expertise in relation to its inquiry.

Strategy 1

1. Form a linear ordering O_{role} of these candidates according to their role values.
2. Form a linear ordering O_{fq} of these candidates according to their normalized frequencies.
3. Calculate the utility value of any candidate $a_i \in Q_a$ based on the utility function U_1:
   $U_1(a_i) = W(a_i, O_{role}) + W(a_i, O_{fq})$, for $que_{a,a_i,t} \leq N_{th}$;
   $U_1(a_i) = WEIGHT \times W(a_i, O_{fq})$, otherwise.
4. Select the candidate with the highest utility value calculated in Step 3 for inquiry. If there is more than one such candidate, then randomly pick one.

The terms and the operation of this strategy are described as follows. Basically, Strategy 1 considers two factors, namely, the **ranking** of an agent and its **statistical** information on acceptable responses. Unacceptable answers include the *is_unknown* and control responses. These two factors are discussed below.

First, the scheme presumes that the amount of information encoded in a cooperating agent is proportional to its position in the organization hierarchy, that is, the higher an agent's position, the more it knows. In addition, a higher ranking agent generally accesses more information sources than a lower one. Thus, in Step 1 of the strategy, the agent derives a preference ordering O_{role} based on the role values of all possible candidates for its inquiry.

Second, the previous response messages from a particular agent recorded in the discourse table would provide insight on the likelihood of success in querying that agent. In Step 2, using the information recorded in the discourse table, the inquirer calculates the frequency of acceptable responses of an agent by taking the ratio of the previous inquiries to that agent with acceptable answers and the total number of inquiry messages, excluding pending ones, sent to that agent. It further normalizes these frequencies with their standard deviations to reflect the size of the samples, i.e., the number of previously responded inquiries, and forms a second ordering O_{fq} of all candidates using these normalized frequencies. Remember that, for a frequency f of a sample size N, the standard deviation of f is $((f \times (1-f))/N)^{1/2}$ [38].

In Step 3, we initially consider that both factors are roughly of equal importance and calculate the utility value of a possible candidate by adding its weights in O_{role} and O_{fq}. The weight is calculated as follows. If an ordering O of n candidates has agent b in the j^{th} position, then, the weight of b in that ordering is: $W(b,O) = n + 1 - j$.

The term $que_{a,a_i,t}$, shown in the same step, denotes the total number of query messages with acceptable answers that a has sent to a_i up to that point in time t. As more messages are exchanged between two agents, more is known about each other's response pattern on inquiries through the information recorded in the discourse table, and the importance of one agent's position in making a guess will be diminished. To reflect such a change, once $que_{a,a_i,t}$ is over a threshold value N_{th}, we ignore the role factor and assign a weight, WEIGHT, to the frequency ordering O_{fq}. This weight is a system parameter tunable for a particular application (it is set to 2 in BDN application).

An example may help to show how this strategy works. Let WEIGHT still be 2. Suppose agent a wants to know about a proposition p, and it identifies three possible candidates: b, c, and d. Let the values of the roles of b, c, and d be 2, 1, and 1, respectively. Hence, the role ordering O_{role} is: $b > c, d$, and their weights are 3, 2, and 2 (see the formula above for assigning weights). Next, agent a obtains the total number of inquiries responded to and the total number of acceptable replies from b as 10 and 6, c as 30 and 15, and d as 5 and 4. Based on these numbers, a calculates the normalized frequencies of b, c, and d as, 3.87, 5.48, and 4.47 (refer to the calculation of normalized frequencies in this section), and we have that O_{fq} is $c > d > b$.

Note that although d has the highest percentage of acceptable answers, 80%, due to its small sample size (5), the normalization result shows that d does not dominate c, which has a lower percentage (15/30 = 50%). Accordingly, the weights of b, c, and d, in O_{fq} are: 1, 3, and 2. Since the number of inquiries to any of these agents is less than 100 (N_{th}), the utility values assigned are: $U_1(b) = 4$, $U_1(c) = 5$, and $U_1(d) = 4$. Thus, a picks c for its inquiry.

In addition, for efficiency, a local handler in BDN continuously updates the number of inquiries and normalized frequencies of incoming messages. When Strategy 2 is invoked, the linear orderings can then be obtained directly from these updated values, with little searching and calculation. Remember that the handler and the reasoning program of a BDN agent are two concurrent processes. At the software level, this ongoing updating does not interfere with the operation of the reasoning program. This, however, does interfere to a certain extent at the operation system level, as both processes share the same computing resources of the UNIX workstation in which they reside. To compensate for this, the priority of such ongoing updating is set to be lower than that of the local reasoning program's jobs.

4.3 Which incoming message to execute

At any point of the operation, an agent might have several incoming messages in its handler waiting to be executed. The task of selecting which one to execute first is important from the perspective of computational efficiency.

Let the set of messages waiting in the knowledge handler of an agent a be buf_a. Let us denote the set of response messages in buf_a as M_R and the remaining action messages as M_A. Let us further denote the sender of message m as m_x and its time queuing in the handler as m_t. The following strategy is adopted to decide which incoming message to execute.

Strategy 2

1. If there exists one or more control messages, then execute any one of them arbitrarily and exit. If there are no response messages, then skip to Step 4. Otherwise, go to Step 2.
2. Calculate the utility value of every message m in M_R using the utility function:
   U_2 = role value of m_x + $(\mathrm{m}_t/\mathrm{TIME})$, where TIME is a system-defined normalization parameter.
3. Send the decoded information of the message with the highest utility value to the reasoning program, and then exit.
4. Repeat Steps 2 and 3 with M_A in replace of M_R.

Essentially, Strategy 2 treats the following factors in descending order of importance. First, the agent would select a response message before interpreting any action message. One can say that the agent prefers receiving information to giving information; moreover, the more it knows, the better it answers questions. Second, a response message for control purposes, such as notifying the circularity found in a discourse, has a higher priority than other direct responses. Third, the higher the rank of the sender, the higher the priority (or the greater the strength) of its message. Note that the information about the ranking is indicated by the message type. Last, the longer a message has been waiting in the handler, the sooner it will be executed. This follows the first-in-first-out approach of job scheduling that is used in most operating systems. Further, the importance ordering of the last two factors can be adjusted by modifying the normalization parameter TIME. This tunable parameter depends on the estimation of the allowed time in the buffer, the frequency of incoming messages, and the preemption rate of pending messages.

4.4 Which pending message to throw out

Ideally, every agent in a network should be able to entertain all messages sent to it. During heavy traffic communication, however, an agent would have so many messages in its handler that those with low utility values, as determined by Strategy 2, might never be executed. Such delays might have a negative impact on the performance of the overall system. It is better for an agent to let the other party know about its busy schedule so that the latter can take alternative actions, such as working on other unrelated problems temporarily, before sending the message again.

In COSMO, an agent imposes a limit on the number of pending action messages allowable from each of the other agents. Whenever the limit of a particular agent is exceeded, the local agent must decide which of the pending action messages, not the response messages, to throw out and must send a message of type *busy* to that particular agent about the rejection. Such a limit is set while considering two factors: (1) the estimated processing power of the local agent, and (2) the ranking difference between the two agents. Agents of higher ranks in an organizational hierarchy would have more privilege to access information than lower ones. Therefore, for three agents a, b, and c such that $v(role_a) > v(role_b)$ and $v(role_b) = v(role_c)$, the maximum number of pending action messages which b allows for a is more than what b allows for c.

For any agent a, let us denote its set of pending action messages from b as $wait_b$. The strategy that agent a uses to discard one of the messages in $wait_b$ is shown below.

Strategy 3

1. For every action message in $wait_b$, find its utility value according to function U_2 of Strategy 2.
2. Discard the one with the lowest value, and send a response message of type *busy* to notify b about the rejection.

One can see that the function of U_2 is twofold. An agent can use it to select a pending message for interpretation, or at the same time, to throw out an overflow message.

5 Communication Protocols

Communication protocols are sequences with precisely defined steps to convene more complex intentions than single communicative acts. They impose **orderly** interactions among cooperating agents to solve specific classes of problems. The purpose of this section is to present some of the key protocols implemented in COSMO for various purposes of cooperation. The first protocol is concerned with the allocation of tasks and is generic in nature, whereas the other protocols aim to resolve conflicts among agents and are tied to particular decision-making methods. We anticipate that when the protocols become more sophisticated, they will be more closely related to the domain knowledge of the agents.

For clarity of presentation, we assume that the underlying communication network is reliable and ignore the case in which some agent will reject a message, such as when it is too busy to participate. In COSMO, when any faulty process is detected, it would abort the protocol. Meanwhile, the handler of the sender of a rejected message will keep trying the same message a number of times before it notifies the sender or asks the initiator agent to abort the protocol. We leave it up to the reader that, for these protocols, the redundant and circular acts never occur and the delay of issuing messages (due to different internal computation times) will not cause malfunctions.

In addition, there are many ways to compose protocols from the set of message types listed in Figure 2 that will solve the same problems. But the efficieny of a protocol is measured by the amount of time required for its computation and communication steps, thus, it is possible to compare the efficiency of these alternate protocols for a set of automated agents. Analysis on the cost performance of protocols, however, is beyond the scope of this paper.

5.1 Contract Nets

A contract net [49] consists of a group of distributed agents that communicate to solve a problem by task sharing. Every agent has the same communicative competence, i.e., every agent is capable of accepting and assigning tasks. Every agent can take on all possible roles. The agents can take on the role of **manager** (one who announces contracts, evaluates and accepts bids, supervises task execution, and processes the results of execution) or the role of **contractor** (one who makes bids and executes tasks in contracts). These roles are taken up dynamically during the course of problem solving. Let the manager of a cooperative system be a and the set of contractors identified by the manager for a set of tasks be $a_1, ..., a_m$. Figure 6 shows the announcement and bidding phases of a single-stage contract net protocol of COSMO (the reward and result phases will be shown in the following figure).

To improve computational efficiency, Step 1 of the protocol considers a contract of

(1) $a \rightarrow a_1, ..., a_m$	*tell*, contract([p1,...,pn]), l
(2) a_i	decide bid(yes, list_i) such that $\text{list}_i \subseteq 2^{\text{p1,...,pn}}$; otherwise bid(no,nil)
(3) $a_i \rightarrow a$	*report*, bid(yes, list_i), l
(4) a	if no combination of bids can complete contract([p1,...,pn]) then abort; else select contractors
	... (rewards and result phase)

Figure 6: A single-stage contract net protocol of COSMO.

(1)	$a \rightarrow b, c, d$	*tell*, contract([p,q,r]), l
(2)	$b \rightarrow a$	*report*, bid(yes, p), l
	$c \rightarrow a$	*report*, bid(yes, [q,r]), l
	$d \rightarrow a$	*report*, bid(yes, [p,q]), l
(3)	$a \rightarrow b$	*tell*, p, l
	$a \rightarrow c$	*tell*, [q,r], l
(4)	$b \rightarrow a$	*report*, result_p, l
	$c \rightarrow a$	*report*, [result_q, result_r], l

Figure 7: Communication steps in the contract net protocol in Figure 6.

multiple independent tasks, p1,...,pn, instead of one task as in conventional contract net protocols. (In the syntax of the global language of COSMO, [...] represents a list of terms.) This simultaneous announcement of several tasks may not always be possible. For example, the tasks p1,...,pn are interrelated such that the result of any one of them will affect the solution of others. In Steps 2 and 3, every contractor evaluates a list of tasks that it wants to bid for and sends the list to the manager. Note that 2^x denotes a power set of x. In Step 4, the manager attempts to identify a combination of agents that can complete the contract. If none is found, the manager aborts the operation.

Figure 7 shows an example of the communication steps involved in this protocol. This example assumes one manager and three contractors bidding for three tasks: p, q, and r. In Step 4, the term result_x denotes the computed results of task x. The manager awards the contract to *b* and *d*, as these two together complete the tasks indicated in the contract.

COSMO also adapts a more general form of contract net protocol: multi-stage negotiation protocol [10, 11]. In short, this protocol involves the decomposition of a global goal into subgoals (this presumes that the agents have the planning ability to do so) and iterative announcements and bids on subgoals to acquire sufficient knowledge to solve the mutual goal. This protocol is not shown here since the implementation of contract nets in COSMO is illustrated adequately in Figures 6 and 7.

5.2 Preference-Based Negotiation

This subsection describes a set of protocols that support the negotiation of conflicts. There are two sorts of conflict. Some are conflicts among agents and some are conflicts that an agent has with itself. Let us call these **social** and **personal** conflicts.

What, then, do we mean by "conflict"? An agent (knowledge-based system) conflicts with itself when it asserts two propositions that it knows can't both be true at the same time. Such an agent asserts both α and β and believes that $\alpha \rightarrow \neg\beta$ and thus $\beta \rightarrow \neg\alpha$. This personal conflict is caused by inconsistency in its knowledge. In a social conflict, this is parcelled out among several agents. In this case, some agents want α and others want β. Each of them thinks that they cannot have both. Research in the past has been focused on the problems of personal conflict, i.e., the consistency of a single computer program. The recent attempt to use cooperative problem solving systems for broader, more complex tasks has brought home to researchers the difficult issues of social conflict.

Where now is the problem of conflict? A conflict usually causes a standstill, that is, it **inhibits** further action during the computation process. Not every conflict is like this however. It can happen that agents in a conflict don't even know they are in one, or don't care enough to make an effort, or are too weak to block the others.

The protocols presented in this paper are mainly concerned with the type of social conflicts which paralyze action, since its impact on the entire cooperative system is immediate and detrimental. Something or other must be done at once, and agents stand in each other's way.

These protocols are used in conjunction with the group decision making methods in **social choice theory** [46, 4], a rich research field in decision science. In essence, these methods generally include an agenda which contains a list of criteria for each mutual problem. The agents would first form individual orderings of preferences on competing alternatives of a problem according to the specified criteria (which may vary for each agent). They, then, apply some aggregation procedure to select an outcome out of these individual preferences. Such an outcome is often known as a **collective choice.**

Nevertheless, it is difficult to obtain a "fair" choice that satisfies all agents. Thus, in COSMO, we allow negotiation among agents to iron out the differences and uncertainties of individual preferences. In this regard, we depart from the social choice theory, which normally does not include the notion of **feedback** in the group decision making process.

The process of gathering and forming preference orderings in our scheme is briefly

O_a	O_b	O_c	O_g
z	x	z	z
x	y	y	x
y	z	x	y

Table 1: Individual ordering and aggregated ordering.

described as follows. An agent of the **coordinator** role would gather all individual orderings of preferences (using the contract net protocol in Figure 6) and combine them into one ordering according to a certain aggregation procedure. If any agent disagrees with the result, it can then protest to the coordinator in order to start a session of negotiation. This strategy allows only one coordinator in a cooperative system.

In addition, the formation of individual preferences for cooperative problem solving is neither fixed nor arbitary. In the context of knowledge base applications, individual preferences are normally derived from domain knowledge encoded in individual agents. If agents interact with the users, then the derivation of preferences may include the additional information provided by the users. To express the individual preferences of a problem, w, for different criteria, $i_1, ..., i_n$, an agent a requires justification from different parts of its domain knowledge, $KB_{a,i_1}, ..., KB_{a,i_n}$. Likewise, another agent b would derive its preferences on w from $KB_{b,i_1}, ..., KB_{b,i_m}$. Note that $m \leq n$, that is, cooperating agents may not share the same set of criteria for their mutual problem. This is expected as the agents often possess different kinds of expert knowledge and problem priorities. The agents may also update or revise their knowledge when they accumulate more information in their interaction. This in turn may dynamically affect the ordering of individual preferences.

A scheme based on the formalism of **preferential logic** that supports the derivation and aggregation of individual preferences for knowledge based applications has been presented in [57, 54]. The discussion of this scheme is beyond the scope of this paper, however, a simple example on aggregation is given below.

Let us suppose that there are three cooperating agents a, b, and c, where a is the coordinator and b and c are peers such that v(coordinator) > v(peer). Let us further suppose that each agent has its own criterion to judge its preferences, and the coordinator uses a standard procedure of aggregation: the simple majority rule. This rule specifies that, for a criterion, if both a and c prefer z to x while b laone prefers x to z, then the aggregated preference is that z is preferred to x, or z > x, with respect to that criterion. Table 1 shows the set of individual orderings O_a, O_b, and O_c, and the aggregated ordering of preferences O_g of a problem with competing alternatives x, y, and z. In this table, z is a collective choice as it is the highest ranking alternative in O_g.

If there is a dispute, then the following strategy of negotiation is used.

Strategy 4

1. Each of the agents computes a heuristic index by calculating the ranking difference of every feasible alternative between its individual ordering and the aggregated ordering.
2. Each of the agents checks if its index is over a threshold value, and, if so, asks the user whether it should complain or not; otherwise, it exits.
3. If any of the agents complains, start **bargaining**, and if bargaining fails, try **forcing**.

Strategy 4 includes the users of individual agents in the decision making process. A variant of this strategy would be to have one or more agents to not ask local users. In Step 1, every agent first obtains a heuristic index of bargaining. For example, in Table 1, agent b calculates x's ranking difference between its ordering O_b and the aggregated ordering O_g as 1 and the total difference, that is, its heuristic index H_b, as 4. Similarly, we have $H_a = 0$ and $H_c = 4$. An agent uses its heuristic index to decide whether to flag the users for conflicts and to estimate whether the negotiation is converging towards a satisfactory solution.

In Step 2, suppose that a uniform threshold, H_{th} = number of alternatives $- 1 = 2$, is applied across all agents. Thus, b and c both would query their users. Let us further suppose that the user of c decides not to complain, since z is in a reasonably good position in O_c. The user of b, however, would like to complain about the outcome as z is its least preferred choice.

The predominant mode of negotiating over a conflict in this strategy follows Galbraith's notion of **bargaining**, i.e., the agents push for acceptance of the alternative which is preferred by them and occasionally "give in" by making incremental changes to their preferred alternatives [21, 22]. This treatment differs from the prevalent approach in distributed AI that presumes the knowledge of conflict resolution can be encoded and centralized in a special negotiator agent [32, 50, 51, 42].

COSMO uses many different types of bargaining protocols. For brevity, this section illustrates three of them and focuses only on the discussion of agents' preferences. A more extensive interchange of domain knowledge and partial conclusions is frequently involved. In BDN application, for example, the agents can query the background knowledge of one another that deduces conflicting preferences (see Chapter 8 of [54]). Note that, in this prototype, the agents' knowledge is represented with a mix of representation schemes such as object-orientation, semantic nets, and production rules [56, 54] but that are mapped into the same global format for communication.

5.3 Negotiation Protocols

In Figure 8, we show a protocol of bargaining, bp_I. The protocol considers a set of agents $a, b, a_1, ..., a_n$ of the following role assignments: a is the coordinator and $b, a_1, ..., a_n$ are peers.

(1)	$b \rightarrow a$	*protest*, protocol(bp_I), l
(2)	a	if approve then start bp_I; else tell b to abort
(3)	$a \rightarrow b$	*approve*, protocol(bp_I), l (assume approved)
(4)	b	determine to swap p (may exchange
		inform all agents about p (background knowledge)
(5)	$b \rightarrow a$	*recommend*, p, l
	$b \rightarrow a_1, ..., a_n$	*offer*, [p, protocol(bp_I)], l
(6)	$a, a_1, ..., a_n$	if p is feasible then accept; else reject
(7)	$a \rightarrow b$	*endorse/reject*, nil, l
	$a_i \rightarrow b$	*accept/refuse*, nil, l
(8)	b	if all agree then swap; else abort

Figure 8: The bargaining protocol bp_I of COSMO.

Basically, any agent which wants to complain must first seek the approval of the coordinator. If approved, the agent would then attempt to persuade all other agents to swap two alternatives' positions in their individual orderings (computed in Step 4 as p), such that the expected aggregation results would favor its best choice. We skip the interchange of background knowledge among agents here. In Step 5, the protesting agent b also informs other agents about the protocol used so that the latter can select appropriate operations for this protocol. The protocol bp_I is successfully terminated only when all agents agree to the change. The reader may want to refer to Figure 2 for the message types used in Figure 8.

One should also note that, sometimes, the content of a response message is ignored when its type carries sufficient information. Step 7 of Figure 8 shows one such occasion, where the feasibility of p is indicated by the response types.

Let us continue the previous example in Table 1. We show a sequence of steps using the bargaining protocol bp_I in Figure 9. In Step 5, feasible(swap(a,x,z)) denotes whether it is feasible to swap the positions of x and z in a's ordering. One of the conditions of feasibility is that the new index after the change should not be over the threshold value, i.e., 2. Other conditions would involve checking the background knowledge which derives the preferences. This part concerns the underlying problem-solving methods of the reasoning programs, which will be discussed in a separate paper. Table 2 shows the new set of orderings after the completion of the bargaining in Figure 9. The set of heuristic indices are: $H_a = 2$, $H_b = 0$, and $H_c = 0$, where H_{th} is still 2 (see Section 5.2).

Often it is difficult for all agents agree to a change. Consider the previous example, for c

(1)	$b \rightarrow a$	*protest*, protocol(bp_I), l
(2)	a	start bp_I
(3)	$a \rightarrow b$	*approve*, protocol(bp_I), l
(4)	b	determine swap x, z
		inform all agents
(5)	$b \rightarrow a$	*recommend*, feasible(swap(a, x, z)), l
	$b \rightarrow c$	*offer*, [feasible(swap(c, x, z)), protocol(bp_I)], l
(6)	a	test that swap(a, x, z) is feasible
	c	swap(c, x, z) is feasible
(7)	$a \rightarrow b$	*endorse*, nil, l
	$c \rightarrow b$	*accept*, nil, l
(8)	b	invoke the change
(9)	$b \rightarrow c$	*offer*, swap(a, x, z), l
	$b \rightarrow a$	*recommend*, swap(c, x, z), l
(10)	$a \rightarrow b$	*endorse*, updated, l
	$c \rightarrow b$	*accept*, updated, l

Figure 9: A sequence of bargaining using Protocol bp_I.

O_a	O_b	O_c	O_g
x	x	x	x
z	y	y	y
y	z	z	z

Table 2: Preference orderings after the bargaining process in Figure 9.

to change its best preference to the worst would require considerable revision of its domain knowledge, but b is not in a position to do so. Thus, the strategy of another bargaining protocol, bp_II, is to have an agent attempt to persuade, not all, but as many agents as possible to its preferred alternative, and hopefully, the new ordering of aggregation derived will be close to its expectation. One problem of this approach to bargaining is that an agent which refuses to change might stand to lose and, thus, would initiate another complaint. To avoid such situations, the protesting agent should also consider potential conflicts arising from its change. This is accomplished by the agent simulating the aggregation and detecting possible problems internally. For example, let us consider the case in Table 1 again. In Figure 10, agent b is still the protester which initiates bp_II, but this time c refuses to accept b's offer.

(1-5) is the same as (1-5) of Figure 9, except bp_II replaces bp_I

(6)	a	*test* that swap(a, x, z) is feasible
	c	swap(c, x, z) is infeasible
(7)	$a \rightarrow b$	*endorse*, nil, l
	$c \rightarrow b$	*refuse*, nil, l
(8)	b	not all agents agree, examine the would-be ordering (see Table 3) if okay then invoke change; else inform a, c to abort.
(9)	$b \rightarrow a$	*recommend*, abort(bp_II), l
	$b \rightarrow c$	*offer*, abort(bp_II), l
(10)	$a \rightarrow b$	*endorse*, nil, l
	$c \rightarrow b$	*accept*, nil, l

Figure 10: A sequence of bargaining using Protocol bp_II

O_a	O_b	O_c	O_g
x	x	z	x
z	y	y	z
y	z	x	y

Table 3: Preference orderings after the bargaining process in Figure 10.

Table 3, apparently, shows why b decides not to press the change in Figure 10. The expected O_g: x > y > z would satisfy b, but would likely upset c as the latter would have an index of 4 and, thus, exceed the threshold value (H_{th}) of 2 (there is no change to the indices of a and b). Further, the 'best' collective choice x would be the worst choice from c's perspective. We consider that this kind of "good faith" negotiation, where the concerns of other agents are taken into account in negotiation, is essential to cooperative behavior. On the other hand, one can say that agent b plans to avoid possible futility, as c definitely would object to this change and would start another round of bargaining to attempt to undo the change.

The two types of bargaining discussed so far are within a **fixed scope** of negotiation, that is, the agents are trying to reach an agreement under the same set of criteria in the problem agenda. Sometimes, the conflict may persist and no amount of such bargaining will provide for agreement. In COSMO, an agent may propose to use a third type of bargaining that proceeds to change the scope of negotiation. Instead of figuring out acceptable solutions

(1-3) is the same as (1-3) of Figure 9, except bp_III replaces bp_I

(4)	$b \rightarrow a$	*recommend*, [feasible(add_criterion(new)), reason(δ_{new})], l
	$b \rightarrow c$	*offer*, [feasible(add_criterion(new)), reason(δ_{new})], l

(5)	a, c	if reason(δ_{new}) is okay
		then accept add_criterion(new) is feasible;
		else reject.

(6)	$a \rightarrow b$	*endorse*, nil, l (assume both accept)
	$c \rightarrow b$	*accept*, nil, l
(7)	$b \rightarrow a$	*recommend*, [success(bp_III), add_criterion(new)], l
(8)	$a \rightarrow b$	*endorse*, nil, l
(9)	a	invoke contract net protocol to form new preference orderings.

Figure 11: A successful bargaining sequence of Protocol bp_III

Old			New			Aggregate
O_a	O_b	O_c	O_a^{new}	O_b^{new}	O_c^{new}	O_g
z	x	z	x	x	y	x
x	y	y	z	z	x	z
y	z	x	y	y	z	y

Table 4: Preference orderings after the bargaining process using bp_III.

under the existing problem agenda, this type of bargaining attempts to **reshape** that agenda, or rather, tries to replace the set of criteria with another different but closely related set. Changing the agenda is changing the subject of the conflict. Under the new criteria that the persuader brings up, it expects that the other agents (or their users) would likely agree with its proposal. By getting the opposing agents to accept this new agenda, the persuader hopes to get them to join its side. In other words, we can say that this kind of bargaining is an attempt by one agent to change the opposing agents understanding of something, to get them to see it in some way (i.e. change their viewpoints) that prompts them to act as they would not have done otherwise.

The following example is used to illustrate such bargaining in COSMO. Consider agent b and Table 1 again. This time, b tries to persuade a and c to include a new criterion in the problem agenda. Refer to Figure 11, in Step 4, the term, new, is the additional criterion that b proposed while reason(δ_{new}) is the justification of b to add this criterion. If any other agent is not satisfied with this justification, bargaining would be aborted here.

(1)	$b \rightarrow a$	*protest*, protocol(bp_II), l_1
	...	
(n)	c	reject b's offer by n_{max} times
(n+1)	$c \rightarrow b$	*reject*, nil, l_1
(n+2)	$c \rightarrow a$	*protest*, [break_off(bp_II), force(pr(g, z, x))], l_1
(n+3)	$a \rightarrow c$	*approve*, nil, l_1
(n+4)	$a \rightarrow b$	*tell*, abort(bp_II), l_1
(n+5)	$b \rightarrow a$	*report*, "okay", l_1
(n+6)	$a \rightarrow b$	*tell*, [pr(g, z, x), forcing], l_2
(n+7)	$b \rightarrow a$	*report*, "okay", l_2

Figure 12: Forcing after a failure of bargaining.

This protocol succeeds when the coordinator agent, a, is ready to invoke the group decision making method to form a new set of preference orderings, using the contract net protocol in Figure 6. Thus, the kind of bargaining used in bp_III is a **pre-decision** strategy.

Table 4 gives the results of new orderings. In this table, both old and new criteria (and thus the orderings) are taken to have equal importance in aggregation. The aggregated result shows that b gets what it wants out of this bargaining sequence, that is, to push for x as the collective choice. Note, a new way to calculate heuristic indices and to define the threshold is required, but we do not deal with this here.

The bargaining protocols enable us to incorporate deeper cooperative behavior into autonomous agents. They, however, may not be applicable in some occasions. Agents, for example, may resist the persuasion to change their preferences or to consider a new criterion. Occasionally, **forcing** is used to back up the bargaining approach when a lack of agreement stymies the group. The coordinator can use the authority of its position to force a preferred alternative on the rest of the group. The protocol of forcing is invoked when the coordinator approves the request of an agent to break off the bargaining mode due to a lack of progress. In our applications, such a request to break off is triggered when the maximum number of offers n_{max} allocated in the bargaining session is exceeded.

Figure 12 shows an example of forcing after a failure of bargaining protocol bp_II, using the example in Table 1. In Step n, c finds that b exceeds n_{max}. Then, in the following steps, c protests to the coordinator to change the negotiation mode to forcing in order to have b accept that the aggregated choice z is better than x (denoted as pr(g, z, x)). A change of mode means starting a new sequence of steps. This change is indicated by a new message label l_2 in Step (n+6).

A word of caution on the use of forcing follows. This approach settles the problem of action so that the agents can be on the move, that is, the computation process can continue.

It, however, leaves the conflict unresolved. The agents are still at odds with their opposing beliefs about the matter as well as remaining in conflict. The use of forcing without proper direction from the domain knowledge would simply generate arbitrary solutions.

6 Related Work

Intelligence interfaces between information processing systems is an important research area that has been gaining much attention recently. This body of related work, in fact, covers so much that this section mentions only the most closely related work, and briefly at that. The references listed, however, will help the reader to probe further into this fast growing area. The following is presented in what we see as decreasing relevance to COSMO.

- Mike Huhns and his coworkers propose the use of communication aides to support the interaction between expert systems. These aides are implemented in the RAD distributed expert system shell at MCC [29, 2]. Similar to knowledge handlers in COSMO, a communication aide can be attached to an existing expert system and enables the expert system to interact with the aides of other expert systems through message passing. The common language used by aides is SURF, which has been proposed to as a superset of the knowledge representation languages of existing expert system shells [25].

  Compared to the design of COSMO, this work focuses on the low-level communication and reasoning primitives necessary for beneficial agent interactions.[1] However, the semantics of message types and the type hierarchy (speech acts) are not defined in RAD, and the aides ignore the importance of background knowledge, such as the history of acts and the priority of messages, in the discourse. Another distinction is that the interaction between RAD agents still adheres to the **master-slave assumption**, i.e., one agent has desires and issues a command or request and the other agent takes that command and executes it faithfully. Ongoing work includes the developing of advanced protocols that go beyond contract nets for distributed problem solving.

- There is other related work within the Distributed AI community [16, 15, 17, 23], and, to a lesser extent, the Distributed Database community [39, 59]. COSMO, however, is, to my knowledge, unique in its precise definition of communicative acts and the resulting set of communication strategies and protocols. Furthermore, many of the reported works are confined to conception or toy problems.

[1]There are also architectural differences between COSMO's handlers and RAD's aides at this level. For example, a communication aide uses UNIX pipes for message passing while COSMO adopts TCP. Aides are expected to use a nameserver in the local network to keep agents' addresses, but there is no such central nameserver in COSMO.

- Yoav Shoham's work on Agent Oriented Programming (AOP) [47]. Shoham's interest lies primarily in forming a new programming framework, in which an agent contains the mental state of various components, that is, beliefs, decisions, capabilities, and obligation, and interacts with other agents through speech act primitives. The model of communication does not include protocols and organizational hierarchy.

  A restricted form of the AOP framework is implemented in a programming language called AGENT-0. Communication actions in AGENT-0 are limited to three types: *inform*, *request*, and *unrequest*. COSMO covers the functions of these primitive types. In addition, it takes account of the agents' priorities and uses the primitive types as building blocks to form more complex protocols.

  In contrasting with COSMO, agents in AGENT-0 are serialized program modules that are located in the same machine. The information is thus rather centralized. The designer has a global view and control of the problem solving abilities of these agents. In COSMO, agents are physically as well as logically distributed and are often preexisting systems for particular applications. A global view and control of the information flow cannot be presumed in such cooperative systems.

- The work of Candice Sidner and Barbara Grosz in the SHAREDPLAN model [33, 7, 26] that extends earlier work on single-agent plan representation and recognition in AI and linguistics (e.g., [40, 31, 1, 48]) for multi-agent planning and discourse. In this model, "shared plans" are plans that a group of agents may have in order to complete a particular action or goal. The agents are required to have a mutual belief that they have a given shared plan, and a mutual belief that each agent intends to contribute to it. The SHAREDPLAN model also employs communication acts to convey explicitly the intentions and uses the relationship between these acts to build more complex acts. This idea is similar to the design of COSMO's protocols.

  Unlike COSMO, however, communication acts in SHAREDPLAN are unconstrained. Arbitrary predicate relations can be used to denote such acts in the agents' plan representation structure, the latter sometimes causes notational problems. Further, communication acts ignore the importance of background knowledge in discourse interpretation [28]. This work focuses on the level of conception analysis. Implementation issues of distributed computing, such as communication methods and query management, are not addressed.

- Tim Finin and Gio Wiederhold are leading the efforts of the external interfaces group in the Knowledge Sharing Initiative to define protocols and conventions that will enable interoperability between knowledge bases through queries and responses [36, 9]. Their goal is to create a query language for the interface between knowledge bases and relational or object-oriented databases [19, 35].

In comparison, the interface issues addressed in COSMO are at a level higher than data transactions. This work aims to design communication facilities, based in part on the speech act theory, for cooperative problem solving among distributed knowledge based systems. The current design of COSMO is inefficient in processing large amounts of data and does not commit to particular data structures, except those encountered in its applications.

- Gio Wiederhold has recently proposed mediators as a crucial part of the architecture of future distributed information systems, in particular, multiple heterogeneous databases [52, 53]. Similar to knowledge handlers, mediators are independent units and have to deal with information from multiple sources. They are explicit active modules between the user's application and the data resources. They do not act independently, but respond to queries from applications or to triggers placed in the databases.

  Besides communication and transformation knowledge, the mediators are also expected to embody task-dependent expert knowledge to create and combine information needed for end user applications. In COSMO, this kind of domain knowledge is encapsulated in the reasoning program of the agent that drives the handler. For large-scale and widely distributed knowledge bases, however, it may be necessary to encode such knowledge into the handlers to achieve better maintenance and modularity of knowledge. By taking this direction, a handler becomes a special form of mediator.

7 Conclusion

Cooperative problem solving is an important paradigm for the next generation of information processing systems. Cooperative work will be conducted in many forms among a network of agents and will require the support of advanced communication facilities beyond the "passive" transmission of data and messages provided by the current network technology.

In this paper, we have presented such a scheme, COSMO, for supporting orderly and cooperative interaction among distributed systems. This scheme has been implemented in a cooperative knowledge-based system, BDN, to solve preliminary building design problems in a project team. It is a crucial component of a framework for cooperative problem solving in construction developed in the NSF-ATLSS center at Lehigh [54]. The preliminary concepts of this scheme are: (1) a **coding model** of communication to facilitate an interface between agents with different internal structures; and (2) the notion of **communicative acts** to perform functions indicated by their types, such as *promise, order*, and *offer*. We discussed the key design features of COSMO and provided our answers to the following set of practical questions on cooperative systems communication:

(Q1) When and how should a cooperating agent know which act to perform?
(Q2) What are the conditions of success of a communicative act?

(Q3) What is the relationship between organizational roles and communicative acts?
(Q4) How does an act relate to other acts?
(Q5) Can more complex intentions be convened by a few communicative acts, and if so, how?

A summary of our results is as follows:

- Use an **operational model** to specify when an agent should initiate a communicative act (Q1) and what the conditions of success of such acts are (Q2).
- Use **organizational roles** to select the appropriate strategies of problem-solving and communication, such as the assignment of message types in various acts (Q3).
- Use **message types** to specify particular points or purposes of communicative acts, the strength of acts, and types of response acts (Q4).
- Devise **utility functions** to select the most preferable acts from many feasible alternatives for general communication control purposes (Q1).
- Develop **protocols** to precisely define the sequences of communication and computation steps for cooperation purposes, such as negotiation of conflicts and task allocation (Q5). These protocols should be defined in terms of a few primitive acts and might closely relate to the underlying problem-solving methods.
- Use **knowledge handlers** to achieve all the aforementioned design concepts. The handlers can be attached to the existing knowledge-based systems and thus integrate these systems into a cooperative framework, but without jeopardizing local autonomy.

The design of COSMO is motivated by the lack of a proper communication mechanism for supporting CPS applications, and, in particular, applications in the construction domain [3, 54]. System developers should not be handicapped by the primitive concepts and constructs of existing communication schemes. Higher level concepts and constructs are needed so that developers are free to write the highly specialized parts that provide efficiency and are unique to a given application. This would help developers to focus on even-harder problems, pushing forward the state of the art and providing increasing value to the application users. In presenting this account, we attempt to offer new insight into the use of communicative acts and protocols to form highly advanced communication schemes for CPS systems, and in particular, cooperative knowledge-based systems.

Future work will use the present COSMO as a testbed to determine what existing constructs can be extended as well as what new concepts are needed to develop new protocols and strategies. This will involve performance analysis on alternate protocols, that is, analyzing the amount of communication and computation time required to solve a given problem for the agents in these protocols and selecting the one with the least time. Another method to enhance the expressive power of agents is to enrich the communicative acts with complex illocutionary acts, i.e., conditional and conjunctional acts [45], and more message types for new functions. The long term goal is to enhance the generality and portability of knowledge handlers for various cooperative knowledge based systems. This may require the study of new ways to encode more administrative and domain knowledge in the handlers' knowledge bases, that is, a move towards the mediator architecture advocated in [52].

References

[1] Allen, J. F., Perrault, C. R. "Analyzing intention in utterances," AI Journal, vol. 15, pp.143-178.

[2] Arni, N. V., et. al., "Overview of RAD: A hybrid and distributed reasoning tool," MCC Technical Report No. ACT-RA-098-90, MCC, Austin, TX, March 1990.

[3] ATLSS Center. *Sixth-year renewal proposal to the NSF, Vol. 2: projects, publications, and biosketches*, ATLSS Drive, Lehigh University, Bethlehem, PA, 1992.

[4] Arrow, K. J. *Social choice and individual values*, Wiley, 1951.

[5] Austin, J. L. *How to do things with words*, Oxford: Clarendon Press, 1956.

[6] Barr, A., Cohen, P. R., Feigenbaum E. A. (eds.), *The handbook of artificial intelligence*, Vol. 4, Morgan Kauffman, 1989.

[7] Balkanski, C. T., Grosz, B., Sidner, C. L. "Act-type relations in collaborative activity," Technical Report, Harvard University, February 1990.

[8] Brodie, M. L. "Future intelligent information systems: artificial intelligence and database techniques working together," in Mylopolous J., Brodie, M. L. (eds.), *Readings in Artificial Intelligence and Databases*, Morgan Kaufmann, 1989, pp. 623-641.

[9] Brown, S., "Research in knowledge sharing," in *The Spang Robinson Report on Intelligent Systems*, Vol. 8, No. 1, December 1991 - January 1992, John Wiley and Sons, 1991, pp 19-24.

[10] Conry, S. E., Meyer, R. A., Lesser, V. R., "Multistage negotiation in distributed planning," in A. Bond and L. Gasser (eds.) *Readings in distributed artificial intelligence*, Morgan Kaufmann, 1988, pp. 367-384.

[11] Conry, S. E., Meyer, R. A., Lesser, V. R., "Multistage negotiation for distributed satisfaction," IEEE Trans. on Systems, Man, and Cybernetics, Vol. 21, No. 6, 1991, November/December, pp. 1462-1477.

[12] Bowers, J. M., Benford, S. D. (eds.), *Studies in computer supported cooperative work – theory, practice and design*, North-Holland, 1991.

[13] CSCW 88: Proceedings of the Conference on Computer Supported Cooperative Work, Oregon, ACM, September, 1988.

[14] Daisy Working Group, "Distributed aspects of information systems," Proceedings of the European Teleinformatics Conference 1988, pp. 1029-1049.

[15] Deen, S. M. (ed.) *Cooperative knowledge based systems 1990*, Springer-Verlag, 1991.

[16] Proceedings of the 10th International Workshop on Distributed Artificial Intelligence, Bandera, TX, October, 1990.

[17] Durfee, E. H., Lesser, V. R., Corkill, D. D. "Cooperative distributed problem solving", in Barr, A., Cohen, P. R., Feigenbaum E. A. (eds.), *The handbook of artificial intelligence*, vol. 4, Morgan Kauffman, 1989, pp. 83-148.

[18] Durfee, E. H. (ed.) Special section on distributed artificial intelligence, IEEE Trans. Systems, Man, and Cybernetics, Vol. 21, No. 6, November/December, 1991.

[19] Finin, T., Fritsson, R., McKay, D., McEntire, R., O'Hare, T., "The intelligence system server – delivering AI to complex systems," Proceedings of the IEEE International Workshop on Tools for AI – architecture, languages, and algorithms, March 1990.

[20] Fikes, R., Nilsson, N. J. "STRIPS: A new approach to the application of theorem proving to problem solving," AI Journal, vol. 2, pp. 189-208.

[21] Galbraith, J. *Organizational Design*, Reading, Mass., Addison-Wesley, 1973.

[22] Galbraith, J. *Designing complex organizations*, Reading, Mass., Addison-Wesley, 1977.

[23] Gasser, L., Hill, R. W. "Coordinated problem solvers", in *Annual Reviews of Computer Science*, vol. 4, (1989-90), pp. 203-254.

[24] Gasser, L. "A dynamic organizational architecture for adaptive problem solving," Proceedings of AAAI-91, July, 1991, pp. 185-190.

[25] Genesereth, M. R., Gruber, T., Guha, R. V., Letsinger, R., Singh, N. P. "Knowledge Interchange Format," Logic Group Report, No. Logic-89-13, Computer Science Department, Stanford University, Stanford, CA, January, 1990.

[26] Grosz, B. J., Sidner, C. L., "Plans for discourse," in Cohen, P., Morgan, J., and Pollack, M. (eds.) *Intentions in communication*, MIT Press, 1990, pp. 417-444.

[27] Gupta, A. *Parallelism in production systems*, Morgan Kaufmann, California, 1988.

[28] Hobbs, J. R., "Artificial intelligence and collective intentionality," in Cohen, P., Morgan, J., and Pollack, M. (eds.) *Intentions in communication*, MIT Press, 1990, pp.445-460.

[29] Huhns, M. N., Bridgeland, D. M., Arni, N. V., "A DAI communication Aide," in Proceedings of 10th International Workshop on Distributed Artificial Intelligence, Chapter 29, 1990.

[30] Hurson, A. R., Bright, M. W., "Multidatabase Systems: An advanced concept in handling distributed data," in Yovits, M. (ed.) *Advances in Computers*, Vol. 32, 1991. pp. 150-195.

[31] Kautz, H. A., Allen, J. F. "Generalized plan recognition," Proceedings of AAAI-86, PA, 1986.

[32] Klein, M., "Supporting conflict resolution in cooperative design systems," IEEE Trans. Systems, Man, and Cybernetics, November/December, Vol. 21, No. 6, 1991, pp. 1379-1390

[33] Lochbaum, K. E., Sidner, C., Grosz, B. J., "Models of plans to support communication: An initial report," Proceedings of AAAI-90, pp. 485-490.

[34] Malone, T. W. "Organizing information processing systems: Parallel between human organizations and computer systems," in Robertson, S. P., Zachary, W. W., Black J. B., (eds.), *Introduction to cognition, computation, and cooperation*, Ablex, 1990, pp. 56-83.

[35] McKay, D. P., Finin, T. W., O'hare, A., "The intelligent interface: integrating AI and database Systems," Proceedings of AAAI-90, July, 1990, p.667-684.

[36] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., Swartout, W., "Enabling technology for knowledge sharing," AI Magazine, Fall, 1991, pp.36-56.

[37] Ozsu, M., Valduriez, P. "Distributed database systems: Where are we now?" IEEE Computer, August, 1991, pp. 68-78.

[38] Papoulis, A. *Probability, random variables, and stochastic processes,* McGraw Hill, 1965.

[39] Papazoglou, M. P., Laufmann S. C., Sellis, T. K. "An organizational framework for cooperating intelligent information systems," the International Journal of Intelligent & Cooperative Information Systems, Vol. 1, No. 1, March, 1992, p.169 - 202.

[40] Pollack, M. E. "A model for plan inference that distinguishes between the beliefs of actors and observers," In Proceedings of the 24th Annual Meeting, Association for Computational Linguistics, New York, NY, June, 1986.

[41] Robertson, S. P., Zachary, W. W., Black J. B., (eds.), *Introduction to cognition, computation, and cooperation*, Ablex, 1990.

[42] Robinson, W. "A decision-theoretic perspective of multiagent requirements negotiation,", in the Proc. of AAAI-91 Workshop on Cooperation Among Heterogeneous Intelligent Agents, Anaheim, CA, July, 1991.

[43] Sacerdoti, E. D., *A structure for plans and behavior*, New York, American Elsevier, 1977.

[44] Searle, J. R. *Speech acts*, Cambridge University Press, 1969.

[45] Searle, J. R., Vanderveken, D. *Foundation of illocutionary logic*, Cambridge University Press, 1985.

[46] Sen, A. *Choice, welfare, and measurement*, MIT Press, 1982.

[47] Shoham, Y., "Agent oriented programming," AI Journal, to appear. (See also an earlier version in Technical Report CS-90-1335, Computer Science Department, Stanford University, CA, 1990.)

[48] Sidner, C. "Plan parsing for intended response recognition in discourse," Computational Intelligence, 1, Feb., 1985, pp.1-10.

[49] Smith, R. G. "The Contract net protocol: high-level communication and control in a distributed problem solver," IEEE Transaction on Computers, Vol. C-29, No. 12, 1980, pp. 1104-1113.

[50] Sycara, K., "Resolving goal conflicts via negotiation," Proceedings of AAAI-88, Augest, 1988, pp. 245-250.

[51] Werkman, K., "Knowledge-based model of negotiation using shareable perspectives," Proceedings of DAI-10, October, 1990, Chapter 6.

[52] Wiederhold, G., "Mediators in the architecture of future information systems," IEEE Computer, March 1992, pp.38-49.

[53] Wiederhold, G., et al., "Partitioning and combing knowledge," Information Systems, Vol. 15, No. 1, 1990, pp. 61-72.

[54] Wong, S. T. C. *A Framework for description and design of cooperative knowledge-based systems*, Ph.D. thesis, CSEE Department, Lehigh University, October, 1991.

[55] Wong, S. T. C., Wilson, J. L., "A set of design guidelines for object-oriented deductive systems," IEEE Transactions on Knowledge and Data Engineering, to appear.

[56] Wong, S. T. C. "A network representation for logic programs," Workshop on Knowledge Representation, the International Conference on Fifth Generation Computer Systems, June, Tokyo, Japan, 1992.

[57] Wong, S. T. C. "Cooperative decision making based on preferences," journal submission, 1992.

[58] Woo, T. Y. C., Lam, S. S., "Authentication for distributed systems," IEEE Computer, January, 1992, pp. 39-52.

[59] Woo, C. C., Lochovsky, F. H. "Knowledge communication in intelligent information systems," the International Journal of Intelligent & Cooperative Information Systems, Vol. 1, No. 1, March 1992, pp. 203-228.

[60] Uchida, S. "Summary of the parallel inference machines and its basic software," Proceedings of The International Conf. on Fifth Generation Computer Systems, June, Tokyo, Japan, Vol. 1, 1992, pp.33-49.