

**A NEGOTIATION PLATFORM FOR COOPERATING
MULTI-AGENT SYSTEMS**

**A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING AND
INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY**

by

Faruk Polat

December 1993

A NEGOTIATION PLATFORM FOR COOPERATING
MULTI-AGENT SYSTEMS

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING AND
INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

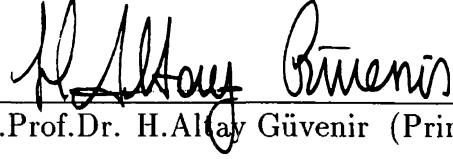
by
Faruk Polat
1993

Faruk polat
.....
tarafından bağışlanmıştır

B 023216

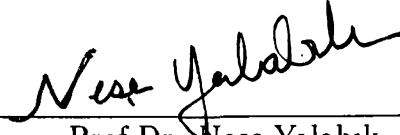
Q
337
.P65
1993

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



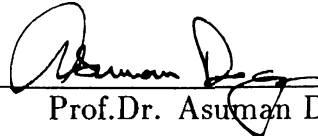
Assist.Prof.Dr. H.Altay Güvenir (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



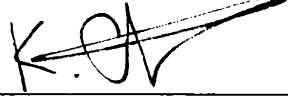
Prof.Dr. Neşe Yalabık

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



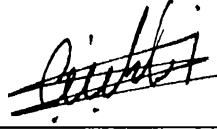
Prof.Dr. Asuman Doğaç

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



Assist.Prof.Dr. Kemal Ofazer

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



Assist.Prof.Dr. İlyas Çiçekli

Approved by the Institute of Engineering and Science:



Prof.Dr. Mehmet Baray,
Director of the Institute of Engineering and Science

Abstract

A NEGOTIATION PLATFORM FOR COOPERATING MULTI-AGENT SYSTEMS

Faruk Polat

Ph. D. in Computer Engineering and Information Science

Supervisor: Assist.Prof.Dr. H.Altay Güvenir

1993

Research in Distributed Artificial Intelligence attempts to integrate and coordinate the activities of multiple, intelligent problem solvers that interact to solve complex tasks in domains such as design, medical diagnosis, business management, and so on. Due to the different goals, knowledge and viewpoints of the agents, conflicts might arise at any phase of the problem-solving process. Managing diverse knowledge requires well-organized models of conflict resolution. In this thesis, a computational model for cooperating intelligent agents which openly supports multi-agent conflict detection and resolution is described. The model is based on the insights that each agent has its own conflict management knowledge which is separated from its domain level knowledge. Each agent has its own conflict management knowledge which is not accessible or visible to others. Furthermore, there are no globally known conflict resolution strategies. Each agent involved in a conflict chooses a resolution scheme according to its self-interest. The problem-solving environment allows a new problem solver to be added or an

existing one to be removed, without requiring any modification of the rest of the model, and therefore achieves open information system semantics.

Keywords: Distributed Artificial Intelligence, Conflict Detection, Conflict Resolution, Conflict Management Knowledge, Open Information System

Özet

YARDIMLAŞAN AKILLI SİSTEMLER İÇİN BİR MODEL

Faruk Polat

Bilgisayar ve Enformatik Mühendisliği

Doktora

Tez Yöneticisi: Assist.Prof.Dr. H.Altay Güvenir

1993

Dağıtık yapay us alanındaki araştırmalar, tasarım, tıp, iş yönetimi gibi karmaşık alan problemlerini çözmek için bir araya gelen birden fazla akıllı problem çözücünün çalışmalarının birleştirilmesini ve koordine edilmesini amaçlar. Problem çözücülerin değişik amaç, bilgi ve bakış açılarına sahip olmaları, problem çözümü aşamalarında çelişkilerin ortaya çıkmasına sebep olur. Dağıtık bilgi yönetimi çok iyi organize edilmiş çelişki yönetimi modellerini gerektirir. Bu tez çalışmasında, çoklu çelişki bulumu ve çözümüne dayalı bir yardımlaşan akıllı sistemler modeli anlatılmaktadır. Bu modelde her problem çözücü alan bilgisinden ayrı olarak çelişki yönetimi bilgisine sahip olup bu bilgi diğer problem çözücüler tarafından bilinmemekte ve erişilememektedir. Böylece tüm problem çözücüler tarafından bilinen bir çelişki yönetimi bilgisi bulunmamaktadır. Her problem çözücü, çelişkinin giderilmesine kendi çelişki yönetim bilgisine dayanarak katkıda bulunur. Geliştirilen model, yeni bir problem çözücünün sisteme entegre olması veya sistemden ayrılmasına olanak verirken, modelin hiç bir şekilde değiştirilmesini gerektirmez. Böylece model açık bilgi sistemleri mantığına erişir.

Anahtar

sözcükler: Dağıtık Yapay Us, Çelişki Bulumu, Çelişki Çözümü, Çelişki Yönetimi Bilgisi, Açık Bilgi Sistemleri.

Acknowledgement

The author wishes to express his deepest gratitude to Assist.Prof.Dr. H. Altay Güvenir for the valuable guidance, patience and support throughout all steps of the development of the thesis work. Dr.Güvenir's invaluable emphasis on various aspects of the thesis have enriched the author's appreciation, understanding and knowledge of the field of Artificial Intelligence and Computer Science. The author expresses his gratitude to the members of the Ph.D. committee for their beneficial comments and remarks.

The author is also deeply appreciative of the beneficial discussions and contributions received from Assist.Prof.Dr. Shashi Shekhar during his stay as a visiting NATO scholar at the Department of Computer Science of University of Minnesota, Minneapolis in 1992-93 academic year. In addition, sincere appreciation is extended to Assoc.Prof.Drs. Maria Gini and Jaideep Srivastava in the same department for their valuable advices on identifying the characteristics of the computational model developed. Thanks to Assoc.Prof.Dr. Varol Akman of Bilkent University for his valuable discussions and comments on AI based design-problem solving. The author would like to thank Drs. Shashi Shekhar, Suzan Lander and Mark Klein for their valuable comments that greatly contributed to the formation of the thesis proposal.

The author is grateful to his colleagues, Ahmet Coşar, Reda Alhajj, Uğur Güdükbay, and Veysi İşler, for their continuous encouragements in all stages of this study.

Thanks to my parents, Sait and Hakime Polat, brother, Fazıl, and sister, Naşide, for their continued encouragements and moral support that greatly helped to get this work completed.

Contents

1	Introduction	1
1.1	Our Approach	4
1.2	Organization of the Thesis	5
2	Distributed Problem Solving	7
2.1	Cooperation in DPS	9
2.2	Goals of Cooperation	10
2.3	Approaches to DPS	11
3	Cooperating Expert Systems	14
3.1	Previous Work	15
3.1.1	Blackboard Architectures and GBB	15
3.1.2	Hearsay-II	17
3.1.3	The Contract Net Protocol	18
3.1.4	The Distributed Vehicle Monitoring Testbed	20
3.1.5	Other Frameworks	22
4	Conflict Management	24
4.1	Models of Human Conflict Resolution	25
4.1.1	Aspects of Negotiation	25
4.1.2	Integrative Agreement Between Two Parties	27
4.1.3	Third-Party Intervention	29
4.2	Computational Models	31

List of Tables

7.1	Evaluation Results for proposal_0, proposal_1 and proposal_2	82
7.2	Evaluation Results for proposal_3, proposal_4 and proposal_5.	84
7.3	Evaluation Results for proposal_10, proposal_11 and proposal_12	90

5	The Computational Model	37
5.1	Why Cooperative Design	38
5.2	Architecture of the Model	38
5.2.1	Representation of Problem and Knowledge	39
5.2.2	Shared Medium	45
5.2.3	Agent Model	48
5.2.4	Conflict Resolution Knowledge	51
5.3	Problem-Solving Phases	54
6	Multi-Agent Conflict Management	58
6.1	Multi-Agent Conflict Detection	60
6.1.1	Computation of Degree of Satisfaction	60
6.1.2	Conflict Detection Algorithm	63
6.2	Multi-Agent Conflict Resolution	65
7	Examples of Cooperating Experts Problems	71
7.1	Office Design	71
7.2	Configuring A Personal Computer	85
8	Conclusions and Future Work	92
	References	96
	Appendix	
A	A Sample Run	108

List of Figures

1.1	Classification of DAI	2
3.1	An Example GBB Blackboard Structure	16
3.2	Task Announcement, Bid and Award Message Examples	19
3.3	The DVMT Problem-Solving Architecture	21
5.1	The Architecture of the Proposed Cooperative Design Environment	39
5.2	Example of a task sequence to be performed by a set of agents. . .	43
5.3	The Shared Blackboard	46
5.4	Internal Structure of a Design Agent in the Model	50
5.5	Problem Solving Steps within Agent α_i	56
6.1	Proposal Evaluation and Conflict Detection within Agent a_i . . .	64
6.2	Conflict Resolution Steps within Agent α_i	66
6.3	Conflict Resolution Steps for Agent α_i in Constraining Search Space.	67
6.4	Conflict Resolution Steps for Agent α_i in Counter-Proposing Alternatives.	68
6.5	Conflict Resolution Steps for Agent α_i in Selecting an Alternative.	70
7.1	Global Layout of the Office	75
7.2	Layout of the Office After proposal _{1_0}	76
7.3	Layout of the Office After Resolution Alternative proposal _{1_1}	78
7.4	Layout of the Office After Resolution Alternative proposal _{1_2} . .	78

Chapter 1

Introduction

Distributed Artificial Intelligence (DAI) is a subfield of artificial intelligence which is concerned with solving problems by using both AI techniques and distributed processing capabilities. A DAI system must include at least two agents and requires that these agents have some degree of information and/or control autonomy, and that some nonempty subset of the agents display sophistication in an artificial intelligence sense (capable of reasoning, planning, etc). DAI is different from distributed processing in that it does not only distribute data, as in the case of distributed processing, but also control. In addition, DAI involves extensive cooperation among problem solvers.

Distributed processing systems address the problem of coordinating a network of computing systems to carry out a set of disparate and mostly independent tasks. There is much less interdependence between tasks in distributed processing than in DAI. This often leads to a concern with issues such as access control and protection, and results in viewing cooperation as a form of compromise between potentially conflicting views and desires at the level of system design and configuration. DAI, on the other hand, aims at combining approaches existing in many disciplines such as negotiation, interaction, contracts, agreement, organization, cohesion, etc [8, 14, 15, 16, 24, 34, 48, 53, 83]. Generally speaking, in terms of level interaction between processes, there are two approaches in AI to distributing data and control for achieving cooperation (Fig. 1.1) :

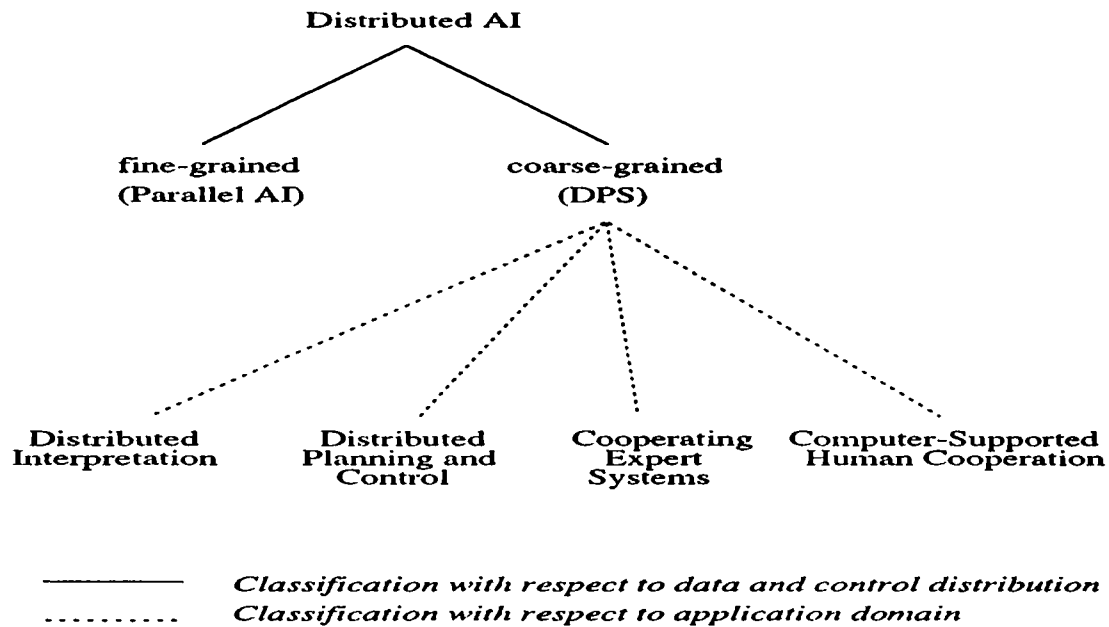


Figure 1.1: Classification of DAI

- Fine-grained (Parallel AI)
- Coarse-grained (DPS)

Parallel AI is concerned with developing parallel computer architectures, languages and algorithms for AI, whereas DPS considers how the task of solving a particular problem can be divided among a number of problem-solving agents which cooperate at the level of dividing and sharing knowledge about the problem and about the developing situation. DPS aims at conceptual advances in understanding the nature of reasoning and intelligent behavior among multiple agents. On the other hand, Parallel AI deals with solving performance problems of AI systems. Although Parallel AI is considered as a sub-discipline separate from DAI, it is important to note that developments in concurrent programming languages and architectures may have profound impacts on DAI system architectures, reliability, knowledge representation and so on.

One of the application domains of DPS (Fig. 1.1) is *cooperating expert systems*. Cooperating expert system approach is concerned with solving complex

tasks that require diverse expertise to generate comprehensive solutions. When human specialists cooperate, they bring together multiple disciplines or multiple viewpoints on a single problem. Bringing together diverse knowledge is a source of robustness and balance which is extremely important in many real-world situations: a civil engineer and an architect work together to design and build a safe and attractive building, or a pediatrician and a cardiac specialist consult to help an infant with a heart problem. The team can solve problems that are beyond the scope of any of the individual experts and the solutions are generated from a rich and varied body of knowledge, providing the potential for creativity and innovation.

Although diversity is beneficial in some respects, there are also difficulties in handling the conflicts that arise from trying to merge multiple perspectives for a common good. Managing diverse expertise is difficult because one has to take into account the problems which will arise in working out solutions in the face of conflicting goals, constraints, viewpoints, and knowledge of heterogeneous experts. Consider a team of human experts who are cooperating in choosing a computer system for a company. The team consists of a computer specialist and a manager. They have the shared goal of selecting an appropriate computer system for their company, but each expert wants to insure that his own perspectives should be reflected in the final solution appropriately. The computer specialist recommends a UNIX-based workstation, a computer system known with its quality in networking and graphical capabilities. The manager recommends a classical DOS-based personal computer because of its lower cost and his and other middle manager's acquaintance with DOS. In this conflicting situation, it is necessary for the two agents to reconcile their difference to reach a globally agreed solution.

Resolution of conflicts usually is accomplished through exchange of information among participants. How to exchange, what to exchange, when to exchange, and who to exchange it with, are questions that have to be considered in developing computational models for conflict resolution. Applications of cooperating expert systems can be seen in human problem-solving tasks such as design, medical diagnosis, research, business management, and human relations. Computer

models of conflict resolution borrows many ideas from these natural application domains.

1.1 Our Approach

In this dissertation, we present a formal computational model, called *NEPTUNE*¹, in which a set of knowledge-based agents cooperate for solving design problems. Throughout this thesis, we use the terms *agent*, *expert* and *problem-solver* interchangeably to refer to the autonomous knowledge-based systems. The model is based on resolution of conflicting solutions generated by experts having different goals, priorities, and evaluation criteria. Many of the existing approaches to conflict management [1, 43, 46, 99] rely on coordinated resolution strategies which require resolution of a conflict based on a globally agreed strategy. In these systems, conflict resolution knowledge is maintained centrally. In any case, one of the disputants is given the power to take control of the conflicting situation and apply a resolution scheme known to everybody. A mediator or a single agent from the team would not have enough detailed knowledge about the problem to be able to make good decision outside of its own expertise.

The novel approach in *NEPTUNE*, however, allows agents to freely choose the most appropriate action, given their understanding of the global and local situations and their own capabilities. They maintain their own set of conflict management knowledge which is not globally known. Using their own conflict knowledge, the participants may come to an agreement on a revised solution. Agents know the reasons behind their decisions and are able to anticipate the impact of various revisions. *NEPTUNE* allows conflicts to be resolved through negotiating agents that act based on their local perspectives of the global situation. *NEPTUNE* is designed for solving problems in the domain of design and is based on the insights that, each agent has its own conflict knowledge separate from its domain-level knowledge, and that this knowledge can be instantiated in

¹*NEPTUNE* stands for NEgotiaton PlaTform for cooperating mUlti-ageNt intelligent systEmS

the context of particular conflicts into specific resolutions. Each agent's conflict knowledge centers around the domain issues through which that agent is contributing to the global solution. Agents are able to evaluate partial solutions to tasks through different issue-perspectives and negotiate over conflicting solutions cooperatively. This is similar to the resolution of conflicts that occur among human beings when solving a problem.

There are several reasons why conflicts need to be resolved by using agents' private conflict resolution knowledge instead of global knowledge about conflict resolution. First of all, this task is very similar to the resolution of conflicts that occur among human beings in solving complex problem tasks in domains like design, diagnosis, business management, etc. When a conflict is detected, it is not resolved by a central authority using global conflict resolution knowledge, but rather by specialists who are involved in the conflict and negotiate a revised solution that will be acceptable to all of them, using their own conflict resolution knowledge and perspectives. Second, global conflict resolution requires consistent merging of the conflict resolution knowledge obtained by each agent. This makes the maintenance of global knowledge difficult. Because when a new agent is added to, or removed from the system, or the conflict resolution knowledge of an agent is revised, the global conflict resolution knowledge must be rebuilt accordingly. Lastly, distribution of knowledge leads to increased reliability and fault-tolerance to agent failures.

1.2 Organization of the Thesis

In the next chapter, an overview of DPS is presented by emphasizing the importance of coherent cooperation and coordination. Existing approaches to the coordination of cooperating agents and application domains in DPS are outlined.

In Chapter 3, cooperating experts approach is described and some typical systems supporting this approach are introduced.

Chapter 4 describes conflict management issues in cooperating expert systems

and summarizes the existing approaches. First, models of human conflict resolution are introduced, and then the existing computational models for conflict resolution, development-time and run-time models, are described.

Chapter 5 explains the new model, *NEPTUNE*, for cooperating experts, and how the problem-solving proceeds within it. The architecture of *NEPTUNE*, representation of knowledge, the agent model, and the main problem-solving steps are presented in details.

Chapter 6 describes how conflict resolution takes place in *NEPTUNE*. The methodology used for jointly detecting and resolving conflicts is presented, along with the relevant algorithms.

Chapter 7 includes two examples from the domain of design to illustrate how problem-solving proceeds in *NEPTUNE*. The examples are chosen from the domains of office design and computer hardware configuration.

Chapter 8, the last chapter, summarizes the novel approach presented in the dissertation and states its contribution to the area of computational models of conflict resolution, and also suggests future work.

Chapter 2

Distributed Problem Solving

In DPS, a group of individual agents come together to solve a difficult global problem. There are four phases in solving a problem cooperatively by several agents. In the first phase, the original problem is decomposed into simpler ones. In the second phase, these subproblems are distributed to the most capable and relevant agents. In the third phase, subproblems are solved cooperatively. The last phase requires the synthesis of the subproblem solutions to obtain a global solution for the original problem which is acceptable to all agents [16, 25, 28, 41, 54, 82].

Advances in hardware technology for processor construction and interprocessor communication make it possible to connect together large numbers of sophisticated processing units that execute asynchronously. Various connection structures are possible, from a very tight coupling of processors through shared or distributed memory, to a looser coupling of processors through a local communication network or to a very loose coupling of geographically distributed processors through a communication network.

Besides the rapid development in processor and communication technology, among the several other reasons that motivate researchers to explore new ideas about problem-solving which requires multiple agents are:

- Many AI applications are inherently distributed. The applications may be

spatially distributed, such as interpreting and integrating data from spatially distributed sensors or controlling a set of robots that work together on a factory floor. It is also possible to have the applications being *functionally* distributed, such as bringing together a number of specialized medical-diagnosis systems on a particularly difficult case. Finally, the applications might be *temporally* distributed, as in a factory where the production line consists of several work areas, each having an expert system responsible for scheduling orders.

- Sometimes problems are simply too large or complex to be solved by a single problem solver. Such problems could only be solved via the cooperation of several independent systems (*synergy effect, emergent functionality*). For instance, multiple expert systems with different, but possibly overlapping expertise, could cooperate to deal with problems that are outside the scope of a particular expert system.
- A DAI system supports the principles of modular design and implementation. The ability to structure a complex problem into relatively self-contained processing modules leads to systems that are easier to build, debug and maintain. For example, the general field of medical diagnosis is complicated and extensive. To manage the field, medical experts divide it into many specialties. In order to build a general medical diagnosis system, someone could exploit the modularity of the field, building a knowledge-based system for each specialty in parallel and with minimum interaction between the systems.
- The environment in which both control and data are distributed should result in reliable computation and graceful degradation. That is, the failure of one agent, for example, should not crash the whole system down.
- One of the goals of AI is to develop systems which are essential for our daily life. These systems should interact with humans intelligently. To achieve this, these systems they must have the ability to intelligently cooperate and

coordinate with each other and with humans with more flexibility. DAI is the first step towards this long-term goal.

However, many AI problems cannot be decomposed into independent subproblems. Furthermore, it is often impossible to solve the subproblems in isolation. Even if they were, merging the independently formed solutions would be difficult. Therefore, problem-solving agents should cooperate at every phase of problem-solving. It is also necessary to develop appropriate control regimes that allow the coordination of activities of cooperating problem solvers.

2.1 Cooperation in DPS

Cooperation where no single agent has sufficient expertise, resources and information to solve a problem independently is an important area of research in the field of DPS [7, 11, 14, 21, 22, 52, 72, 73, 78, 79, 100]. Different agents might have the expertise necessary for solving different parts of the problem. For example, consider the problem of designing a steam condenser. One agent might have expertise on the motor, another on the heat exchanger, yet another on the pump components of the steam condenser, etc.

The agents in a DPS network might utilize different resources. Some might be very fast as far as computation is concerned, a third party might have connections that speed up the communication, while others might have excess memory. Finally, different agents might have different information or viewpoints regarding a certain problem. For example, consider a distributed sensing network where geographically separated agents are monitoring aircraft movements. In this case, different agents will have different perceptions because their sensors will pick up different signals. It is possible to form an overall picture of aircraft movement only when agents combine the information about their views.

The amount of cooperation between agents is an important aspect of DPS. It may range from fully cooperative to antagonistic. In fully cooperative systems, there is a high price due to the heavy communication between agents. In antagonistic systems, on the other hand, there is no communication cost since

agents may not cooperate at all. The extent to which agents should cooperate in solving a problem is dependent upon the application domain and is currently an attractive research area.

2.2 Goals of Cooperation

Agents cooperate to improve their own self-interests by sharing subproblem solutions. Cooperation thus requires intelligent local decisions so that each agent performs tasks that generate useful subproblem solutions. There are several goals to be achieved by the agents which are cooperating:

- Improved performance through parallelism (several agents solve different parts of the whole problem concurrently),
- Increase in the confidence of a subsolution by letting agents verify each other's results (in order to get consistent results, agents use their own expertise to interpret the shared data),
- Exchange of tasks among agents by allowing a task to be performed by the most capable agent (fair utilization of agents' computational resources),
- Assigning of important tasks to multiple agents to guarantee a solution even in the presence of agent failures (reliability), and
- Improving the use of individual agent expertise through the exchange of goals, constraints, partial solutions, and knowledge.

Problem solvers cannot achieve all of these goals simultaneously. While concentrating on the achievement of one goal, usually it is not possible to achieve some of the other goals. For example, in a problem where a solution is to be generated as fast as possible, it is necessary to avoid actions causing inter-agent communication. In this case, although we achieve an improved performance through parallelism, we may not improve the use of individual agent's expertise through exchange of partial results and knowledge.

2.3 Approaches to DPS

Coordination of activities in a multi-agent environment is a very important issue in DPS. There are several approaches to improve the coordination among cooperating agents in a distributed problem-solving environment [62, 16]. These are

- Negotiation,
- Functionally-accurate cooperation,
- Multi-agent planning, and
- Organizational structuring.

In the negotiation approach [10, 44, 83, 84], a problem task is decomposed into a set of subtasks which are assigned to the agents based on a *bidding protocol*. Since different agents may have different capabilities, the bidding protocol will offer the opportunity for a task to be assigned to the most appropriate agent. The negotiation approach allows effective use of computing resources and expertise through the exchange of tasks. It also facilitates the generation of reliable solutions through assignment of the same tasks to several agents having different expertise and problem-solving capabilities.

In functionally-accurate cooperative systems [49], agents cooperate by generating and exchanging tentative, partial solutions based on their limited local views of the network problem. By iteratively exchanging their potentially incomplete, inaccurate, and inconsistent partial solutions, the agents eventually converge on an overall network solution. This approach allows the agents to generate solutions without being overly influenced by each other.

In multi-agent planning [8, 27, 28, 36, 42, 47, 80, 81], agents form a multi-agent plan which specifies all of their future actions and interactions. Multi-agent plans can be generated in a centralized or distributed way. In centralized multi-agent planning, agents agree on an agent to solve their planning problems and all pertinent information is sent to that particular agent. On the other hand, in

distributed multi-agent planning, agents cooperatively generate the plan. This is necessary, especially, in an environment where no single agent has a global view of the problem and the environment.

In organizational structuring [17, 21], common knowledge about general problem-solving roles and communication patterns are used to guide agents about how to cooperate. An organizational structure is the pattern of information and control relationships that exist among agents, and the distribution of problem-solving capabilities among agents. Imposing a high level organization on DPS environment gives agents knowledge that improves the way they coordinate, while still allowing them to pursue alternative solution paths that are not dictated by the network.

To achieve coherent cooperation, agents must predict each others' actions during any phase of the problem-solving process. Multi-agent planning requires accurate predictions in order to form acceptable plans. However, negotiation and functionally-accurate cooperation can perform without adequate predictions. Negotiation takes a top-down view of problem-solving while functionally-accurate cooperation takes a bottom-up view. Organizational structuring lies between the strongly top-down view of contracting and bottom-up view of functionally-accurate cooperation.

There are several classes of application domains where DPS is applicable. Some of them are

- *Distributed Interpretation*: These applications require the integration and analysis of distributed data to generate a potentially distributed semantic model of data. Application domains include distributed sensor networks [50, 51] and communication network fault diagnosis [10, 11, 93, 97, 98].
- *Distributed Planning and Control*: These applications involve developing and coordinating the actions of distributed effector agents to perform desired tasks. Application domains include distributed air traffic control [20], cooperating robots, remotely piloted vehicles [87], and distributed process control in manufacturing [59].

- *Cooperating Expert Systems:* In these applications, several expert systems work together to solve a common problem. The heterogeneous character of cooperating experts allows different problem-solving approaches to be used in solving the problem under consideration. Application areas include medical diagnosis [9] and engineering design [40, 43, 45, 46].
- *Computer-Supported Human Cooperation:* Intelligent systems with coordination knowledge assist humans in decision making, through filtering information and focusing attention on relevant information. Application domains include intelligent command and control systems, and multi-user project coordination [13, 58, 74].

There are several technology platforms built for implementing various DPS systems. These include testbeds such as the Contract Net Framework [83], DVMT [50], MACE [26], integrative systems such as ABE [18], blackboard systems such as GBB [12], BB1 [35], object-based tools such as ORIENT84/K [94], belief-based systems such as Agent0 [76, 77, 100].

Chapter 3

Cooperating Expert Systems

In the cooperating experts approach, several specialized agents work together to solve a global problem. Examples of the integration of expertise through cooperation can be seen in human problem-solving tasks such as design, diagnosis, business management, and human relations. As a subfield of DPS, cooperating experts approach has the following distinguishing characteristics:

- agents are heterogeneous in their problem-solving capabilities and knowledge,
- the knowledge of each agent is potentially inconsistent or incompatible with that of others,
- agents are logically independent and negotiate with each other around interacting subproblems, and
- local goals, constraints, priorities, and evaluation criteria may be conflicting among agents.

3.1 Previous Work

In the following sections, we briefly describe blackboard architectures and GBB (Generic BlackBoard), Hearsay-II, the Contract Net Protocol, Distributed Vehicle Monitoring Testbed, and some other frameworks which are typical examples reflecting the cooperating experts approach.

3.1.1 Blackboard Architectures and GBB

The blackboard architecture [19, 35, 56, 57] is one of the architectures that can be used to implement a cooperating experts system application. The blackboard problem-solving approach offers superior flexibility in structuring complex AI applications. Blackboard systems perform problem-solving by using three basic components:

- *a blackboard*, which is a global database containing input data, partial solutions, and other data that are used in various problem-solving phases,
- *knowledge sources* (KSs), which are independent modules that contain the knowledge needed to solve the problem, and that can be widely diverse in representation and in inference techniques. KS modularity facilitates application development and simplifies maintenance and enhancement, and
- *a control mechanism*, which is separate from the individual KSs and makes dynamic decisions about which KS is to be executed next.

GBB [12, 23] is a flexible, high-level tool for building efficient blackboard systems. GBB provides the following facilities that developers need in constructing high-performance blackboard applications:

- a blackboard database compiler and runtime library, which support pattern-based, multidimensional range-searching algorithms for efficient retrieval of blackboard objects,
- KS representation languages,

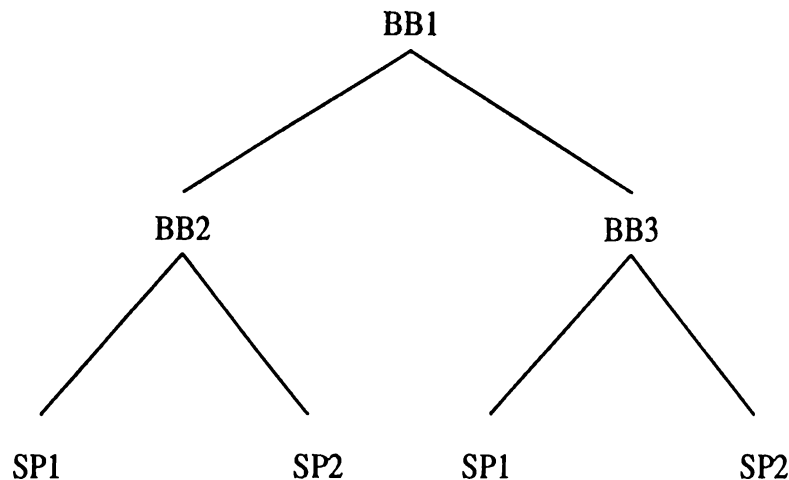


Figure 3.1: An Example GBB Blackboard Structure

- generic control shells and agenda-management utilities, and
- interactive graphics for monitoring and examining blackboard and control components.

GBB views the blackboard as a hierarchical forest of nested blackboards. Blackboards are the containers for holding spaces and other blackboards. Blackboard objects reside in *spaces*, which are the leaves of this hierarchy. Each space can be defined as a highly structured, n -dimensional volume, with blackboard objects occupying some extent within the space. Spaces can be viewed as the “containers” that hold blackboard objects (called *units*). Units contain *slots* which hold data values and *links* which are special purpose slots that contain link pointers between units. The space on which a unit is to be stored can be specified by the sequence of nodes traversed from a root blackboard node through all intermediate blackboard nodes to the leaf space node. For example, if the blackboard BB1 had the blackboards BB2 and BB3 as components, and BB2 and BB3 had spaces SP1 and SP2 as components as shown in Figure 3.1, the two paths (BB1 BB2 SP1) and (BB1 BB3 SP1) specify different instances of the space SP1.

Efficient insertion and retrieval of blackboard objects is achieved using a language specifying the dimensional structure of each space and a separate specification of how that space is to be implemented. Any change to a blackboard object or to the state of the blackboard database is called an *event* in GBB terminology. For example, creation or deletion of a unit, access or modification of a slot, and access or modification of a link are typical examples of events. When an event occurs, *event triggering mechanism* of GBB instantiates those knowledge sources that declared their interest in this particular kind of event. They are scheduled and executed in a priority-based manner. Activation of a KS is managed by a centralized scheduler.

There are several problems in developing distributed knowledge-based system applications on GBB. The autonomous behavior of agents in a typical application must be simulated by using knowledge sources of the GBB. However, event triggering mechanism of GBB does not allow real autonomy since knowledge sources are specific modules that are executed upon particular events on the blackboard.

3.1.2 Hearsay-II

Hearsay-II [19] is a continuous speech understanding system developed at Carnegie Mellon University. It can be considered as the first system using the cooperating experts approach. It consists of a set of knowledge sources, a blackboard, a priority based task scheduler, and a focusing mechanism for meta-level control. The original Hearsay-II did not have a distributed architecture. Later on Lesser and Erman developed a distributed Hearsay-II architecture which consisted of a set of functionally-accurate complete Hearsay-II systems, each one with its own blackboard, sampling one time-continuous segment of the speech signal.

Since speech processing is time localized except for the highest semantic levels, these systems need only to exchange high level intermediate results consisting of phrase hypotheses. They could converge on complete interpretations despite the loss of some messages. This kind of communication is nearly ideal, but distributing a generic Hearsay-II architecture in this way would not necessarily work

for other applications. This work highlights a basic trade off between communication and computation in DPS networks. The more communication takes place the more reduced the inconsistency is because agents will have more common information. Less communication leads to more inconsistency and causes agents to spend more effort to resolve inconsistencies.

Cooperation among specialized knowledge sources occurs implicitly through the incremental extension of globally available hypothesis. Knowledge Sources containing expertise are instantiated in response to a particular pattern on the blackboard. They cannot be suspended or reinstated once execution has terminated. They also do not keep the history of their actions.

3.1.3 The Contract Net Protocol

The Contract Net Protocol developed by Smith and Davis [83] provides a general paradigm to design cooperating expert systems in a distributed environment. It models transfer of control in a distributed system with the metaphor of negotiation among autonomous intelligent agents. The Contract Net consists of a set of nodes (agents) that achieve the desired goal of coordinating their activities through contracts. There are three classes of nodes in the net: *manager*, *bidder*, and *contractor*. The manager is the node that identifies a task to be done, and assigns it to other nodes for execution. The bidders are the nodes that make offer to perform the task announced by the manager. The contractor is a successful bidder, the one whose bid has been accepted by the manager.

Contracting occurs through exchange of information between interested parties followed by a final agreement by mutual selection depending on the available information. It differs from voting in that parties are free to exit the process rather than being bound by the decision of the majority. At the beginning, the manager receives a large task and decomposes it into smaller subtasks in a pre-defined way. It announces the task to the idle nodes. Nodes that have enough resources, expertise, and information to perform the announced task send their bids to the manager. Later, the manager evaluates the bids and awards the task

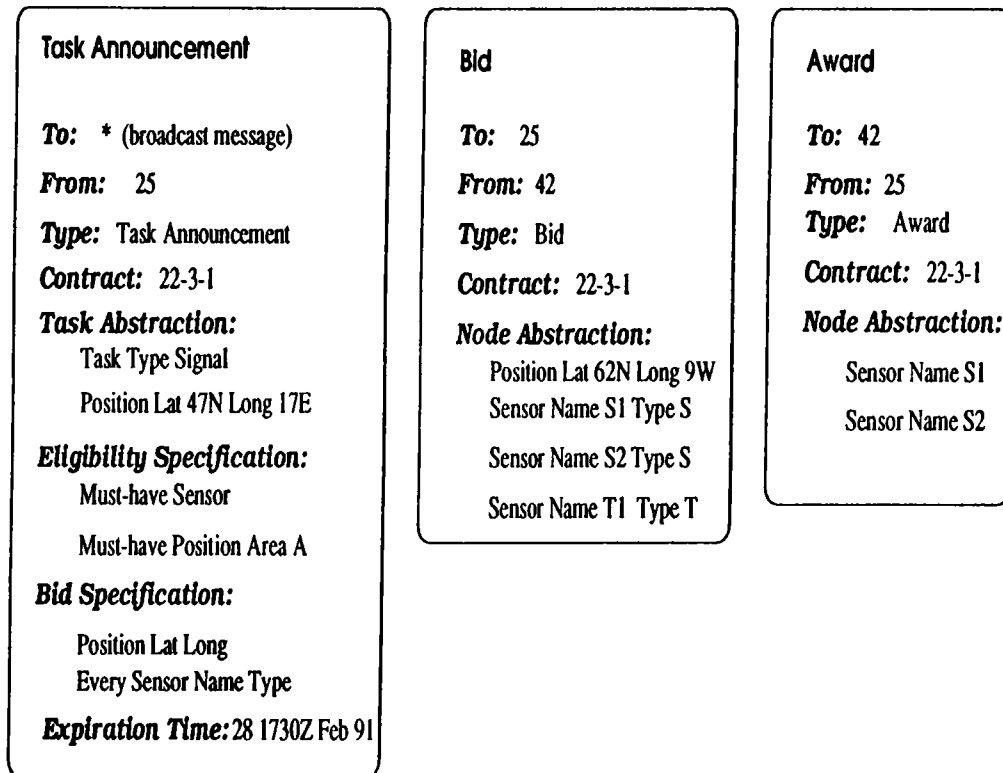


Figure 3.2: Task Announcement, Bid and Award Message Examples

to the most capable node. Afterwards, the manager sends the task information to the contractor who reports back the progress and eventually the final result of the tasks. The manager may choose to award the task to several nodes if it wants to increase reliability.

Smith and Davis investigated a distributed interpretation application in the Contract Net Framework, where the network should track vehicles over a wide geographical area. The network contains two types of nodes. Sensor nodes extract signal features from the data they sense, manager nodes process signals obtained from different sensors to construct a map of vehicle movement. A manager node tries to form contracts with sensor nodes through exchange of messages. Every message includes information about its source, destination, type, and contract identifier. Fig. 3.2 shows the use of three messages (task announcement, bid, and award) for the distributed sensor net applications.

Task announcement message contains abstract information about the task, expected capabilities for potential contractors, information that a bid should contain and a deadline for the bids to be received. In vehicle monitoring, task abstraction indicates the task type and manager's location. The expected capabilities specify the sensory capabilities, and location of a potential contractor, and bid specification includes the sensor's location and sensory abilities. A task bid message includes the information requested in the task announcement's bid specification. In this application, bid information consists of the position and sensory capabilities of the sensor node. After evaluating bids, manager issues a task award message for each node that is awarded the task. In this application, the message specifies which of a sensor's sensory capabilities are to be utilized.

The Contract Net Framework is concerned with the allocation of tasks to several problem solvers through a bidding protocol. It requires a top-down decomposition of large tasks and the allocation of the subtasks to appropriate agents. It is well-suited for applications with well-defined task hierarchies, and for cases in which tasks are initially presented to a few nodes in the network.

3.1.4 The Distributed Vehicle Monitoring Testbed

Distributed Vehicle Monitoring Testbed (DVMT) simulates a network of cooperating expert systems, called nodes, to track vehicle movement using sounds recorded by acoustic sensors [50]. The spatially distributed nodes detect the sounds of vehicles, and each applies the knowledge of vehicle sounds and movements to track a vehicle over its spatial area. Nodes exchange information about vehicles they have tracked to build a map of vehicle movements through the entire area. There is a need for organizational structuring to guide their processing and communication decisions, otherwise nodes may overwhelm each other with tentative partial results.

For the specification of organization in DVMT, each node is associated with a set of interest areas which defines what, when and to whom information should be transmitted. It also indicates how much priority should be given to processing

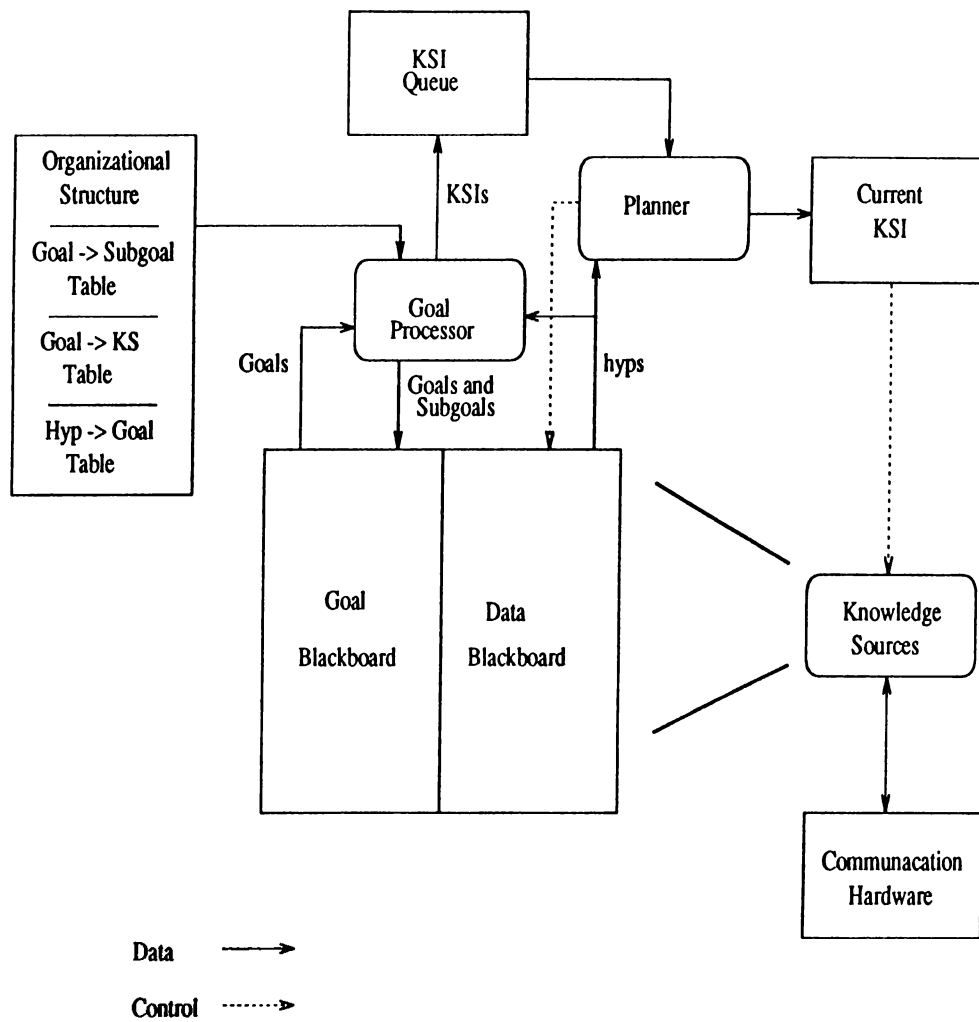


Figure 3.3: The DVMT Problem-Solving Architecture

externally received goals versus internally generated goals. Each node in DVMT is represented as a blackboard-based problem solver, with levels of abstraction and knowledge sources appropriate for vehicle monitoring (Fig. 3.3). A knowledge source performs the basic problem-solving tasks of extending and refining hypotheses. Each hypothesis tentatively indicates where a certain type of vehicle was at different discrete sensed time points. In this model, the basic Hearsay-II architecture has been augmented to include more sophisticated local control and the capability of communicating hypotheses and goals among nodes. In particular, a goal processing module and communication knowledge sources have been added.

3.1.5 Other Frameworks

Georgeff [27] has developed a framework in which agents cooperate without explicit communication. It requires that each agent must have complete knowledge of others' abilities and payoffs. This is, however, too restrictive for the majority of real-world problems. One of the motivations for developing a multi-agent environment is to allow agents to negotiate solutions when they do not have good models of each other's abilities and payoffs.

Shekhar [75] developed a shell for cooperating expert systems, called *Coop*. It supports cooperation models to characterize three essential decisions in the cooperation process. These reason about the need for cooperation, understanding global knowledge to locate relevant expert systems and selecting appropriate cooperation plans. *Coop* environment enables experts to autonomously resolve the three fundamental decisions at run-time. Each agent in *Coop* uses a common representation which combines a theory of fuzzy sets with a theory of evidence and logic programming. After performing fuzzy set computations, an expert decides whether it can achieve a given goal independently, or needs help from others.

Chandrasekeran [9, 33], in the MDX approach, proposes a "cooperating community of specialists" where each specialist contains its own knowledge-base and

inference mechanism for solving disease diagnosis problems. MDX is based on a hierarchy of specialists with those at top being more general than those at lower levels. Task distribution is done according to the hierarchical structure of the system. For diagnosticians, this hierarchy serves the function of organizing their troubleshooting knowledge. The concrete details for each disease are encoded in the production rules attached to the appropriate concepts.

Chapter 4

Conflict Management

In the cooperating experts approach, several specialized agents combine to solve a common problem. During any phase of the problem-solving, conflicts might appear as a result of incorrect and incomplete local knowledge, different goals, priorities, and solution evaluation criteria. When there are several conflicting proposed solutions for a (sub)problem, the agents involved in a conflict must either agree to choose one proposal, cooperatively revise one, or search for a new solution that will be acceptable to everyone.

A conflict may be either direct or indirect. A *direct conflict* occurs when two or more agents have beliefs that are explicitly inconsistent. This type of inconsistency is due to the uncertainty inherent in the domain. For example, when several agents are designing an office, an agent might prefer a round desk while another insists on a rectangular one. An *indirect conflict* occurs when agents have constraints that do independently contain a shared object. For example, in designing an office the functionality expert wants to place the PC desk in front of a window, while the electricity expert says that the PC desk must be within 2 meter of the electrical plug, because the PC comes with a 2m long main power cable. This is a problem if the only window in the office is 4 meter away from the electrical line. The inconsistency is not inherent in the knowledge, rather it is due to a particular configuration of objects. Detection of conflicts in cooperating experts is a difficult task and is actually dependent on the problem domain.

There are several methods that have been used to resolve conflicts. Existing research related to conflict resolution in cooperative problem solving can be divided into two categories: models of human conflict resolution and computational models.

4.1 Models of Human Conflict Resolution

There are considerable amount of work concerning the resolution of conflicts that occur between individuals or groups of individuals in domains such as business, international relations and so on [71]. However, most of these techniques cannot be directly applied to computational models because much of the work addresses issues specific to the psychology of human participants that are not present in machine agents and that greatly influences the process. Human motivation is more complex than the state of our current understanding; for example, sometimes the disputants involved in a conflict do not know what they really want. As a result, the level of description of conflict resolution expertise is more abstract than appropriate for machine based agents.

4.1.1 Aspects of Negotiation

It is possible to make important observations in the human model of negotiation behavior that are worth considering for computational models. Initially, Pruitt [71] described levels of demand and rates of concession for parties involved in negotiation. Later, he described how the demand levels and concession rates can affect the motivation and expectations behind the proposal process. Each party's expectations can be described in terms of issue tracking, position and image loss as well as limit and level of aspiration. Each party uses a model of itself and its opponents to evaluate and generate proposals. Finally, the behavior of parties during negotiation can be influenced by the perceived power that one party maintains over the issues. We describe each of the aspects of negotiation in detail in the following subsections.

Demand Level and Concession Rate

A party's perceived benefit that it gains from any proposal it makes during negotiation describes a demand level. The other party's behavior can be somewhat determined by the demand level of the proposal of the first party. Research among human subjects showed that bargainers that make low initial demands tend not to agree or cause negotiation take longer [6]. The party responding to the initial weak demand expects further large concessions from the perceived weak party. This prevents the responding weak party from making large concessions of its own. The opposite behavior is seen when more aggressive proposals are made. In this case, aggressive proposals tend to require aggressive counter-proposals causing parties to quickly overshoot their final point of agreement in the negotiation. Therefore, it is recommended that good negotiations must avoid the hazard of moving too slowly or too quickly towards the agreement. A party's demand level can be measured in terms of the position the party takes on its issues. The concept of demand level is not so clearly defined in many computational models of negotiation.

Tracking, Position and Image Loss

The response of one party to another party's original proposal is based on a phenomenon called *tracking*. One party tracks the other party in order to estimate that party's ultimate demand level. Tracking a party's demand level is very useful when there is no other information available to help make this determination (such as facial expression, body language, etc.). A party's perceived position and image loss will affect the other parties' response behavior. The position of a party is described as its desired benefit it gains given its level of alternatives. Image loss is the impression the other party makes of the first party. If the first party abandons its position, the impression gained by the second party will be that the former lacks firmness in position. This can cause the second party to have a higher demand level.

Limit and Level of Aspiration

A negotiation limit is the absolute lowest value that a party will agree upon for its issues. This is the bottom boundary in a party's negotiation position. The top most negotiation boundary is determined by the parties' level of aspiration. A party's aspiration is its perceived value that seems attainable at any time during the negotiation [71]. Also, the level of aspiration will always be greater than or equal to the bargainers' limits. There are also interesting relationships between limit and level of aspiration:

- limit tends to remain constant over time, whereas aspiration declines towards the limit,
- limit and aspiration are positively correlated, and
- the strength of the correlation increases over time.

4.1.2 Integrative Agreement Between Two Parties

When parties want to develop an agreement which includes some aspects of each party's issues into the final solution, they can enter into a form of coordinated behavior which is called *integrative agreement*. In the literature, there are four types of integrative agreement: *cost cutting*, *compensation*, *log-rolling* and *bridging*. These four types can be classified into two groups. The first group represents behaviors which improve the other party's position without reducing the first party's position (cost cutting and compensation). The second group represents behaviors which change the position of both parties (log-rolling and bridging).

Cost Cutting

The first party makes a proposal that is intended to benefit the second party by reducing some cost of the second party. During this action, the first party maintains the same level of aspiration while attempting to entice the other to make some form of a concession. Cost cutting can also be done by a third party

such as an arbitrator. During cost cutting, parties need to exchange relevant information about their priorities. Cost cutting allows the parties to maintain their positions in addition to providing some form of negotiation for the other party's future concerns about an agreement.

Compensation

There are three types of compensation behaviors: *specific compensation*, *homologous compensation* and *substitute compensation*. Specific compensation can be interpreted as a type of cost cutting behavior. It provides the first party with some means of reducing tensions for the second party. Parties inform the others of their "worries." Through specific compensation, one party specifically compensates the other's issues by some secondary means. In homologous compensation, the first party concedes the same aspects for a similar concession made by the second party. The idea is for the first party to provide some type of benefit for the second party. This is achieved indirectly by demonstrating to the second party that the first party is losing an amount of benefits which is equal to what it is gaining.

In substitute compensation, parties propose substitutes for the other's requested issues. The first party can use a stereotype of the second as well as the extend of the second party's issue costs so as to produce an adequate compensation response. In general, this type of behavior requires a different realm of reality and therefore is not as useful in computational models of negotiation where mechanical parties do not respond emotionally to proposals as humans would.

Log-rolling

The first party exchanges or swaps a set of its issues with a set of issues from the second party. In this form of integrative agreement, each party changes current bargaining position, but in the most minimum fashion possible. For log-rolling to occur, parties have to know the priority ranking among the other party's issues.

Bridging

During bridging a new proposal is generated that benefits both parties in terms of their most important issues. Thus both parties change their negotiation positions, but for their common benefit. The new proposal can come in several forms. First, both parties agree to accept one party's issues this time if the other party's issues could be met during the next encounter. Bridging can also result from the resource shortage. In this case, parties must either schedule the resource themselves or rely on a third party for help.

4.1.3 Third-Party Intervention

Sometimes intervention by a third party-during negotiation is useful when the parties cannot develop any alternatives themselves. It has been shown that negotiators concede much faster when a suggested alternative is made by a third-party as opposed to conceding to alternatives made by the other party [30]. One idea for this behavior is that negotiating parties use a third party to legitimize their own interests in their proposals. In general there are three types of third-party intervention:

- mediation,
- fact finding, and
- arbitration.

Third-Party Mediation

A major area of research in human negotiation has been in the area of third-party mediation. Third-party mediation is a form of intervention which attempts to coordinate behavior in a cooperative fashion between disputing parties. Mediation may come about in several ways. First, each party can make a request for mediation if it predicts that other party is powerful and firm in its position. A time limit could also cause the disputing parties to request mediation which

would lead to a quick and equitable solution. Mediation can also occur through third-party observation of the negotiation process. The third-party could then intercede and make suggestions.

Mediators can help in several ways. First, they can set the “right” environment just by being present during negotiations. This type of negotiation is described as *process mediation*. Second, they can take a more active role and get involved in learning about the issues under discussion, which is named as *content mediation*.

Third-Party Fact Finding

A second form of third-party intervention is fact finding. During fact finding, the third party listens to both sides of the conflict and produces a set of non-binding recommendations. It is up to the parties to elect to follow the suggestion as a solution or to throw it out. Not much research has been performed in this area of human negotiation.

Third-Party Arbitration

The final form of third-party intervention is arbitration. During arbitration, the third-party performs services similar to the fact finding in order to learn about the parties issues. This allows the arbitrator to make recommendations based on these issues. Unlike fact finding, recommendations based on arbitration are binding. Arbitration can be either requested by the parties, or it can be forced upon them by a prior decision, such as a court order. There are three forms of arbitration: *conventional*, *final offer* and *mediation-arbitration combination*.

In conventional arbitration, the arbitrator has the option of making a decision that seems the best for all parties involved. In final offer arbitration, the arbitrator uses what the parties have currently proposed without any improvisations. Finally, mediation-arbitration combination provides the arbitrator with the greatest amount of flexibility. In this method, the arbitrator initially performs mediation to get the two sides to agree. If that fails, the arbitrator then renders

a binding decision. This form of arbitration allows the arbitrator to reason with the participants before throwing this weight around.

4.2 Computational Models

A common practice in building knowledge-based systems is to avoid potential conflict situations through analysis and consistency checking of the knowledge-base at development time [31, 55, 60, 61, 66, 70, 89]. Traditional knowledge-based systems rely on all of the potential conflict between different perspectives being resolved at *knowledge-base development time*. This is done by checking knowledge-bases for consistency and completeness. Consistency checking includes detecting conflicts, redundancies and subsumptions. Completeness checking includes testing whether the system answers all reasonable situations within its domain of expertise. This approach, although effective, is very costly as the amount and diversity of knowledge increases. Also, it may be impossible to foresee all possible conflicts which may arise in a given domain. Resolving all conflicts, no matter how unlikely, at development time can be prohibitively time-consuming. Moreover, dividing the domain knowledge into smaller internally consistent collections is difficult.

Problems encountered when resolving conflicts at development time can be avoided by allowing conflicts to occur and be resolved at *run-time*. In other words, participating agents are allowed to generate conflicting solutions to the subproblems at run-time. In the case of a conflict, a set of strategies could be used to resolve the conflict. An agent's proposed solution can either be acceptable or unacceptable to another agent depending on how the proposed solution benefits the latter agent. When proposals are unacceptable, agents are in conflict. These conflicts may involve single or multiple perspectives and must be resolved by some form of iterative negotiation if the agents are to agree on a solution. To prevent the overall performance degradation of a system, the negotiation mechanism must allow agents to quickly converge on a solution. The fundamental advantage of

run-time conflict resolution is that it constitutes a more realistic model of cooperative problem-solving than development time conflict resolution does. This is achieved both by constituting a better model of human group problem-solving, as well as by reducing the complexity of the individual agents to more manageable levels. Moreover, there are a number of advantages to allowing conflicts to be detected and resolved at run-time using explicitly represented conflict resolution strategies:

- *improved comprehensibility*: Run-time conflict resolution allows us to maintain in separate knowledge sources the different bodies of expertise as originally produced by the human domain experts. This makes it easier for a domain expert to modify a knowledge source at a later time. If all conflicts are resolved at development time, we have in effect replaced these multiple bodies of expertise by a single one that is difficult for any one domain expert to understand or modify.
- *increased extensibility*: We can add new bodies of expertise to a system without having to resolve, at development time, all the conflicts that may arise among the knowledge sources. Run-time conflict resolution thus helps insuring the independence of the bodies of expertise in a complex knowledge-based system.
- *increased flexibility*: When conflicts are resolved at run-time instead of development time, we have flexibility in the choice of which conflict resolution strategy we use. We can, in fact, use conflict resolution knowledge added to the system after the knowledge sources were originally developed.
- *involving human problem solvers*: Among the most compelling reasons for using run-time conflict resolution is the role that humans can play in cooperative systems with both human and machine-based agents. It is not practical to expect the human participants to resolve all potential conflicts before they participate in the system's operation. The use of run-time

conflict resolution strategies constitutes a better model of how cooperation among human teams takes place, and thus provides a more natural framework for systems with human and automated participants.

Some examples of conflict resolution strategies in computational models include *backtracking*, *compromise negotiation*, *integrative negotiation*, *constraint relaxation*, *case-based and utility reasoning methods*, and *multi-agent truth maintenance* [1, 41, 43, 46, 63, 64, 90, 91, 99, 102, 103, 104].

There are two generally used computational models for conflict resolution: *compromise negotiation* and *integrative negotiation*:

- In *compromise negotiation*, a solution is iteratively revised by sliding a value or set of values along some dimension until a mutually acceptable middle point is determined. In some sense, compromise acts to fine-tune a solution that is close to acceptable. Compromise negotiation has certain requirements that must be satisfied in order for it to be effective:
 - there should be a small number of dimensions involved,
 - there should be some method for evaluating whether the proposed values are moving towards each other, and
 - there should be a common scale on which agents can fine-tune their findings.
- *Integrative negotiation*, on the other hand, is useful for finding solutions for problems that are not appropriate for compromise negotiation or in situations where novel solutions are desirable. The main point of integrative negotiation is to identify the most important goals of each agent, and to find a solution which fulfills all of these goals. If the goals are incompatible, it may be necessary to decouple and abandon some of the goals.

There are several important studies that emphasize the use of conflict resolution within cooperating expert systems. Below, we describe studies that come closest to providing conflict resolution expertise with a first-class status.

PERSUADER

Sycara [90, 91, 92] presents a model of negotiation in which the system (PERSUADER) acts as a mediator in union/management labor disputes. PERSUADER attempts to find an equitable solution for a set of conflicting goals within the context of industry standards. PERSUADER's preferred reasoning method is *precedent-based reasoning*. A case memory is searched for cases which are similar to the current dispute based on a set of salient features. If one is found, the values used in that case are adjusted according to domain heuristics and presented to the disputants as a possible solution. In most cases, the proposals presented by the disputants are ignored by the mediator in developing a compromise solution. This may be responsible in domains where there is an accessible database of standard cases such as law or labor relations. It does not work in domains where problems require unique solutions or where there is no easily accessible compilation of standards.

In PERSUADER, precedent-based reasoning breaks down when no appropriate precedents exist. Preference analysis, a simplified derivative of multi-attribute decision theory, is used to generate a solution based on the disputants' utilities when this happens. Preference analysis relies on the existence of a central mediator with access to the solution evaluation criteria of all agents. This is unrealistic in cooperating experts approach since the knowledge represented is so diverse and also against to the idea of modularity and open systems semantics. A characteristic of this model is that negotiation is the main task performed by the system. The system performs conflict resolution on disputes that are provided by outside agents. We note that our own work views negotiation as an integral part of a general problem-solving process rather than as a separate task.

DFI

Werkman [99] developed a system called *Design Fabricator Interpreter* (DFI) which is a framework for distributed cooperative problem-solving among construction agents. The DFI system reflects the distributed nature of the construction industry by providing a multi-agent architecture which models design, fabrication, and erection processes. Conflicting recommendations issued by design agents are resolved by a *third-party arbitrator* agent. The arbitrator makes suggestions based on the globally known conflict resolution knowledge. It operates in both passive, and active mode. In *passive* mode, the arbitrator monitors the agent proposal process and intercedes when a problem is evident. In *active* mode, arbitrator mediates during the agent's proposal process when called upon by the agents.

CDE

Klein and Lu [43] proposed a model, called CDE (Cooperative Design Environment), for cooperative design, that emphasizes the parallel interaction of design agents. This work addresses the problem of how conflicts among different experts can be resolved, as follows: there are several design experts and a particular conflict resolution expert. Given a design problem, design experts attempt to solve the subproblems relevant to their expertise. When a conflict is detected, the conflict resolution expert takes control and tries to resolve it. This particular expert maintains the global conflict management knowledge, which contains conflict classes and corresponding resolution strategies. Within this knowledge, the more abstract classes represent domain-dependent classes and corresponding strategies, while more specific classes apply only to a particular domain, which is gathered in the phase of knowledge acquisition.

CEF

Lander and Lesser [46] proposed a framework, called CEF (Cooperating Experts Framework), to support cooperative problem-solving among sets of knowledge-based systems. The participating agents solve subproblems relevant to their specific expertise and integrate their efforts using conflict resolution strategies that are appropriate to the problem-solving context. All of the agents have a global knowledge of conflict resolution strategies. When a conflict is detected, agents involved in the conflict propose their alternative resolution strategies. Eventually they agree on a resolution scheme. Later, the conflict is resolved by one of the chosen agents based on that scheme.

NTC

Adler et al. [1] discuss methods of conflict resolution in the domain of telephone network traffic control. A homogeneous group of agents has geographically divided responsibilities with no overlap. The basic problem that the agents are to solve is excessive demand for the resources in some parts of the network. Two negotiation protocols are described:

- *conflict-driven plan merging*, a bottom up approach to resolving a conflict that has already occurred, and
- *shared plan development*, a top down approach to avoiding conflicts as plans are developed and refined.

Their research addresses how conflicts on the usage of resources could be resolved. The strategies range from a priori protocols for avoiding conflict situations to arbitration of conflicts.

Chapter 5

The Computational Model

The cooperating problem-solving environment, *NEPTUNE*, is organized as a community of cooperating problem-solving agents, where each agent is represented as a fully functional and autonomous knowledge-based system. *NEPTUNE* is designed for solving problems particularly in the domain of design. Most complex design problems are based on the insights that each design agent has its own conflict resolution knowledge separate from its domain-level design knowledge. Such knowledge can be instantiated in the context of particular conflicts into specific advice for resolving these conflicts. *NEPTUNE* allows a new problem solver to be added, or an existing one to be removed, without requiring any modification to the rest of the system. *NEPTUNE*, therefore, can be considered as achieving *open systems semantics*¹ [37, 38, 39] in the sense that it not only allows scalability (the ability to increase the scale of commitment) but also robustness (the ability to keep commitments in the face of conflicts), which are two primary indicators in open systems semantics.

¹**Open systems** deal with large quantities of diverse information and exploit massive parallelism.

5.1 Why Cooperative Design

Design is the process of constructing an artifact description that satisfies certain requirements. It is based on the interaction of multiple diverse expertise. In traditional approaches, it is accomplished by a group of experts asserting and evaluating design decisions in a sequential and iterative manner [29, 32, 65, 85]. Several iterations may be required before a design that satisfies all sources of expertise is produced. This is a very time-consuming process and may sometimes lead to poor design outcomes.

The model that we propose supports a parallel rather than a sequential interaction among the design experts. There are several characteristics which make the design process particularly suitable to cooperating experts approach:

- resources and knowledge of a single individual may not overcome the cost, scale, and complexity of many design problems,
- design problems can be characterized as routine, creative, or innovative, with each type of problem requiring a different design methodology, and
- design commitments and critiques could be asserted in parallel by several knowledge sources.

The process of design has been studied extensively, thereby providing a basis for the implementation of knowledge-based design systems [2, 3, 4, 29, 30, 65, 86, 95, 96]. With the exception of a few, none of the existing knowledge-based design systems support parallel interaction among design experts. To achieve parallel interaction among design agents, there is a need for an environment in which design agents communicate with each other in solving a particular design problem.

5.2 Architecture of the Model

As shown in Fig. 5.1, *NEPTUNE* is composed of a set of design agents which are fully functional knowledge-based systems and a shared medium [67, 68, 69].

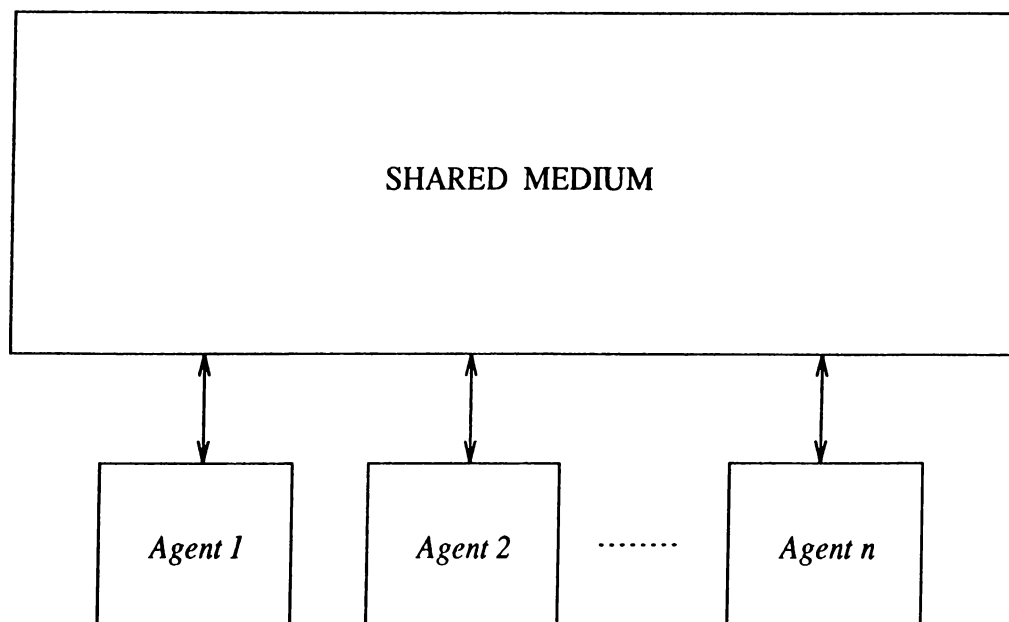


Figure 5.1: The Architecture of the Proposed Cooperative Design Environment

The agents communicate by posting their assertions on the shared medium. Assertions are expressed in a common language. This requires that the agents have translation capabilities from and into this common language.

NEPTUNE is implemented on a network of workstations running under the UNIX operating system. Each agent is modeled as a process running on a workstation which is an autonomous knowledge-based system that makes an offer to solve subproblems within its interest area, may create subproblems to get help from others, and cooperate with others to resolve conflicts to be encountered.

5.2.1 Representation of Problem and Knowledge

In this section, we formally describe the representation of elements used in defining problem and knowledge in our computational model.

Definition (objects)

$\mathcal{O} = \{o_1, o_2, \dots, o_{N_{obj}}\}$ represents the set of objects that contain information to be used by the agents in their design processes. Each object represents a separate element in the universe of objects.

For each object $o_i \in \mathcal{O}, 1 \leq i \leq N_{obj}$, $Attributes(o_i) = \{att_{i_1}, att_{i_2}, \dots, att_{i_{M_i}}\}$ denotes a set of data attributes that include information in the form of either numerical/symbolic constants, or procedures (methods) that yield numerical/symbolic values and constitute the derived attributes from basic data. An attribute of an object may also point to a non-atomic structure, hence forming a nested complex structure. Each object has two generic attributes other than those in $Attributes(o_i)$. They are `object-id` and `domain-name`. While `object-id` is a symbolic constant which uniquely identifies an object among others, `domain-name` refers to the domain or class of the object which is a collection of similar object instances.

In many systems, knowledge about a domain usually centers around the description of objects and their component pieces. The constituents of this knowledge in our system are a database of design elements and their component pieces. The following is an example of a floor beam object taken from a framing system.

Object

```

object-id      : symbolic
domain-name   : Floor-Beam
  beam-id      : symbolic
  beam-width   : numeric
  beam-height  : numeric
  beam-depth   : numeric
  beam-section : symbolic
  begin-reaction : numeric
  end-reaction  : numeric
  uniform-load : procedure calculate-beam-load ...

```

EndObject

Definition (relations)

$Rel = \{rel_1, rel_2, \dots, rel_{N_{rel}}\}$ denotes the set of relations where a relation has a name and arity. Each relation is used to establish a relationship between a set

of objects involved in the problem to be solved. Each $rel_i \in \mathcal{R}el, 1 \leq i \leq N_{rel}$ denotes hierarchical or logical relationship among certain types of objects. A solution to a specific problem at any level of granularity is represented as a set of relationships which is a subset of $\mathcal{R}el^*$.

Below is a sequence of relationships that relates the objects `motor`, `pump`, `vbelt`, and `elastic-platform` which are some typical components to be used in the design of a steam condenser:

Relationships

```
connected(motor, vbelt)
connected(pump, vbelt)
on(motor, elastic-platform)
on(pump, elastic-platform)
on(vbelt, elastic-platform)
```

In this example, $\{\text{connected}/2, \text{on}/2\} \subset \mathcal{R}el$ and `motor`, and `pump` objects are connected through `vbelt` objects, all of them lying on `elastic-platform` object. The functionality of the `motor` is to run the `pump` by delivering sufficient power through `vbelt`. Positioning of objects `motor`, `pump` and `vbelt` on `elastic-platform` is represented with other relationships.

Definition (domains)

$\mathcal{D} = \{D_1, D_2, \dots, D_{N_{dom}}\}$ denotes the set of domains. Each object o_i in \mathcal{O} is taken from some domain D_j , i.e., $(\forall o_i \in \mathcal{O}) (\exists D_j \in \mathcal{D}) [o_i \in D_j]$ where $1 \leq i \leq N_{obj}$, and $1 \leq j \leq N_{dom}$. In the same way, each attribute att_{i_n} of object o_i is an element taken from some domain D_k , that is $(\forall att_{i_n} \in \text{Attributes}(o_i)) (\exists D_k \in \mathcal{D}) [att_{i_n} \in D_k]$ where $1 \leq k \leq N_{dom}$ and $1 \leq n \leq M_i$.

In addition to the objects and their attributes, there is a set of parameters which are used in the design problem-solving. These parameters take values from domains in \mathcal{D} . These parameters stand for the domain values (constants, or object instances) to be computed and presented as solutions.

We define a function $\text{domaintype}(D_i)$ which denotes type of domain $D_i \in \mathcal{D}$

$$\text{domaintype} : \mathcal{D} \rightarrow \{\text{numeric}, \text{symbolic}, \text{complex}\}.$$

All domains in \mathcal{D} are bounded domains. A numeric domain is represented as $[l, u]$ where l is the lower bound and u is the upper bound in the domain. A symbolic or complex domain is represented as an ordered set of values (symbolic constants or object instances). For $D_i \in \mathcal{D}$ and $domaintype(D_i) = complex$ or $symbolic$, $lower(D_i)$ denotes the first element in D_i and $upper(D_i)$ denotes the last element in D_i .

Note that the union of domains in \mathcal{D} forms the universe of discourse, called *Universe*. That is, $Universe = \bigcup_{D_i \in \mathcal{D}} D_i$.

Definition (parameters)

$\mathcal{P} = \{p_1, p_2, \dots, p_{N_{par}}\}$ is the set of parameters for which the agents will be collaboratively seeking values from domains in \mathcal{D} . Parameters can be grouped into two categories according to their underlying domains:

- atomic parameters, and
- complex parameters (non-atomic parameters).

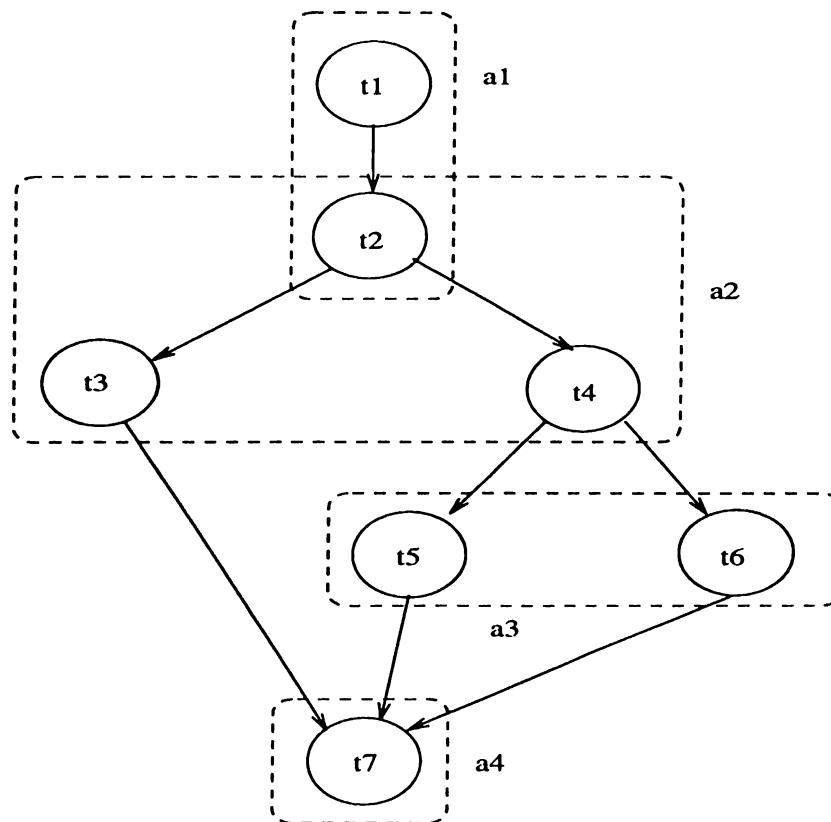
$P^A \subseteq \mathcal{P}$ represents the set of atomic parameters. Atomic parameters denote variables that take primitive constant values which can be either numeric constants (integer or real) or symbolic (non-numeric) constants. $P^C \subseteq \mathcal{P}$ represents the set of complex parameters. Complex parameters denote objects which are structured composite data types. We define a function $domain(p_i)$ which returns domain of parameter p_i .

$$domain : \mathcal{P} \rightarrow \mathcal{D}, (\forall p_i \in \mathcal{P}) (\exists D_j \in \mathcal{D}) [domain(p_i) = D_j].$$

Note that P^A and P^C are disjoint, i.e., $P^A \cup P^C = \mathcal{P}$ and $P^A \cap P^C = \emptyset$.

Definition (tasks)

$\mathcal{T} = \{t_1, t_2, \dots, t_{N_t}\}$ represents the set of tasks to be performed. A task represents simply a goal, at any level of granularity, that must be satisfied by at least one of the agents in the problem-solving network. There exists a partial order \mathcal{Po} over the tasks in \mathcal{T} that the agents should follow, defined as



Tasks: t1, t2, t3, t4, t5, t6, t7

Agents: a1, a2, a3, a4

----- agents' areas of interests

—————> sequence of tasks

Figure 5.2: Example of a task sequence to be performed by a set of agents.

$\mathcal{Po} = \{po(t_i, t_j) | \exists t_i, t_j \in \mathcal{T} \text{ and } t_j \text{ is immediate successor of } t_i\}$ denotes the set of intertask dependencies. Figure 5.2 shows an illustrative set of tasks to be attempted by a set of agents. In this example, circles represent tasks, arrows represent flow of control and intertask dependencies while dashed areas illustrate agents' areas of interest. Agents that have overlapping interest areas may produce conflicting solutions since they have common interface parameters. In the figure, agents $a1$ and $a2$ may produce conflicting proposals through task $t2$ they are sharing. In the meanwhile, agents may indirectly cause conflicting situations even though they do not have overlapping interest areas. For example, agent $a2$ may recommend values for some parameters in task $t2$ that indirectly restrict parameters in task $t5$ which is within the interest area of agent $a3$.

In the design problem-solving, each task identifies the design parameter to be instantiated with an appropriate value acceptable by all of the agents in the problem-solving network. This instantiation can be viewed in two different perspectives:

- selecting an appropriate object instance from a domain, or
- finding a numeric, or symbolic value which will not cause any dissatisfaction.

In general design process is terminated when all of the tasks are finished which results in a set of relationships to be established, all design parameters are instantiated to appropriate values and all design agents agree on the final design outcome. In *NEPTUNE*, agents may cancel some of the tasks which may lead to poor design outcomes upon negotiating over different issues.

Definition (constraints)

In our multidisciplinary design process, problem-solving agents generate partial local solutions by assigning values to parameters and by exchanging assigned values in a particular strategy to reach a globally consistent satisfiable solution. In assigning values to parameters, agents typically satisfy their requirements and design procedures which are considered to be design constraints. Each agent has

its own constraint, in addition to the global design constraints which are imposed by the agent that initiated the problem.

$\mathcal{C} = C(\alpha_1)UC(\alpha_2)U\dots UC(\alpha_{N_a})UC_G$ denotes the set of constraints in the design where $C(\alpha_i)$ is the set of constraints specific to agent α_i , C_G is the set of global design constraints, N_a is number of agents in the problem-solving network and $1 \leq i \leq N_a$. Each agent α_i is only aware of the constraints $C(\alpha_i)UC_G$. Given $P_k \subseteq \mathcal{P}$, $c_k(P_k)$ represents a constraint which restricts the values that may be assumed by the parameters in P_k which is a subset of \mathcal{P} . Parameters of a constraint are classified into two categories: input parameters and output parameters. Input parameters are the only ones that may affect output parameters. Moreover, each constraint has a set of methods assigned.

Definition (actions)

$Act = \{a_1, a_2, \dots, a_{N_{act}}\}$ denotes the set of allowable actions (or moves) that the agents execute in achieving their assigned tasks. Each $a_i \in Act, 1 \leq i \leq N_{act}$, represents an action which involves a set of parameters.

Actions can be defined differently in different domains. An action aims at establishing a relationship among a set of objects. In the engineering design domain, an action can be finding values for a set of design parameters (selecting appropriate object instances to be instantiated to a complex variable, or finding a constant value to be assigned to an atomic parameter), and/or adding/deleting/modifying a relationship that involves certain types of objects.

5.2.2 Shared Medium

The shared medium is a public repository available to all agents. This permits the storage of “global” information, although the information can only be used locally by the agents. Alternatively, it would be possible to convey information directly through point-to-point communication channels or through reserved-spot communication [101]. The shared medium is partitioned into four sections, allowing fast access, delete and update operations of units (Fig. 5.3). These sections are called *problem*, *solution*, *proposal* and *conflict areas*.

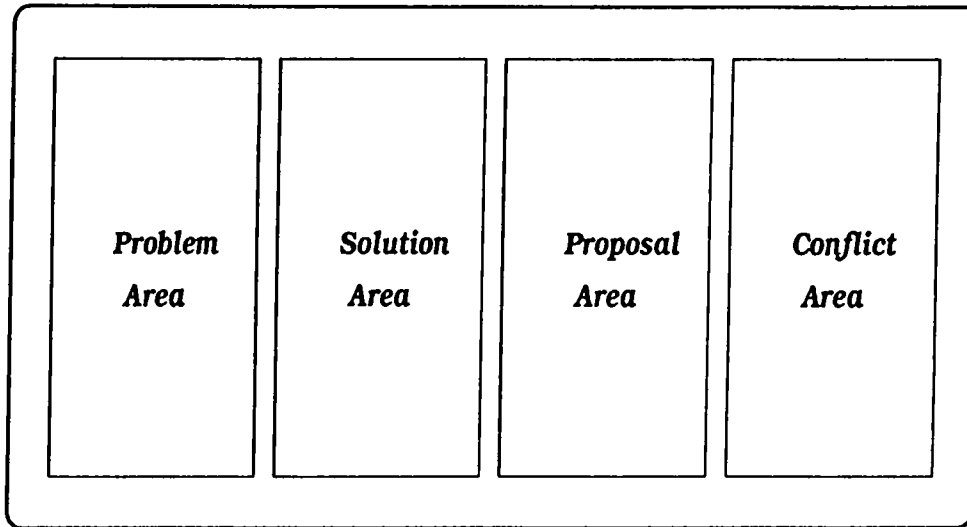


Figure 5.3: The Shared Blackboard

The problem area of the shared medium contains the initial problem definition and overall requirements that must be taken into account by the design agents. A problem definition can be asserted by any of the agents that exist in the problem-solving environment. In addition to other agents, the owner of the problem also attempts to solve the subproblems within its area of interest. A problem instance is a tuple of the form

ProblemInstance = $\langle \alpha_o, \mathcal{T}, \mathcal{P}_o, C_G, I \rangle$ where

α_o denotes the problem originator, an agent that defines the problem to be solved,

\mathcal{T} is the set of tasks that must be satisfied for a design to be accepted,

\mathcal{P}_o is the set of intertask dependencies,

C_G is the set of constraints that design agents should not violate²,

I denotes the initial problem information (such as the layout of a room if the problem is to design an office).

²Some of the constraints may be violated through negotiation with the problem originator.

The solution area of the shared medium maintains the evolving design template \mathcal{R} , to which non-conflicting design commitments produced by the agents are added. The evolving design template is composed of a set of relationships that represents current state of design at any phase of problem-solving. \mathcal{R} is updated with new relationships introduced by a proposal when agents cooperatively agree on the solution proposal. Note that $\mathcal{R} \subseteq \mathcal{Rel}^*$.

The proposal area includes partial and incomplete solutions produced at several layers of abstraction by design agents. Design agents insert their solutions as proposals into this area. A proposal instance is a tuple of the form

$$Q = \langle q_j, \alpha_i, T_{ij}, Act_{ij}, R_{ij} \rangle \text{ where}$$

q_j is the identity of the proposal,

α_i denotes the owner of the proposal,

T_{ij} denotes the set of tasks for which the proposal has been generated,

$Act_{ij} \subseteq Act^*$ is an ordered set of proposed actions to update the current design template,

R_{ij} consists of the relationships to be established upon reflecting changes offered by the actions in Act_{ij} .

The conflict area is the place where agents put their objections and critiques related to a new design commitment. A portion of this area provides a communication medium with agents that are involved in a conflict situation. This area holds evaluation results and conflict resolution recommendations issued by design agents. An evaluation result instance is a tuple of the form

$$ER = \langle q_j, \alpha_i, Ra_{ij}, Re_{ij}, Rd_{ij} \rangle \text{ where}$$

q_j is the identity of the proposal evaluated.

α_i denotes the owner of the evaluation result tuple,

Ra_{ij} is the set of ratings (evaluation results) for each action in Act_{kj} within proposal q_j ,

Re_{ij} is degree of satisfaction or dissatisfaction caused by proposal q_j
 Rd_{ij} indicates whether the proposal q_j is acceptable or not. $Rd_{ij} \in$
 $\{\text{conflicting-proposal, nonconflicting-proposal}\}$

A conflict resolution instance is a tuple of the form

$CR = \langle q_j, \alpha_i, Res_{ij}, Rref_{ij} \rangle$ where

q_j is the identity of the related proposal,

α_i is the owner of the conflict resolution tuple,

Res_{ij} indicates whether agent α_i is to be constraining the search space, or is to be counter-proposing a new alternative, and $Res_{ij} \in \{\text{constraining, counter-proposing}\}$ and

$Rref_{ij}$ includes the set of constraints if $Res_{ij} = \text{constraining}$, the set of tasks $T_l \subseteq T$ for which the agent α_i will be counter-proposing an alternative proposal. The original task set for proposal q_j is a subset of T_l .

In the resolution phase, agents that have overlapping interest areas concerning the task set under consideration try to generate alternative proposals. Other agents attempt to restrict search spaces of the agents that have potential to counter-propose.

5.2.3 Agent Model

In this section, we describe a general model of an agent which constitutes the base for the computational model developed.

Definition (agents)

$\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_{N_a}\}$ represents the set of agents that will participate in the problem-solving process. As shown in Fig. 5.4, an agent α_i ($1 \leq i \leq N_a$) contains:

- a *database* that includes

- $T(\alpha_i)$ which denotes the set of tasks that agent α_i can perform ($T(\alpha_i) \subseteq \mathcal{T}$).
- $P(\alpha_i) = P^A(\alpha_i) \cup P^C(\alpha_i)$ denotes the set of parameters occurring in task descriptions $T(\alpha_i)$ for agent α_i , where $P(\alpha_i) \subseteq \mathcal{P}$, $P^A(\alpha_i) \subseteq P^A$, $P^C(\alpha_i) \subseteq P^C$.
- $D(\alpha_i) = \{D'_m | (\exists p_n \in P(\alpha_i))(\exists D_m \in \mathcal{D})[domain(p_n) = D_m \text{ and } D'_m \subseteq D_m]\}$ denotes the set of domains from which agent α_i picks up values for its parameters whose values are unknown, or not known precisely. Note that each domain in $D(\alpha_i)$ is a subset of a domain in \mathcal{D} .

- a *knowledge-base* that includes
 - *domain knowledge*,
 - *control knowledge* and
 - *conflict management knowledge*, and
- *control procedures which are*
 - proposal generation,
 - proposal evaluation,
 - conflict detection and resolution, and
 - communication.

As problem-solving process starts, each agent examines the global task set, \mathcal{T} , and selects those tasks that it can perform. An agent checks the objects and the relationship to be established among the objects that have been introduced in a particular task definition. The agent may later decompose the task into several subtasks (goals) according to its problem-solving capabilities.

The knowledge-base includes domain and control knowledge, just like in a classic knowledge-based system. In addition, it also contains conflict management knowledge that can be used in cooperatively managing conflicts with other agents. This knowledge is not available to other agents, and it varies with respect to

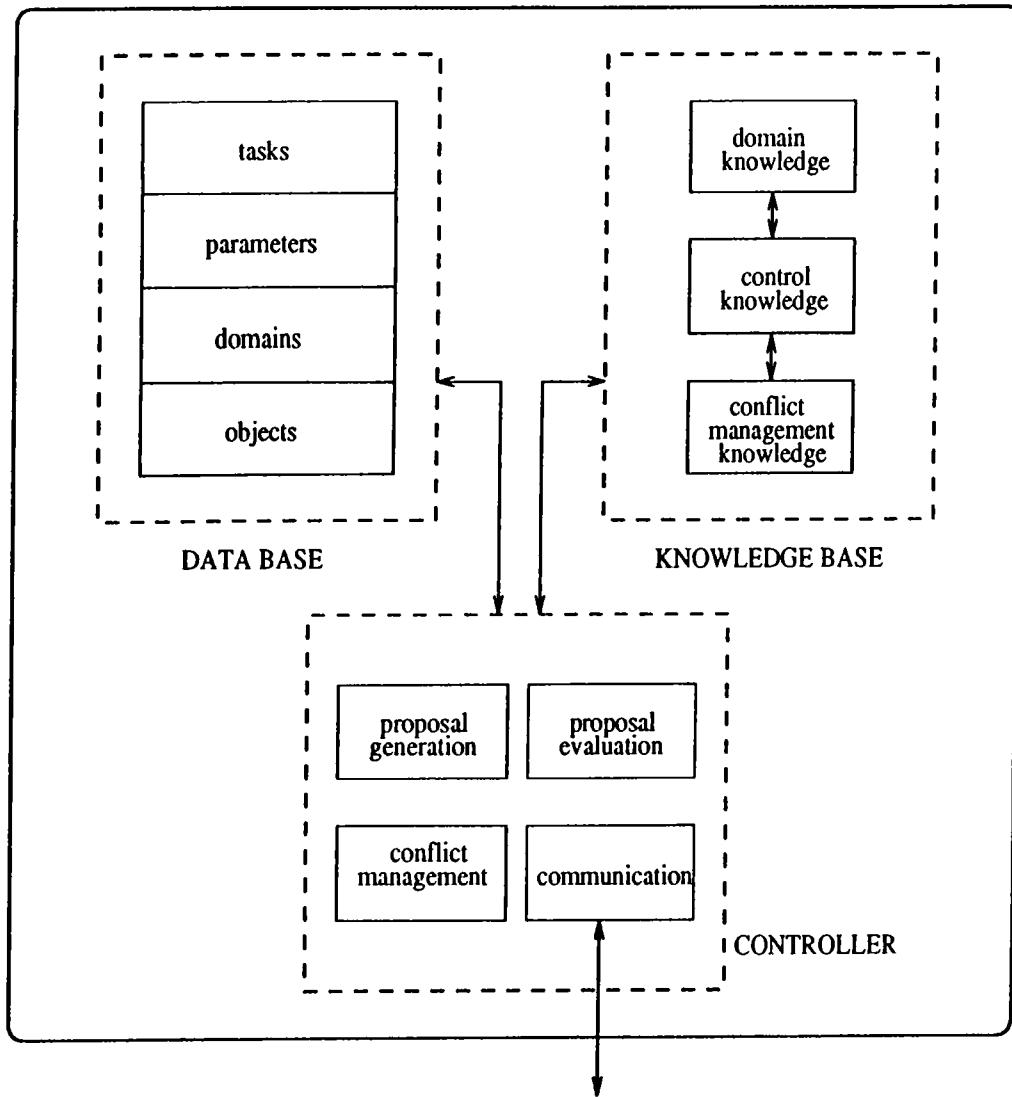


Figure 5.4: Internal Structure of a Design Agent in the Model

the agents' beliefs and understanding of the environment. The general controller includes procedures for generating and evaluating design commitments, managing conflicts, and communicating with other agents.

The agents are actually heterogeneous in the sense that they might use different knowledge representation techniques and inference mechanisms. Agents are assumed to generate proposals (solutions for subproblems) according to their knowledge. They cooperate to achieve the common goal of solving a global problem. Local knowledge is represented in whatever language desired but cannot be accessed by any agent except its owner. Several knowledge representation techniques have been developed for the domain of design [5, 29, 32, 85, 86, 88, 94, 95, 96]. If the internal language is not the same as the shared language, translation procedures should be incorporated within the agents. In the case of a conflict, agents might make some or all of their goals, constraints and even knowledge available to others.

5.2.4 Conflict Resolution Knowledge

Conflict resolution knowledge of agents centers around the domain issues through which each agent is to be contributing to the global solution. Each such issue is identified during the knowledge engineering process.

Agents' Issues and Preferences

Definition (issues)

$Issues(\alpha_i) = \{\beta_j | \beta_j \text{ is a domain-dependent issue}\}$ denotes the set of issues over which agent α_i is to be evaluating proposed partial solutions and negotiating with others in generating agreeable solutions. β_j in $Issues(\alpha_i)$ represents a domain-dependent issue of agent α_i . $P(\alpha_i, \beta_j) \subseteq P(\alpha_i)$ denotes the set of parameters which are involved in issue β_j of agent α_i . Some parameters involved in one issue may be within the scope of other issue. This may happen in two ways:

- $(\exists \alpha_i \in \mathcal{A}) (\exists p_m \in P(\alpha_i, \beta_k)) [p_m \in P(\alpha_i, \beta_l)]$ where $l \neq k$. That is, different issues within same agent can share parameters.

- $(\exists \alpha_i, \alpha_j \in \mathcal{A}) (\exists p_m \in P(\alpha_i, \beta_k)) [p_m \in P(\alpha_j, \beta_l)]$ where $i \neq j$ and $l \neq k$.
That is, different issues in different agents can share parameters.

As an example of an issue and its parameters, consider the electricity agent which is concerned with electrical devices to be installed in an office. One of the issues of this agent is *configurability* which involve location parameters of the electrical devices to be placed in the office.

Each issue is assigned a set of constraints which restricts values of parameters. $C(\alpha_i, \beta_j)$ denotes the set of constraints which restrict some parameters within the scope of issue β_j of agent α_i . Note that

$$\forall c_m (P_m) \in C(\alpha_i, \beta_j) [P_m \cap P(\alpha_i, \beta_j) \neq \emptyset].$$

A proposal generated by an agent which consists of a series of actions may affect another agent in its way to assign values to its parameters within its domain of expertise. In this sense, we talk about *satisfaction* of an agent from a proposal. This requires a fair evaluation of the proposal and understanding its total effects from all related issues for each agent. Agents declare preferences along with their issues and quantify their degree of satisfaction to be caused by any possible proposal based on their issues.

Definition (issue preferences)

Each agent defines a set of preference values for all of its issues. $\Sigma_{\alpha_i} = \{\sigma_{\alpha_i, \beta_j} | 0 \leq \sigma_{\alpha_i, \beta_j} \in \mathfrak{R} \leq 1\}$ denotes the set of preference values for all issues of agent α_i . $\sigma_{\alpha_i, \beta_j}$ reflects the degree of importance that agent α_i gives to its issue β_j . The issue preference values must sum up to 1, i.e.,

$$\sum_{\beta_j \in Issues(\alpha_i)} \sigma_{\alpha_i, \beta_j} = 1.$$

Definition (parameter preferences)

In addition to issue preferences, each agent defines a set of preference values for parameters within its domain of expertise. $\Lambda_{\alpha_i, \beta_j} = \{\lambda_{\alpha_i, \beta_j, p_k} | 0 \leq \lambda_{\alpha_i, \beta_j, p_k} \in \mathfrak{R} \leq 1\}$ denotes the set of preference values that agent α_i assigns to its parameters within

its issue β_j . $\lambda_{\alpha_i, \beta_j, p_k}$ reflects the degree of importance that agent α_i gives to its parameter p_k in its issue β_j . These preference values must add up to 1, i.e.,

$$\sum_{p_k \in P(\alpha_i, \beta_j)} \lambda_{\alpha_i, \beta_j, p_k} = 1.$$

By this way, agents rank their parameters within their issues.

Agents' Satisfiability

Satisfaction or dissatisfaction of an agent caused by a proposal is quantified according to the partial effects caused by the actions within the proposal. Each such partial effect can be seen as an assignment of a new value to a parameter from a domain. The aim is to decide whether this change has occurred in a desired direction or not.

Definition (desired changes for parameters)

$\Psi_{\alpha_i} = \{\psi_{\alpha_i, p_j} | p_j \in P(\alpha_i) \text{ and } \psi_{\alpha_i, p_j} \in \{-1, 1\}\}$ denotes the set of desired directions for all parameters of agent α_i . ψ_{α_i, p_j} takes the value of -1 or 1 depending on whether a decrease or increase in the value of p_j is desired by agent α_i . In other words, $\psi_{\alpha_i, p_j} = 1$ means that an increase in the value of p_j is desired by agent α_i .

In case of computing a numeric value to be assigned to a parameter, an agent tries to minimize or maximize the values to be picked up. For example, consider a proposal which changes the value of parameter **power** in the design of a steam condenser from 85 to 90 whose domain ranges from 50 to 120, i.e., $[50, 120]$. **Motor** and **pump** agents are directly concerned with the parameter **power**. The **motor** agent, from its **efficiency** issue, tries to minimize the **power** whereas the **pump** agent, from its **functionality** issue, tries to maximize the **power** in order to run properly. According to the proposal the **motor** agent is negatively affected, while the **pump** agent is satisfied.

In non-numeric domains, agents keep values in their domains in an ordered manner which reflects agents' preferences. For each of its non-numeric domains,

an agent defines a preference set whose elements denote agent's degree of preference given to the corresponding non-numeric value in the relevant domain. If a parameter is shared by different issues within an agent, the agent defines separate preference sets for each occurrence of that parameter.

$\Pi_{\alpha_i, \beta_j, D'_k} = \{\pi_{\alpha_i, \beta_j, D'_k, v_l} | 0 \leq \pi_{\alpha_i, \beta_j, D'_k, v_l} \in \mathfrak{R} \leq 1\}$ denotes the set of preference values for the values in domain $D'_j \in D(\alpha_i)$ of agent α_i from issue β_j . Note that $\Pi_{\alpha_i, \beta_j, D'_k}$ may be undefined if none of the parameters within issue β_j of agent α_i takes values from domain D'_k .

$$\Pi_{\alpha_i, \beta_j, D'_k} = \begin{cases} \bigcup_{l=1}^{card(D'_k)} \{\pi_{\alpha_i, \beta_j, D'_k, v_l}\} & (\exists p \in P(\alpha_i, \beta_k)) (\exists D'_k \in D(\alpha_i)) [D'_k \subseteq domain(p)] \\ \emptyset & \text{otherwise} \end{cases}$$

$\pi_{\alpha_i, \beta_j, D'_k, v_l}$ is a real number between 0 and 1 which represents preference value defined by the agent α_i corresponding to the l^{th} value of the domain D'_k from issue β_j . A preference value of 1 indicates that the value to be picked up is desired most, whereas that of 0 indicates that the value to be picked up is not desired at all. The values in $\Pi_{\alpha_i, \beta_j, D'_k}$ are arranged in non-decreasing order, i.e., $\pi_{\alpha_i, \beta_j, D'_k, v_x} \leq \pi_{\alpha_i, \beta_j, D'_k, v_y}$ if $x \leq y$.

5.3 Problem-Solving Phases

Problem solving in the system is initiated by one of the agents asserting a problem definition into the problem area of the shared medium. Fig. 5.5 outlines the basic steps in problem-solving phases. All the agents have the right to access the shared medium. When a problem definition is asserted into the shared medium, each agent examines the task set, \mathcal{T} , to determine which of the tasks it can perform. An agent α_i adds a task into its task set $T(\alpha_i)$ if it has the knowledge to select particular object instances from its database and to establish the necessary relationships among the objects. When this process is completed by all of the agents, each agent knows the tasks it is going to perform. Some of the tasks might have been attempted by more than one agent, since the knowledge and problem-solving capabilities of different agents might overlap. However, one agent might

have *deep* knowledge, whereas another might have *shallow* knowledge. An agent has a set of domains which constitute the agent's problem-solving capabilities. This means that an agent can perform a task if parameters in the task definition take values from the domains of agent's interest. An agent assigns a confidence factor to each of its domains to illustrate the degree of confidence in offering values from a domain. This confidence factor indicates whether an agent has deep or shallow knowledge regarding with a particular domain. In the negotiation phase, the proposal issued by the agent that has deep knowledge has more credibility than the proposals issued by other agents.

Each agent is also aware of the partial order among the tasks. Therefore, an agent can attempt the next task in its task queue if and only if plans for all of its preceding tasks have been generated and agreed on. All interested agents, after examining the problem definition, are instantiated and then they start producing design proposals related to their expertise, knowledge, and viewpoints. When a design proposal is generated, it is put into the proposal area.

After a proposal, q has been generated, all of the agents are signalled. An agent does not interrupt its proposal generation process if it is currently working on another proposal, but it immediately awakens another process, called *the evaluation process*, that will run in parallel with the proposal generation process. The evaluation process first informs the owner of the proposal whether it is going to criticize the proposal. If not, the evaluation process will go to sleep and wait for another proposal to be asserted.

If the agent is interested in the proposal, it evaluates the proposal and posts the result in the conflict area of the shared medium. The owner of a proposal also evaluates its proposal, usually as a part of the solution generation process. It is necessary for the owner to indicate its confidence in its own solution, because it might have used incomplete or inaccurate knowledge in producing that solution. The evaluation process results in a rating being produced which shows the "quality" of a solution with respect to the goal criteria. However, the agents use their internal evaluation criteria, and therefore may not share a common rating scale for their findings.

```

Algorithm Problem-Solving
begin
  if  $\alpha_i$  is the problem originator then
    insert the problem definition tuple,
       $ProblemInstance = \langle \alpha_i, T, \mathcal{P}o, C_G, I \rangle$  into the shared medium
  else
    wait until  $ProblemInstance$  tuple is asserted by
      the problem originating agent.
   $T(\alpha_i) = \emptyset$ 
   $Po(\alpha_i) = \emptyset$ 
  for  $j = 1$  to  $N_t$  (for each  $t_j \in T$ )
    if domains of all parameters in  $Parameters(t_j)$  is in the interest
      area of  $\alpha_i$ , i.e.,  $domains\_of(Parameters(t_j)) \subseteq D(\alpha_i)$  then
      begin
         $T(\alpha_i) = T(\alpha_i) \cup \{t_j\}$ 
        for each  $po(t_x, t_y) \in \mathcal{P}o$ 
          if  $(t_x = t_j)$  or  $(t_y = t_j)$  then
             $Po(\alpha_i) = Po(\alpha_i) \cup \{po(t_x, t_y)\}$ 
        end
      if  $T(\alpha_i) = \emptyset$  then quit
      wait until all agents identify their own task sets
      repeat
        take a task set  $T$  from  $T(\alpha_i)$  which is to be attempted
          next according to partial order in  $Po(\alpha_i)$ 
        for each  $t \in before(T)$ 
          if a plan for  $t$  has not been generated and agreed on then
            block until such a plan is generated and agreed on
          generate a proposal  $q$  for the task set  $T$ 
          assert the proposal  $q$  into the shared medium
          perform  $Evaluate-Proposal(q)$ 
          wait until all agents finish evaluating  $q$ 
          if a conflict is detected by any agent then
            perform  $Conflict-Resolution(q)$ 
           $T(\alpha_i) = T(\alpha_i) - T$ 
        until  $T(\alpha_i) = \emptyset$ 
      end Problem-Solving

```

Figure 5.5: Problem Solving Steps within Agent α_i .

After all the interested parties have finished evaluating the newly asserted proposal, those agents that have identified the proposal under consideration as conflicting with their beliefs come together to resolve the conflict (those interested agents that put evaluation result tuples in which the overall result is indicated as *conflicting-proposal*). Agents in *NEPTUNE* do not have a global knowledge of conflict resolution. However, in other systems, agents are assumed to be knowledgeable about the global conflict resolution knowledge. In *NEPTUNE*, each agent has its own conflict resolution knowledge that allows it to participate in the process of conflict resolution. The result of conflict resolution is either revision or abandonment of the proposed solution.

When none of the interested agents detects any conflict related to a proposal, the partial design template residing in the solution area is updated by using the design contribution that exists in the proposal. This process continues until the design template meets the requirements specified by the agent that put the initial problem definition into the problem area of the shared medium. The design process may also be terminated, although the agent that put in the problem definition will not be satisfied. This may happen in cases where none of the agents can generate a nonconflicting design proposal.

Chapter 6

Multi-Agent Conflict Management

The agents may be working on different problems which may result in incidental overlaps in the solution space. Also, they may be working on the same problems and have different criteria for generating and evaluating solutions. An agent, α_i , maintains an issue set, $Issues(\alpha_i)$, which includes domain issues used to evaluate partial plans proposed as solutions to tasks. A set of Evaluation procedures is attached to each issue in $Issues(\alpha_i)$. These procedures are used for detecting potential conflicts in the proposed solution from the agent's perspective on this issue.

Upon detecting a conflict situation in a proposal, the agent uses its conflict resolution knowledge to overcome the conflict from its perspective. When a conflict is detected, all agents involved in it participate in the resolution process based on their own conflict resolution knowledge. Each agent may utilize different conflict resolution strategies. For example, suppose that a team of agents are given the problem of designing an office. Two members of the team are the functionality and computer agents. One of the tasks is to identify the location of the PC desk. The functionality agent suggests that the PC desk should be put close to the window, so that a PC user could have a look outside when (s)he is bored and use the daylight. On the other hand, the computer specialist, detecting a

conflict, argues that daylight could damage the PC. The computer specialist uses a conflict-resolution strategy which says, “put electrical devices far away from windows.” The functionality agent, however, uses a domain-independent resolution strategy, the “try other subgoal alternatives.” Eventually the two agents revise the proposal, by using different resolution strategies, such that the PC desk is put into a place in the office which is not exposed to daylight. In deciding which strategy to apply, an agent uses the following piece of information:

- Critiques made by the interested parties to the proposal (after examining the outcomes of evaluation procedures of other agents, an agent chooses an appropriate resolution strategy taking into account different viewpoints).
- The relevance of the agent to particular problem being solved (if an agent is more knowledgeable and capable compared to others, it should participate in resolution of a conflict according to its relevance).
- Flexibility or insistence of agents involved in conflicts (this is important for an agent to decide how to behave in a compromise type of conflict resolution).
- Behavior and actions of other agents in resolving the conflict (by examining this information, an agent might decide to alter the conflict resolution strategy it has been using).
- Number of agents involved in the conflict (depending on the domain, if the number of agents involved in a conflict situation exceeds a certain amount, some of the agents thinking that they could not be effective for resolving the conflict compared to others, may continue to generate alternative solutions rather than participating in conflict resolution).
- Available problem-solving resources (agents may have different problem-solving capabilities).

Note that this criteria to choose the appropriate strategy is not just for design problem-solving, rather they can be applied to many other types of problems.

6.1 Multi-Agent Conflict Detection

Agents evaluate a proposal (partial solution for a certain task) in order to detect the degree of effects to be caused by actions within the proposal from different issues. An agent examines all actions in a proposal one by one. It identifies which of the parameters within its area of interest will be affected by the action under consideration. An affected parameter is identified in one of the following two ways:

- *directly*: the action explicitly addresses a change in the value of the parameter within the agent's area of interest,
- *indirectly*: the action addresses a change in the value of another parameter which restricts the parameter in consideration.

The aim of an agent in the problem-solving network is to decrease its dissatisfaction to be caused by the actions in a proposal. In order to understand the degree of effect caused by an action changing the value of a parameter, an agent tries to detect whether the offered change occurs in a desired direction, or not.

6.1.1 Computation of Degree of Satisfaction

Each agent defines a method for each of its parameters, to compute the degree of satisfaction to be caused by an action. The definition of such methods depends upon the agents' preferences. Each method aims at normalizing the change offered for a parameter with respect to its base domain. It returns a real value between -1 and 1 where a positive value denotes the degree of satisfaction whereas a negative value denotes the degree of dissatisfaction.

Suppose that the value of a parameter is changed in two different ways but in the same direction (either increase or decrease); first from x_1 to x'_1 , second from x_2 to x'_2 on where $x'_1 - x_1 = x'_2 - x_2$. Although the amount of change is the same, an agent may be affected differently depending on where these changes occur in a base domain. In describing the method for computing the degree of effect, we have to consider the following cases:

- *case I*: the same amount of change may affect an agent in the same way no matter where this change occur in the base domain.
- *case II*: an agent may prefer the same amount of change occurring towards the lower bound of the base domain to the one occurring towards the upper bound of the base domain. In other words, if $x_1 \leq x_2$ and $x'_1 \leq x'_2$ then the change from x_1 to x'_1 is preferred to the change from x_2 to x'_2 .
- *case III*: an agent may prefer the same amount of change occurring towards the upper bound of the base domain to the one occurring towards the lower bound of the base domain. In other words, if $x_1 \leq x_2$ and $x'_1 \leq x'_2$ then the change from x_2 to x'_2 is preferred to the change from x_1 to x'_1 .

Effect of an Action on an Agent's Parameter

We define a function to compute the degree of effect of an action on a parameter within the interest area of an agent as follows:

$$g_{\alpha_i p_j a_k}(p_j, p'_j, D'_{p_j}, \psi_{\alpha_i p_j}) = \frac{(p_j - p'_j)\psi_{\alpha_i p_j}}{h_{\alpha_i p_j}(p_j, p'_j, D'_{p_j})}$$

where

α_i is the evaluating agent; $\alpha_i \in \mathcal{A}$,

p_j is the parameter within an issue of agent α_i ; $p_j \in P(\alpha_i)$, $(\exists \beta_l \in Issues(\alpha_i)) [p_j \in P(\alpha_i, \beta_l)]$,

a_k is the action affecting the parameter p_j of α_i ; $a_k \in \mathcal{Act}$,

D'_{p_j} is the domain for p_j ; $D'_{p_j} \in D(\alpha_i)$,

$\psi_{\alpha_i p_j}$ is the desired direction defined by agent α_i for its parameter p_j , $\psi_{\alpha_i p_j} \in \Psi_{\alpha_i}$,

$h_{\alpha_i p_j}$ is a function for normalization defined below.

The normalization function, h_{α, p_j} , returns the base value for which the change will be normalized according to the cases introduced above.

$$h_{\alpha, p_j}(p_j, p'_j, D'_{p_j}) = \begin{cases} upper(D'_{p_j}) - lower(D'_{p_j}) & \text{for case I} \\ max(p_j, p'_j) - lower(D'_{p_j}) & \text{for case II} \\ upper(D'_{p_j}) - min(p_j, p'_j) & \text{for case III} \end{cases}$$

Effects of an Action on an Agent from an Issue-Perspective

Each agent evaluates the degree of effect of an action from all of its issues. Total effect of an action from an issue of an agent is defined as follows:

$$f_{\alpha_i, \beta_j, a_k} = \sum_{p_l \in P(\alpha_i, \beta_j)} \lambda_{\alpha_i, \beta_j, p_l} \cdot g_{\alpha_i, p_l, a_k}(p_j, p'_j, D'_{p_j}, \psi_{\alpha_i, p_j})$$

Here, $f_{\alpha_i, \beta_j, a_k}$ denotes the degree of effect caused by action a_k from issue β_j of agent α_i . $\lambda_{\alpha_i, \beta_j, p_l}$ denotes the degree of importance that agent α_i gives to its parameter p_l in its issue β_j . g_{α_i, p_l, a_k} denotes the degree of satisfaction caused by action a_k on agent α_i 's parameter p_l .

Total Effect of an Action on an Agent

Total degree of effect of an action on an agent is computed as follows:

$$s_{\alpha_i, a_k} = \sum_{\beta_j \in Issues(\alpha_i)} \sigma_{\alpha_i, \beta_j} \cdot f_{\alpha_i, \beta_j, a_k}$$

Here, s_{α_i, a_k} denotes the total degree of effect caused by action a_k on agent α_i from all of its issues. $\sigma_{\alpha_i, \beta_j}$ denotes the degree of importance given to issue β_j by agent α_i .

Total Effect of a Proposal on an Agent

Total degree of effect of a proposal on an agent is computed as follows:

$$\mu_{\alpha_i, q} = \sum_{a_k \in Act_q} s_{\alpha_i, a_k}$$

Here, $\mu_{\alpha_i, q}$ denotes the total degree of effect caused by proposal q on agent α_i . Act_q denotes the actions proposed by the proposal q . This value reflects an agent's total degree of satisfaction (or dissatisfaction) caused by a proposal. Three cases are possible

- $\mu_{\alpha_i, q} > 0$ means that agent α_i is positively affected by proposal q .
- $\mu_{\alpha_i, q} < 0$ means that agent α_i is negatively affected by proposal q .
- $\mu_{\alpha_i, q} = 0$ means that agent α_i is not affected in any way by proposal q .

An agent α_i detects a conflict on a proposal q if it is negatively affected by the proposal. In other words, if $\mu_{\alpha_i, q} < 0$ then the agent detects a conflict.

6.1.2 Conflict Detection Algorithm

Fig. 6.1 outlines the basic steps in proposal evaluation and conflict detection. When a proposal is asserted, each agent evaluates the solution based on these issues. However, it is not necessary for an agent to criticize a proposal from all of its issues since it may not have the knowledge to criticize the proposal from some perspectives. An agent may detect simultaneous several conflicts in a proposal based on different issues. Each such conflict is specified in the evaluation result tuple that will be put into the shared medium.

In evaluating a proposal, an agent first determines the set of parameters, affected by all of the actions within the proposal. A parameter can be identified as an affected parameter in two different ways. First, a parameter can be explicitly offered a new value by an action. Second, actions in the proposal may indirectly cause a parameter to be offered a new value.

After determining the set of affected parameters, an agent starts computing the degree of effect of each action proposed. First, the agent identifies which of the affected parameters is directly or indirectly offered new values by the action under consideration. Second, taking into account these parameters, the agent computes the partial degree of satisfaction caused by the newly offered values for the parameters. Third, the agent examines its issue set and identifies which of

```

Algorithm Evaluate-Proposal( $q$ )
begin
   $Ra = \emptyset$ 
   $\mu_{\alpha_i} = 0$ 
  for each  $a_k \in Act_q$ 
    begin
       $P_{\alpha_i, a_k} = \emptyset$ 
      for each  $p_j \in P(\alpha_i)$ 
        begin
           $g_{\alpha_i, p_j, a_k} = 0$ 
          if  $p_j$  is affected by action  $a_k$  then
             $P_{\alpha_i, a_k} = P_{\alpha_i, a_k} \cup \{p_j\}$ 
          end
        for each  $p_j \in P_{\alpha_i, a_k}$ 
          begin
             $\Delta_p = (offered(p_j) - current(p_j)) * \psi_{\alpha_i, p_j}$ 
            case FunctionType( $\alpha_i, p_j$ ) of
              begin
                1:  $\Delta_d = upper(D'_{p_j}) - lower(D'_{p_j})$ 
                2:  $\Delta_d = \max(offered(p_j), current(p_j)) - lower(D'_{p_j})$ 
                3:  $\Delta_d = upper(D'_{p_j}) - \min(offered(p_j), current(p_j))$ 
              end
             $g_{\alpha_i, p_j, a_k} = \Delta_p / \Delta_d$ 
          end
         $s_{\alpha_i, a_k} = 0$ 
        for each  $\beta_j \in issues(\alpha_i)$ 
          begin
             $f_{\alpha_i, \beta_j, a_k} = 0$ 
            for each  $p_l \in P(\alpha_i, \beta_j)$ 
               $f_{\alpha_i, \beta_j, a_k} = f_{\alpha_i, \beta_j, a_k} + \lambda_{\alpha_i, \beta_j, p_l} * g_{\alpha_i, p_l, a_k}$ 
             $s_{\alpha_i, a_k} = s_{\alpha_i, a_k} + \sigma_{\alpha_i, \beta_j} * f_{\alpha_i, \beta_j, a_k}$ 
          end
         $\mu_{\alpha_i, q} = \mu_{\alpha_i, q} + s_{\alpha_i, a_k}$ 
         $Ra = Ra \cup \{s_{\alpha_i, a_k}\}$ 
      end
    if  $Re = u_{\alpha_i, q} \leq 0$  then  $Rd = \text{conflicting}$ 
    else  $Rd = \text{non-conflicting}$ 
    insert the evaluation tuple  $\langle q, \alpha_i, Ra, Re, Rd \rangle$  into the shared medium
  end Evaluate-Proposal

```

Figure 6.1: Proposal Evaluation and Conflict Detection within Agent a_i

its issues are involved. Then it computes the degree of satisfaction from all of its issues. The total degree of effect of the action is computed by combining scores computed from all issues according to the issues' preferences. Finally, the agent forms the evaluation tuple and indicates whether it has detected a conflict, or not.

6.2 Multi-Agent Conflict Resolution

Agents involved in a conflict situation start negotiating over the proposed solution. Figure 6.2 outlines the basic steps in the conflict resolution phase. Agents may have different roles in the negotiation process according to their knowledge and problem-solving capabilities.

If an agent is not capable of proposing a resolution alternative, it may use its perspective on the issue to constraint the search space of other agents. If it has the ability to counter-propose a resolution, then it tries its domain-dependent strategies first. Agents that detect a conflict from their issues might use relaxation techniques so that an acceptable resolution could be generated even if the resolution alternatives they are proposing cannot be agreed upon.

If an agent detects a conflict and then chooses a strategy to resolve the conflict, this does not mean that the agent may not alter the resolution strategy it has chosen. That is, upon observing the actions of other agents during the conflict resolution phase, the agent may improve its understanding of the overall problem and the particular conflict encountered. This feature allows agents to alter strategies that they think will benefit from a change. An agent may decide to constrain the search space of others if it is counter-proposing alternatives which are based on *shallow* knowledge. Also, the agent may also quit the conflict resolution phase. Moreover, it may update its conflict management knowledge after a resolution session, which affects its further activities. When an agent proposes a revised solution based on its resolution scheme, it also evaluates the new solution in order to reflect its degree of satisfaction. This enables other agents involved in the conflict to choose the most appropriate action in the resolution

```

Algorithm Conflict-Resolution( $q$ )
begin
  form  $C_{\alpha,q}$  as the set of all constraints violated by actions in  $q$ 
  wait until all agents finish evaluating  $q$ 
  retrieve the set of agents involved in conflict that cannot counter-propose  $A_q^c$ 
  retrieve the set of agents involved in conflict that can counter-propose  $A_q^p$ 
  if  $\alpha_i \in A_q^c$  then
    begin
      perform Constraining (with all constraints)
      if  $score \leq 0$  then
        begin
          update  $C_{\alpha,q}$  in shared medium as
             $C_{\alpha,q} = C_{\alpha,q} - \{c | c \in C_{\alpha,q} \text{ and } type(c) = soft\}$ 
          perform Constraining (with hard constraints)
        end
      end
    end
  else
    begin
      perform Counter-Proposing (with all constraints)
      if  $score \leq 0$  then
        perform Counter-Proposing (with hard constraints)
      end
    end
  if  $score / card(Act_q) \leq threshold$  then
    begin
      if  $\alpha_i$  is the owner of  $q$  then
        delete all proposals and evaluation results for the task set  $T_q$  in  $q$ 
      if  $\alpha_i \in A_q^p$  then
        remove  $T_q$  from  $T(\alpha_i)$ 
        remove any  $po(t_x, t_y)$  where  $t_x$  or  $t_y \in T_q$ , from  $Po(\alpha_i)$ 
      end
    end
  else
    augment the design template  $\mathcal{R}$  with relationships in proposal preferred
    where preferred represents the proposal agreed upon
  end
End Conflict-Resolution

```

Figure 6.2: Conflict Resolution Steps within Agent α_i

```

Procedure Constraining
begin
  insert  $C_{\alpha,q}$  into the shared medium
  wait until the agents in  $A_q^p$  finish inserting alternative proposals
    into the shared medium upon constraining their search
    spaces by the constraints provided by all agents in  $A_q^p$ .
  retrieve the set of alternative proposals from the shared medium,  $Q_q^a$ 
  for each  $q_j \in Q_q^a$ 
    perform evaluate-proposal( $q_j$ )
  wait until the preferred proposal is identified
  retrieve the overall rating, score for the preferred proposal, preferred
end Constraining

```

Figure 6.3: Conflict Resolution Steps for Agent α_i in Constraining Search Space.

process.

After quantifying the total degree of effect of a proposal, an agent checks whether this evaluation process results in a dissatisfaction, or not. If not, the agent has not detected any conflict and can continue its processing. Otherwise, the agent has detected a conflict related to the proposal that has to be resolved. When all agents in the problem-solving network finish evaluating the proposal, agents detected conflict in the proposal are involved in the process of conflict resolution.

The owner of the proposal can generate alternative proposals for the tasks for which the initial proposal has been generated and could not be agreed on. Other agents can contribute to the process of generation of alternative proposals in two ways:

- an agent involved in the conflict may not have expertise to generate alternative proposals, rather it can restrict the search space of other agents so that its priorities are also considered in the generation of new proposals.
- an agent involved in the conflict may have expertise to generate alternative proposals.

In the first case, an agent restricts the search spaces of other agents that have potential for counter-proposing alternatives (Fig. 6.3). In order to do that, the


```

Procedure Counter-Proposing
begin
  wait until all agents in  $A_q^c$  finish asserting their constraints into
  the shared medium
   $C_q = \emptyset$ 
  for each  $\alpha_k \in A_q^c$ 
    begin
      retrieve  $C_{\alpha_k q}$ 
       $C_q = C_q \cup C_{\alpha_k q}$ 
    end
  generate an alternative proposal  $q'$  constraining
  search space with constraints in  $C_q$ 
  update in shared medium  $Q_q^a = Q_q^a \cup \{q'\}$ 
  wait until all agents in  $A_q^p$  finish asserting their
  alternative proposals into the shared medium
  retrieve the set of alternative proposals  $Q_q^a$ 
  for each  $q_j \in Q_q^a$ 
    perform evaluate-proposal( $q_j$ )
  wait until all agents finish evaluating proposals  $Q_q^a$ 
  if  $\alpha_i$  is the owner of  $q$  then
    perform Choose-Proposal( $q$ )
  wait until the preferred proposal is identified
  retrieve the overall rating, score for the preferred proposal, preferred
end Counter-Proposing

```

Figure 6.4: Conflict Resolution Steps for Agent α_i in Counter-Proposing Alternatives.

agent provides all of its relevant constraints to be violated by the actions within the proposal under consideration. These constraints are the ones that cause some parameters to take certain values which result in changes occurring in undesired directions. The agent presents its relevant constraints to all agents by asserting them to the shared medium. The constraints are classified as *hard* constraints and *soft* constraints. Hard constraints are the ones that must be satisfied by any candidate proposal. Soft constraints can be relaxed during the design process and are not essential for achieving a globally satisfiable solution. In this way, the agent forces other agents to generate alternative proposals taking into account its expectations.

In the second case, an agent may have the ability to generate an alternative

proposal in two different ways (Fig. 6.4). First, the tasks for which the original proposal was recommended can also be within the problem-solving scope of another agent. Therefore the agent can generate a completely new candidate proposal for the same task set. Second, not the whole tasks, but rather some of its subtasks can be within the problem-solving scope of the agent. This results in the generation of an alternative proposal which is a revised form of the original conflicting proposal. Moreover, the agent can introduce new tasks to enhance the original task set with new ones, and hence it can generate a candidate proposal. In this way, the agent can propose a new solution whose task set is the super set of the original task set.

After all agents involved in the conflict generate their alternative proposals, they post them to the shared medium. Later, the agents pick up each of the alternative proposals one by one and evaluate them. The next task is to identify a proposal among the candidates which will be acceptable by all agents (Fig. 6.5). A score is computed for each alternative proposal by combining evaluation results of all agents. This score is computed as the minimum of these evaluation results. It represents the global effect of the proposal on the design. The aim is to reduce the dissatisfaction of the most negatively affected agent. After finding the degree of dissatisfaction or satisfaction caused by each alternative solution, agents collaboratively choose a proposal which will be acceptable by all agents.

In this process, the aim is to choose a proposal among the alternative proposals including the original one that has the most positive effect on all agents. This is done by selecting the proposal which has the maximum rating among others. Later, the design template residing in the solution area of the shared medium is updated. This is done by augmenting the partial design template (the set of agreed relationships asserted so far) with new relationships to be established by the execution of actions in the accepted proposal.

The rating for the preferred proposal may drop under a predefined threshold value (which is supposed to be less than 0). This means that any proposal which has rating below the threshold cannot be considered as a solution. When the ratings for all candidate proposals drops under the threshold value, it is necessary

```

Procedure Choose-Proposal
begin
  preferred = q
  score = Re for q
  for each  $q_j \in Q_q^a$ 
    begin
       $score_{q_j} = 1$ 
      for each  $\alpha_t \in (A_q^c \cup A_q^a)$ 
        begin
          retrieve Re for  $q_j$  from the evaluation tuple owned by  $\alpha_t$ 
          if  $Re < score_{q_j}$  then
             $score_{q_j} = Re$ 
          end
        if  $score_{q_j} > score$  then
          begin
             $score = score_{q_j}$ 
             $preferred = q_j$ 
          end
        end
      end
    end
  assert preferred proposal, preferred, and its rating, score
  into the shared medium.
  signal others that the preferred solution is identified
end Choose-Proposal

```

Figure 6.5: Conflict Resolution Steps for Agent α_i in Selecting an Alternative.

to delete some subset of the original tasks, or even the whole tasks from the task sets of all agents along with their associated dependencies. Therefore, it is possible to make progress in the design process.

Chapter 7

Examples of Cooperating Experts Problems

In this chapter, the computational model described in the previous chapters are applied on two different design problem domains. The first example is chosen from the domain of office design. The other example illustrates the application of the model to the problem of configuring a personal computer. These problems exemplify the applicability of *NEPTUNE* to various types of problem characteristics.

7.1 Office Design

The following example is taken from the domain of office design to exemplify the problem-solving process used by cooperating agents in our implementation. The motivation for choosing this example is that it is in a concrete, rather than an abstract, domain and that it can be understood easily because of its suitability for simple, two-dimensional graphical representation. Here, we present a simplified layout problem for an office design and describe design agents and their interactions. A well-designed office encompasses different areas of expertise concerning aesthetics, functionality, energy efficiency, etc. In this example we have incorporated four agents in the framework. They are

- the client agent,
- the functionality agent,
- the electricity agent, and
- the cost agent.

The client agent is the one that puts the problem definition specifying general constraints and the global design goal to be satisfied. It may be invoked by a person who uses the office being designed or is the department chairman who is having the office designed for a prospective faculty member. The functionality agent specializes in the efficient use of objects and spaces. Electricity agent is concerned with all the electrical and electronical devices including computers, telephones, facsimile systems, etc. It is interested in their maintenance issues and wiring. The cost agent is required to control the overall cost of the design and avoid wasteful use of resources. It may propose less expensive alternatives for proposed objects. When a proposal is generated, each interested agent evaluates it to detect a possible conflict from its own perspective. A conflict is detected when an agent finds a conflict situation (upon examining its knowledge-base) that matches the proposal under consideration.

The design process is initiated by the client agent that puts the following problem definition into the problem area of the shared blackboard. As an example of a task description, task *t4* indicates that an object instance from domain *pcdesks* should be selected and assigned to parameter *p_pcdesk*. Similarly, tasks *t5* and *t6* indicate instantiation of the location parameters for the object instance selected in task *t4* and their relevant domains.

```
problem_tuple(client_agent,
  [task(t1,p_desk,desks),
   task(t2,p_desk_locx,locationx),
   task(t3,p_desk_locy,locationy),
   task(t4,p_pcdesk,pcdesks),
   task(t5,p_pcdesk_locx,locationx),
```

```

    task(t6,p_pcdesk_locy,locationy),
    task(t7,p_lampx,lamps),
    task(t8,p_lampy,lamps),
    task(t9,p_lampz,lamps), ...],
[po(t1,t2),
 po(t1,t3),
 po(t4,t5),
 po(t4,t6), ..],
[constraint(c1,[p_total_cost], P_total_cost < 1000),
 constraint(c2,[p_pc,p_pcdesk], P_pc==mac, P_pcdesk = pcdesk6),
 ...],
[layoutobject(room),
 domains([rooms,doors,windows,eplugs,pplugs,lplugs...]),
 domaintype(rooms,complex),
 domaintype(doors,complex),
 domaintype(windows,complex),
 domaintype(eplugs,complex),
 domaintype(pplugs,complex),
 domaintype(lplugs,complex),
 domain(rooms,[room1,...]),
 domain(doors,[door1,...]),
 domain(windows,[window1,...]),
 domain(eplugs,[eplug1,...]),
 domain(pplugs,[pplug1,...]),
 domain(lplugs,[lplug1,lplug2,lplug3,...]),
 object(room1,rooms),
 attributes(room1,[shape,length,width,height,
    door>window,eplug, pplug,lplugx,lplugy,lplugz]),
 attribute(room1,shape,symbolic,rectangular),
 attribute(room1,width,numeric,5),
 attribute(room1,length,numeric,4),

```

```

attribute(room1,height,numeric,2.5),
attribute(room1,door,complex,door1),
attribute(room1>window,complex>window1),
attribute(room1,eplug,complex,eplug1),
attribute(room1,pplug,complex,pplug1),
attribute(room1,lplugx,complex,lplug1),
attribute(room1,lplugy,complex,lplug2),
attribute(room1,lplugz,complex,lplug3),
object(door1,doors),
attributes(door1,[shape,cornerx,cornery,length,
                 width,height,made]),
attribute(door1,shape,symbolic,rectangular),
attribute(door1,cornerx,numeric,5),
attribute(door1,cornery,numeric,3),
attribute(door1,length,numeric,1),
attribute(door1,width,numeric,0.2),
attribute(door1,height,numeric,2),
attribute(door1,made,symbolic,wood) ...]),

```

Fig. 7.1 shows the global layout of an office. In this example, we ignore the third dimension; instead the height attribute of objects is used only when necessary. Also we are not concerned with the precise coordinates of objects. After examining the problem definition, all of the interested parties start producing their design commitments. First, the functionality agent, according to its expertise and understanding of the problem, asserts the following proposal into the proposal area of the shared medium which updates the template as shown in the Fig. 7.2.

```

proposal(proposal_0,
         functionality_agent,
         [t1,t2,t3,t4,t5,t6],
         [ assign(p_desk,desk1),

```

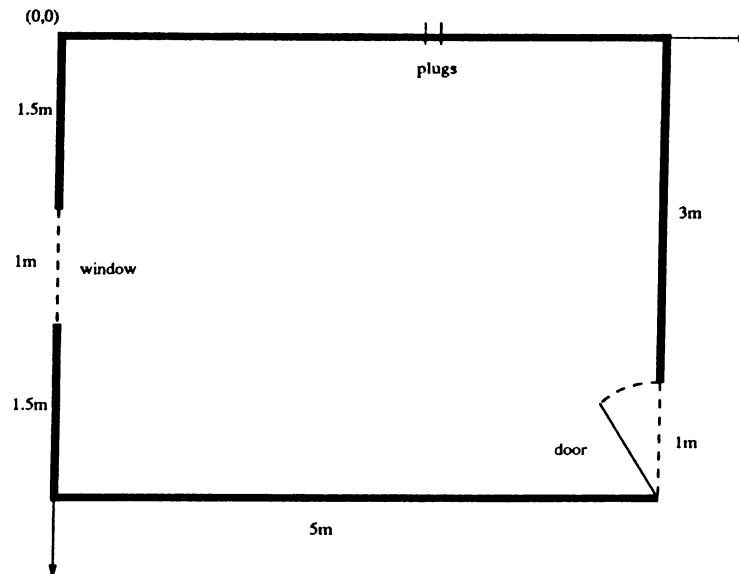


Figure 7.1: Global Layout of the Office

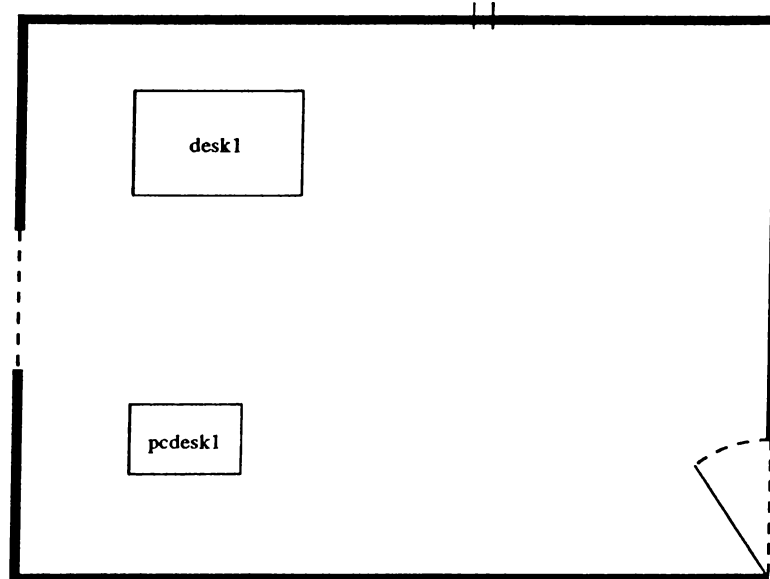
```

assign(p_desk_locx,0.75),
assign(p_desk_locy,0.5),
assign(p_pcdesk,pcdesk1),
assign(p_pcdesk_locx,0.75),
assign(p_pcdesk_locy,2.75)]
[ add(on(desk1,layout)),
  add(on(pcdesk1,layout)),
  add(location(desk1,0.75,0.5),
    add(location(pcdesk1,0.75,2.75)))]

```

The functionality agent has decided to put a desk and a PC desk nearer to the window so that the occupant could not only have a good view but also utilize daylight.

This proposal triggers the evaluation procedures within other interested agents. The client agent detects a conflict after evaluating the proposal. With this configuration, the client agent notices that since the occupant is going to be an engineer who will be using the computer frequently, (s)he must walk too much due to the distance between the main desk and the PC desk (that will be put on

Figure 7.2: Layout of the Office After proposal_{1.0}

the PC desk). The client agent detects the conflict from its *usability* issue. p_desk_locx , p_desk_locy , p_pcdesk_locx and p_pcdesk_locy are the only parameters within issue *usability* for which new values are offered. p_desk_locx and p_desk_locy are not affected positively or negatively by the proposed values whereas p_pcdesk_locx and p_pcdesk_locy are affected negatively. This means that changes offered for the values of parameters p_pcdesk_locx and p_pcdesk_locy do not occur in the desired direction. Quantification for the total degree of effects for the parameters p_pcdesk_locx and p_pcdesk_locy from the only affected issue *usability* results in a negative value. Since no other issue in client agent's issue list has been activated, the total degree of satisfaction will be negative leading to a conflicting situation.

The electricity agent detects a conflict from its *configurability* issue. No parameter within this issue has been offered new value. However, parameters p_pcdesk_locx and p_pcdesk_locy indirectly affects the location parameters of the PC, which is represented by p_pc_locx and p_pc_locy . This is due to the fact that location of the PC and the PC desk should almost be the same, since a PC will be put on a PC desk. Electricity agent computes the degree of effect of

the indirectly offered values for parameters `p_pc_locx` and `p_pc_locy` within its issue *configurability*. This evaluation results in a dissatisfaction since electricity agent wants to keep electrical devices close to the electricity plug which is not the case. Quantification for the total degree of satisfaction from all issues results in a negative value leading to a conflicting situation. The client and electricity agents assert the following evaluation results into the shared medium.

```
evaluation_result(proposal_0,
                  client_agent,
                  [0,0,0,0,-0.12,-0.18],
                  -0.3,
                  conflicting_proposal)
```

```
evaluation_result(proposal_0,
                  electricity_agent,
                  [0,0,0,0,-0.1,-0.3],
                  -0.4,
                  conflicting_proposal)
```

The functionality, client and electricity agents combine to resolve the conflict encountered. The client agent uses a specific resolution scheme which states that “keep frequently used objects close to each other.” Since it cannot propose any alternative proposal, it can only constrain other agents’ search spaces. The functionality agent is capable of generating alternative proposals and tries other location alternatives. The electricity agent is capable of generating an alternative solution only for a subset of the original task set, namely [t5,t6].

The functionality agent has two alternatives to resolve the conflict from its perspective. It may put the PC desk either to the left, or to the right of the other desk. The functionality agent proposes to put the PC desk to the left of the other desk so that the PC desk will be close to the window and hence the occupant can utilize daylight and have a better view (Fig. 7.3). The electricity agent counter-propose an alternative in which it moves PC desk close to the electricity plug

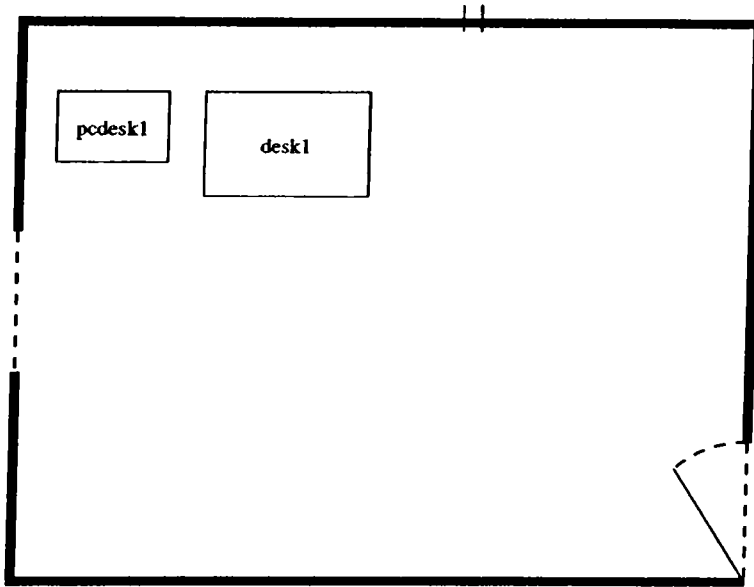


Figure 7.3: Layout of the Office After Resolution Alternative proposal 1.1

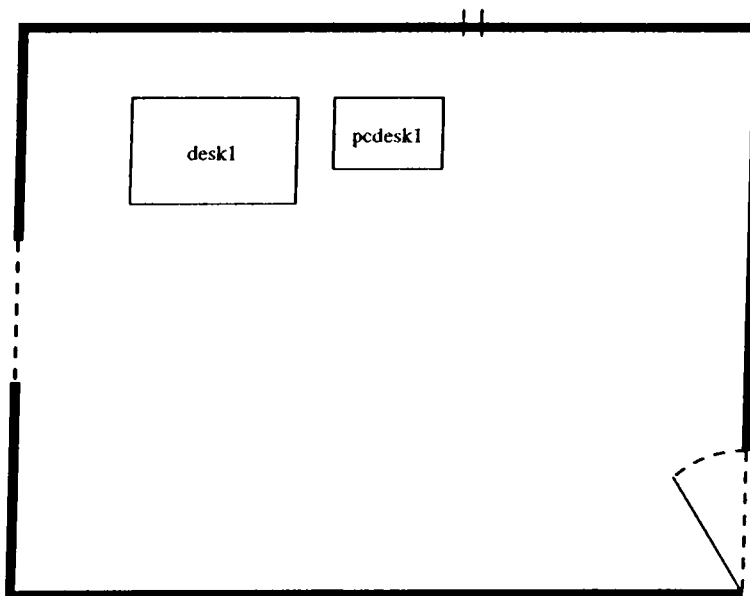


Figure 7.4: Layout of the Office After Resolution Alternative proposal 1.2

(Fig. 7.4). The functionality and electricity agents assert the following conflict resolution tuples along with their counter-proposals.

```

conflict_resolution(proposal_0,
    client_agent,
    constraining,
    [constraint(c8, [p_desk, p_desk_locx, p_desk_locy,
        p_pcdesk, p_pcdesk_locx, p_pcdesk_locy],
        compute_place(P_desk, P_pcdesk, P_desk_locx, P_desk_locy
            Rxu, Rxl, Ryu, Ryl),
        P_pcdesk_locx < Rxu, P_pcdesk_locx > Rxl,
        P_pcdesk_locy < Ryu, P_pcdesk_locy > Ryl])
conflict_resolution(proposal_0,
    electricity_agent,
    counter_proposing,
    [t5, t6]).
proposal(proposal_1,
    functionality_agent,
    [t1, t2, t3, t4, t5, t6],
    [ assign(p_desk, desk1),
      assign(p_desk_locx, 1.25),
      assign(p_desk_locy, 0.5),
      assign(p_pcdesk, pcdesk1),
      assign(p_pcdesk_locx, 0.25),
      assign(p_pcdesk_locy, 0.5)]
    [ add(on(desk1, layout)),
      add(on(pcdesk1, layout)),
      add(location(desk1, 1.25, 0.5)),
      add(location(pcdesk1, 0.25, 0.5))]
proposal(proposal_2,
    electricity_agent,
    [t1, t2, t3, t4, t5, t6],

```

```
[ assign(p_desk,desk1),
  assign(p_desk_locx,0.75),
  assign(p_desk_locy,0.5),
  assign(p_pcdesk,pcdesk1),
  assign(p_pcdesk_locx,3.15),
  assign(p_pcdesk_locy,0.5)]
[ add(on(desk1,layout)),
  add(on(pcdesk1,layout)),
  add(location(desk1,0.75,0.5)),
  add(location(pcdesk1,3.15,0.5)))]
```

After generation of alternatives, agents involved in the conflict evaluate `proposal_1` and `proposal_2` in the same way. The client agent is not negatively affected from any of its issues since both solutions keep desks `desk1` and `pcdesk1` close to each other. On the other hand, the electricity agent is negatively affected by the offered values for `p_pcdesk_locx` and `p_pcdesk_locy` in `proposal_1` which indirectly causes PC to be far away from the electricity plug. The functionality agent is not negatively affected by `proposal_2` since the proposal does not cause any of its parameters to be changed in an undesired direction. The evaluation results for `proposal_1` and `proposal_2` are given below:

```
evaluation_result(proposal_1,
                  client_agent,
                  [0,0,0,0,0.1,0.15],
                  0.25,
                  nonconflicting_proposal)
```

```
evaluation_result(proposal_1,
                  electricity_agent,
                  [0,0,0,0,-0.3,0],
                  -0.3,
                  conflicting_proposal)
```

```

evaluation_result(proposal_2,
                  client_agent,
                  [0,0,0,0,0.09,0.08],
                  0.17,
                  conflicting_proposal)

```

```

evaluation_result(proposal_2,
                  functionality_agent,
                  [0,0.1,0.05,0,0.05,0.05],
                  0.25,
                  conflicting_proposal)

```

In the resolution phase, overall score for each of the candidate proposal has to be computed. Table 7.1 contains agents' evaluation results of alternative proposals `proposal_0`, `proposal_1` and `proposal_2` for the task set $[t_1, t_2, t_3, t_4, t_5, t_6]$. These quantitative values reflect agents' degree of satisfaction or dissatisfaction upon execution of the actions in the proposals. In order to resolve the conflict, agents jointly identify a proposal that will be acceptable by all agents. First, a score is computed for each candidate proposal which is the minimum of evaluation results for the proposal. The aim is to take into account the most negatively affected agent's perspective. In this case $\text{score}(\text{proposal}_0) = -0.4$, $\text{score}(\text{proposal}_1) = -0.3$ and $\text{score}(\text{proposal}_2) = 0$. In the second phase, the proposal having the maximum score value is identified and chosen to be the preferred solution for the task set $[t_1, t_2, t_3, t_4, t_5, t_6]$.

In the above example, `proposal-2` is chosen as the solution since it has the highest score. After the resolution phase, design template in the solution area of the shared medium is updated by the relationships within the accepted proposal. After the resolution steps these three agents come to an agreement to put the PC desk close to the electricity plug as shown in Fig 7.4.

Suppose that at a particular time, the electricity agent takes the next task in its task queue, which is to "put lamps into the slots in the ceiling." The electricity

Ratings Computed for Alternative Proposals			
	proposal_0	proposal_1	proposal_2
client_agent	-0.3	0.25	0.17
electricity_agent	-0.4	-0.3	0.25
functionality_agent	0.4	0.3	0.25
cost_agent	0.0	0.0	0.0

Table 7.1: Evaluation Results for proposal_0, proposal_1 and proposal_2

agent generates a solution for this task in which it proposes to fix three 100 Watts lamps into the plugs in the ceiling as follows:

```
proposal(proposal_3,
  electricity_agent,
  [t20,t21,t22],
  [ assign(p_lampx,lamp11),
    assign(p_lampy,lamp12),
    assign(p_lampz,lamp13),
  [ add(fixed(lamp11,lplugx)),
    add(fixed(lamp12,lplugy)),
    add(fixed(lamp13,lplugz)) ]])
```

After examining the proposal, the cost and functionality agents detect conflicts and assert their evaluation results into the conflict area of the shared medium. The cost agent argues that having a total of 300 Watts of lighting is too costly, detecting a conflict from its *energy-cost* issue. Therefore, the cost agent disagrees with the lamp instances lamp11, lamp12 and lamp13 selected by the electricity agent for parameters p_lampx, p_lampy and p_lampz. Because these parameters cause p_c_lampx, p_c_lampy and p_c_lampz within *energy-cost* issue of the cost agent to be indirectly affected with new assignments. The functionality agent evaluates the proposal and detects a conflict from the *effective-functionality* issue. The evaluation result tuples to be asserted are

```

evaluation_result(proposal_3,
                  cost_agent,
                  [-0.1,-0.1,-0.1],
                  -0.3,
                  conflicting_proposal)

```

```

evaluation_result(proposal_3,
                  functionality_agent,
                  [-0.2,-0.2,-0.2],
                  -0.6,
                  conflicting_proposal)

```

The electricity, cost and functionality agents come together to resolve the conflict. The cost agent cannot counter-propose alternatives rather it can only restrict the search spaces of others. The electricity agent can propose a new alternative and finds plenty of options to meet the concerns of the cost agent by looking at its database and offering a total lighting package that can be accepted by the cost agent. Meanwhile, the functionality agent is capable of counter-proposing and has a good resolution alternative for the conflict. It claims that the level of lighting accepted by both of the agents can be lowered further. Since the occupant of the office will be working alone most of the time, it is possible to lower the total lighting power consumption by introducing a desk lamp that might be put on top of the desk. Therefore, it generates an alternative proposal by enhancing the original task list with a new task. The list of agents' candidate proposals is given below:

```

proposal(proposal_4,
         electricity_agent,
         [t20,t21,t22],
         [ assign(p_lampx,lamp41),
           assign(p_lampy,lamp42),
           assign(p_lampz,lamp43),

```


Ratings Computed for Alternative Proposals			
	proposal.3	proposal.4	proposal.5
client_agent	0.0	0.0	0.0
electricity_agent	0.2	0.15	0.2
functionality_agent	-0.6	0.1	0.2
cost_agent	-0.3	0.05	0.08

Table 7.2: Evaluation Results for proposal.3, proposal.4 and proposal.5.

```

[ add(fixed(lamp41,lplugx)),
  add(fixed(lamp42,lplugy)),
  add(fixed(lamp43,lplugz)) ]

proposal(proposal_5,
  functionality_agent,
  [t20,t21,t22,t78],
  [ assign(p_lampx,lamp31),
    assign(p_lampy,lamp32),
    assign(p_lampz,lamp33),
    assign(p_desklamp,desklamp1) ]
  [ add(fixed(lamp31,lplugx)),
    add(fixed(lamp32,lplugy)),
    add(fixed(lamp33,lplugz))
    add(on(desklamp1,desk1)) ]
  add(location(desklamp1,4,2)))

```

Evaluation results of these alternative proposals are given in Table 7.2. All agents are affected positively by proposal.4 and proposal.5. Proposal.5 is the solution acceptable by all agents. In the resolution phase, the cost agent constrains the search space so that the total amount of lighting power should be less than 220 Watts. The functionality agent uses a domain-dependent resolution alternative to augment a final solution that is acceptable to all of the agents.

The design proceeds in this manner until it reaches the requirements specified by the client agent. In this example, we only gave a segment of problem-solving process emphasizing the resolution of conflicts. However, this short scenario sets a good example to illustrate the resolution of conflicts, which is described in *NEPTUNE*. Appendix contains a segment of the sample run of this example.

7.2 Configuring A Personal Computer

In this example, we present a configuration problem for a personal computer and describe design agents and their interactions. The aim is to identify hardware requirements for a personal computer system which will meet needs of an end user. Three agents are involved in solving the configuration problem. They are

- the client agent,
- the technical agent, and
- the cost agent.

The client agent defines the problem specifying general constraints and the global design goal to be satisfied. The technical agent specializes in hardware components and their functionality. The cost agent controls the overall cost of the design and avoid wasteful use of resources. The client agent initiates the design process putting the following problem definition into the problem area of the shared blackboard.

```
problem_tuple(client_agent,
  [task(t1,p_processor,processors)
   task(t2,p_resolution,resolutions)
   task(t3,p_color,colors)
   task(t4,p_speed,speeds)
   task(t5,p_fdisk,fdisks)
   task(t6,p_hdisksize,hdisksizes)
```

```

    task(t7,p_ramsize,ramsizes) ...],
[po(t1,t4),
  po(t2,t3),
  po(t4,t5), ...],
[constraint(c1,[p_total_cost], P_total_cost < 2000),
  constraint(c2,[p_processor,p_coprocessor_cost,p_processor_cost],
  P_processor=p486, P_processor_cost=P_coprocessor_cost-20),...],
[layoutobject(pc_layout)
  domains([processors,resolutions,colors,speeds,fdisks,
           hdisksizes,ramsizes ...])
  domaintype(processors,complex)
  domaintype(resolutions,complex)
  domaintype(colors,symbolic)
  domaintype(speeds,numeric)
  domaintype(fdisks,symbolic)
  domaintype(hdisksizes,numeric)
  domaintype(ramsizes,numeric)
  domain(processors,[p86,p88,p386,p486,...])
  domain(resolutions,[res1,res2,res3,...])
  domain(colors,[black_white,rgb])
  domain(speeds,[15,33])
  domain(fdisks,[f3_2,f5_4])
  domain(hdisksizes,[10,80])
  domain(ramsizes,[1,8])
  object(p68,processors)
  attributes(p68,[wlength,bus_size,speed, ...])
  attribute(p68,wlength,numeric,8)
  attribute(p68,bus_size,numeric,8)
  attribute(p68,speed,numeric,15) ...])

```

After examining the problem definition, all of the interested parties start producing their design commitments. First, the technical agent asserts the following

proposal into the proposal area of the shared medium.

```
proposal(proposal_10,
         technical_agent,
         [t1,t2,t3],
         [ assign(p_processor,p88),
           assign(p_resolution,res1),
           assign(p_color,rgb) ],
         [ add(comp(pc_layout,processor,p88)),
           add(comp(screen,resolution,res1)),
           add(comp(screen,color,rgb)) ])
```

The technical agent offers an INTEL 8088 processor, and a 400*600 color monitor. This proposal triggers the evaluation procedures within other interested agents. The client agent detects a conflict concerning the proposal. The client agent states that the user is to be using windows software which cannot run on a processor of type 8088. The client agent detects the conflict from its *usability* issue. *p_processor* is the only parameter within issue *usability* for which new values are offered. Quantification for the total degree of effects for the parameters *p_processor* from the only affected issue *usability* results in a negative value. Since no other issue in client agent's issue list has been activated, the total degree of satisfaction will be negative leading to a conflicting situation.

The cost agent detects a conflict from its *hardware_cost* issue. No parameter within this issue has been offered new value directly. However, parameters *p_resolution* and *p_color* indirectly affects the cost parameters of screen, which is represented by *p_resolution_cost* and *p_color_cost*. The cost agent computes the degree of effect of the indirectly offered values for parameters *p_resolution_cost* and *p_color_cost* within its issue *hardware_cost*. This evaluation results in a dissatisfaction since the overall hardware cost of the design is increased. Quantification for the total degree of satisfaction from all issues results in a negative value leading to a conflicting situation. The client and cost agents assert the following evaluation results into the shared medium.

```

evaluation_result(proposal_10,
                  client_agent,
                  [-0.4,0,0],
                  -0.4,
                  conflicting_proposal)

```

```

evaluation_result(proposal_10,
                  cost_agent,
                  [0,-0.1,-0.2],
                  -0.3,
                  conflicting_proposal)

```

The technical, client and cost agents combine to resolve the conflict encountered. The cost agent cannot propose any alternative proposal, rather it can only constrain other agents' search spaces. The technical agent is capable of generating alternative proposals and tries other alternatives. The client agent is capable of generating an alternative solution only for a subset of the original task set, namely [t1]. The agents assert the following conflict resolution tuples along with their counter-proposals.

```

conflict_resolution(proposal_10,
                    cost_agent,
                    constraining,
                    [constraint(c5, [p_resolution, p_color],
                                P_resolution_x < 600,
                                P_resolution_y < 800,
                                P_color =\= rgb)])
conflict_resolution(proposal_10,
                    client_agent,
                    counter_proposing,
                    [t1]).

```

```

proposal(proposal_11,
         technical_agent,
         [t1,t2,t3],
         [ assign(p_processor,p386),
           assign(p_resolution,res2),
           assign(p_color,black_white) ],
         [ add(comp(pc_layout,processor,p386)),
           add(comp(screen,resolution,res2))
           add(comp(screen,color,black_white)) ])

proposal(proposal_12,
         client_agent,
         [t1,t2,t3],
         [ assign(p_processor,p386),
           assign(p_resolution,res1),
           assign(p_color,rgb) ],
         [ add(comp(pc_layout,processor,p386)),
           add(comp(screen,resolution,res1)),
           add(comp(screen,color,rgb)) ])

```

After generation of alternatives, agents involved in the conflict evaluate `proposal_11` and `proposal_12` in the same way. The evaluation results for `proposal_11` and `proposal_12` are given below:

```

evaluation_result(proposal_11,
                  client_agent,
                  [0.1,0,0],
                  0.1,
                  nonconflicting_proposal)

evaluation_result(proposal_11,
                  cost_agent,
                  [-0.05,0,-0.04],

```

Ratings Computed for Alternative Proposals			
	proposal_10	proposal_11	proposal_12
client_agent	-0.4	0.1	0.1
technical_agent	0.1	0.08	0.07
cost_agent	-0.3	-0.09	-0.35

Table 7.3: Evaluation Results for proposal_10, proposal_11 and proposal_12

```

-0.09,
conflicting_proposal)

evaluation_result(proposal_12,
    technical_agent,
    [0.07,0,0],
    0.07,
    nonconflicting_proposal)

evaluation_result(proposal_12,
    cost_agent,
    [-0.05,-0.1,-0.2],
    -0.35,
    conflicting_proposal)

```

In the resolution phase, overall score for each of the candidate proposal has to be computed. Table 7.3 contains agents' evaluation results of alternative proposals proposal_10, proposal_11 and proposal_12 for the task set [t1,t2,t3]. In order to resolve the conflict, agents jointly identify the proposal that will be acceptable by all agents. First, a score is computed for each candidate proposal which is the minimum of evaluation results for the proposal. The aim is to take into account the most negatively affected agent's perspective. In this case $\text{score}(\text{proposal}_10) = -0.4$, $\text{score}(\text{proposal}_11) = -0.09$ and $\text{score}(\text{proposal}_12) = -0.35$. In the second phase, the proposal having the

maximum score value is identified and chosen to be the preferred solution for the task set $[t_1, t_2, t_3]$.

In the above example, `proposal_11` is chosen as the solution since it has the highest score. After the resolution phase, design template in the solution area of the shared medium is updated by the relationships within the accepted proposal. In this second example, we aimed at illustrating the applicability of the computational model on a different application domain.

Chapter 8

Conclusions and Future Work

DAI attempts to integrate existing problem-solving methods used in classical AI in order to develop systems that benefit from multiple agents' point of view. Co-operating experts approach has an important role in the field of DAI because many of the problems that are being encountered in real life require the application of complex and diverse expertise. One of the important problems faced in a cooperating community of experts is how to detect and resolve conflicts occurring at any phase of problem-solving. Existing approaches to conflict resolution rely on coordinated conflict resolution strategies. In these approaches, each agent is assumed to have a global knowledge of conflict resolution information. In case of conflicts, they agree on a conflict resolution scheme and a special agent resolves the conflict using a globally agreed resolution strategy.

In this dissertation, we present a computational model, *NEPTUNE*, for co-operative multi-agent systems for solving problems that openly supports multi-agent conflict detection and resolution. Our novel approach allows an agent to choose the most appropriate action given its understanding of the global and local situation and its own capabilities. Each agent has its own conflict resolution knowledge, which is not accessible and known by others. Furthermore, there are no globally known conflict resolution strategies. Each agent involved in a conflict chooses a resolution scheme according to its self-interest. Agents might use different strategies of their own and might still agree on a solution.

NEPTUNE achieves flexibility in its problem-solving which is the most compelling argument for building modular multi-agent systems. A new agent can be added or an existing one can be removed without any modification on the rest of the system. This characteristic of *NEPTUNE* satisfies the requirements of open systems semantics. However, in the existing approaches, addition or removal of an agent requires that the global conflict resolution knowledge be reformed accordingly.

Existing approaches are too restrictive and applicable only to the problems where experts must agree on a known strategy for resolving conflicts. Our approach, we believe, is much similar to the conflict resolution in human problem-solving. This approach also allows agents to alter strategies in resolution phase if they think that it is wise to do that. In the sequel we summarize the salient contributions of our approach:

- Each agent has its own conflict resolution knowledge which is not required to be known by others.
- Each agent may detect conflicts in a proposal based on its different issue-perspectives.
- Each agent can present its degree of satisfaction, or dissatisfaction on a proposal that can be understood by others.
- Agents contribute to the developing solution based on their relevance and problem-solving capabilities.
- Agents involved in a conflict situation may use different strategies for resolving the conflict.
- Agents may alter their own strategies during the process of resolving a particular conflict.
- Agents may generate proposals and deal with resolving conflicts in parallel.
- The model achieves *open systems semantics*.

NEPTUNE is implemented on a network of workstations running under UNIX. All of the problem solvers, agents, are modeled as processes running on different workstations that communicate over internet wide-area network. Agents are fully functional knowledge-based systems and behave autonomously as they are described in the model. There is an equal distribution of control and authority. Therefore, an agent developed by a knowledge engineer may enter the problem-solving environment from a geographically distant site in the network.

Future Research Directions

We believe the research presented in this thesis may form a basis for developing better computational models for DAI. The future research directions are listed below.

- In this thesis, agents are assumed to behave cooperatively. Agents may have solely their own benefit in mind, while in cooperative situations the parties are united by the super-ordinate goal of achieving a global solution. There is a need for developing computational models that support resolution of conflicts in competitive situations.
- The model can be enhanced with the capability to understand the statements made by human participants. This can be done by developing a human model to be incorporated into the system that uses some form of linguistic structure.
- We assume that each agent is a knowledge-based system that makes an offer to solve subproblems (generate proposals) and cooperates with each other in resolving conflicts through negotiation. The model can be augmented to support special agents such as database systems, allowing capabilities of existing systems to be utilized.

This dissertation presented a new approach which contributes to development of a theory of conflict resolution by introducing multi-agent conflict detection and resolution. Therefore it may be used as a basis for developing methodologies to

integrate today's heterogeneous computing environments containing many independent information resources of different types, such as a database management system with its databases, an expert system with its knowledge base, an information repository, an application program, or human beings.

Bibliography

- [1] Adler M. R., et al., "Conflict Resolution Strategies for Non-hierarchical Distributed Agents," *Distributed Artificial Intelligence*, Vol. 2, M.N. Huhns, 1989, pp. 139-161.
- [2] Akman V., et al., "Knowledge Engineering in Design," *Knowledge-Based Systems*, Vol. 1, No. 2, March 1988, pp. 67-77.
- [3] Akman V., ten Hagen P. J. W., and Tomiyama T., "A Fundamental and Theoretical Framework for an Intelligent CAD system," *Computer-Aided Design*, Vol. 22, No. 6, July-August 1990, pp. 352-367.
- [4] Akman V., "Heterogeneous Inference in Design," in *Proceedings of the Engineering Design and Analysis Conference (ESDA)*, PD-Vol. 47-7, ASME, 1992, pp. 143-150.
- [5] Banerjee J., et al., "Data Model Issues for Object-Oriented Applications," *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, January 1987, pp. 3-26.
- [6] Bartos O.J., *Process and Outcome in Negotiation*, Columbia University Press, New York, NY, 1974.
- [7] Bates J., Loyell A. B., and Reilly W. S., "An Architecture for Action, Emotion and Social Behavior," in *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW)*, Rome, Italy, 1992.

- [8] Cammarata S., McArthur D., and Steeb R., "Strategies for Cooperation in Distributed Problem Solving," in *Proc. Int. Joint Conf. Artificial Intelligence*, Karlsruhe, Germany, Aug. 1983, pp. 767-770.
- [9] Chandrasekeran B., "Decomposition of Domain Knowledge into Knowledge Source: The MDX Approach," *Fourth National Conf. of Canadian Society for Computational Studies of Intelligence*, Canada, 1982.
- [10] Conry S. E., Meyer R. A., and Lesser V. R., "Multi-stage Negotiation in Distributed Planning," *Technical Report*, 86-67, Department of Computer and Information Science, University of Massachusetts at Amherst, 1986.
- [11] Conry S. E., MacIntosh D. J., and Meyer R. A., "DARES: A Distributed Automated Reasoning System," in *Proceedings of AAAI*, 1990, pp. 78-85.
- [12] Corkill D. D., Gallagher K. Q., and Murray K. E., "GBB: A Generic Blackboard Development System," in *Proc. AAAI-86*, Philadelphia, 1986, pp. 1008-1014.
- [13] Croft W. B. and Leftkowitz L. S., "Knowledge-based Support of Cooperative Activities," in *Proc. 21st Annual Hawaii Int. Conf. Sys. Sci.*, Vol. 3, 1988, pp. 312-318.
- [14] Durfee E. D., Lesser V. R., and Corkill D. D., "Cooperation Through Communication in a Distributed Problem Solving Network," *Distributed Artificial Intelligence*, Vol. 1, M.N. Huhns, Pitman/Morgan Kaufman, 1987, pp. 29-55.
- [15] Durfee E. D., *Coordination of Distributed Problem Solvers*, Kluwer Academic Publishers, Boston, 1988.
- [16] Durfee E. D., Lesser V. R., and Corkill D. D., "Trends in Cooperative Distributed Problem Solving," *IEEE Transaction on Knowledge and Data Engineering*, Vol. 1, No. 1, March 1989, pp. 63-83.

- [17] Durfee E. D. and Montgomery A., "A Hierarchical Protocol for Coordinating Multi-Agent Behaviors," in *Proceedings of AAAI*, 1990, pp. 86-93.
- [18] Erman L. D., Lark J. S., and Hayes-Roth F., "ABE: An Environment for Engineering Intelligent System," *IEEE Trans. on Software Engineering*, Vol. 14, No. 12, Dec. 1988, pp. 1758-1770.
- [19] Erman L. D., Hayes-Roth F., Lesser V. R., and Reddy D. R., "The Hearsay-II Speech Understanding Systems: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, Vol. 12, June 1980, pp. 213-253.
- [20] Findler N. V. and Lo R., "An Examination of Distributed Planning in World of Air Traffic," *J. Parallel Distributed Comput.*, Vol. 3, 1986, pp. 411-431.
- [21] Findler N. V. and Gao J., "Dynamic Hierarchical Control for Distributed Problem Solving," *Data and Knowledge Engineering*, Vol. 2, 1987, pp. 285-301.
- [22] Fisher G., "Communication Requirements for Cooperative Problem-Solving Strategies," *Information Systems*, Vol. 15, No. 1, 1990, pp. 82-106.
- [23] Gallagher K.Q., et al., GBB Reference Manual, *Technical Report*, 88-66, Department of Computer and Information Science, University of Massachusetts at Amherst, 1988.
- [24] Gasser L., "The Integration of Computing and Routine Work," *ACM Trans. on Office Information Systems*, Vol. 4, No. 3, 1986, pp. 205-225.
- [25] Gasser L., "Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics," *Artificial Intelligence*, Vol. 47, 1991, pp. 107-138.
- [26] Gasser L., Braganza C., and Herman N., "MACE: A Flexible Testbed for Distributed AI Research," *Distributed Artificial Intelligence*, Vol. 1, M.N. Huhns, Pitman/Morgan Kaufman, 1987, pp. 119-152.

- [27] Georgeff M., "Communication and Interaction in Multi-Agent Planning," *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser eds., Morgan Kaufmann, 1988, pp. 200-204.
- [28] Georgeff M., "The Representation of Events in Multi-Agent Domains," in *Proceedings of AAAI*, 1986, pp. 70-75.
- [29] Gero J. S. (ed.), *Artificial Intelligence in Engineering Design*, Elsevier, UK, 1988.
- [30] Gero J. S. (ed.), *Artificial Intelligence in Engineering* 3, Elsevier, 1989.
- [31] Gingberg A., "Knowledge Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy," in *Proceedings of AAAI*, 1988, pp. 4-8.
- [32] Goel V. and Pirollo P., "Design within Information-Processing Theory: The Design Problem Space," *AI Magazine*, Spring 1989, pp. 19-36.
- [33] Gomez F. and Chandrasekaran B., "Knowledge Organization and Distribution for Medical Diagnosis," *Technical Report*, 84-FG-FGBC, Department of Computer and Information Science, The Ohio State University, 1984.
- [34] Gmytrasiewicz P. J., et al., "A Decision-Theoretic Approach to Coordinating Multi-Agent Interactions," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1991, pp. 62-68.
- [35] Hayes-Roth B., "A Blackboard Architecture for Control," *Artificial Intelligence*, Vol. 26, 1985, pp. 251-321.
- [36] Hertzberg J. and Horz A., "Towards a Theory of Conflict Detection and Resolution in Nonlinear Plans," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1989, pp. 937-947.
- [37] Hewitt C., "Offices are Open Systems," *ACM Transactions on Office Information Systems*, Vol. 4, No. 3, 1986, pp. 271-287.

- [38] Hewitt C., "Open Information Systems Semantics for Distributed Artificial Intelligence," *Artificial Intelligence*, Vol. 47, 1991, pp. 79-106.
- [39] Hewitt C., "Traditional Artificial Intelligence and or Open Systems Science," in *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW)*, Rome, Italy, 1992
- [40] Huang G. Q. and Brandon J. A., "Agents: An Object Oriented Prolog System for Cooperating Knowledge-Based Systems," *Knowledge-Based Systems*, Vol. 5, No. 2, June 1992, pp. 125-136.
- [41] Huhns M. N. and Singh M., "The Semantic Integration of Information Models," in *Proceedings of the AAAI-92 Workshop Cooperation Among Heterogeneous Intelligent Agents*, July 1992, San Jose, CA, USA. pp. 38-45.
- [42] Kamel M. and Sayed A., "An Object Oriented Multiple Agent Planning System," in *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser eds., Morgan Kaufmann, 1988, pp. 147-228.
- [43] Klein M. and Lu S. C-Y., "Conflict Resolution in Cooperative Design," *Artificial Intelligence in Engineering*, Vol. 4, No. 4, 1989, pp. 168-180.
- [44] Kraus S. and Wilkenfeld B., "Negotiation Over Time in A Multi-Agent Environments: Preliminary Report," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1991, pp. 56-61.
- [45] Laasri H. and Maitre B., "Cooperating Expert Problem Solving in Blackboard Systems: Atome Case Study," in *Proceedings of the First European Workshop on Modeling Agents in a Multi Agent World (MAAMAW)*, Rome, Italy, 1990.
- [46] Lander S. and Lesser V. R., "Conflict Resolution Strategies for Cooperating Expert Agents," *International Conference on Cooperating Knowledge-Based Systems*, Keele Univ., October 1990.

- [47] Lansky A., "Localized Search for Multi-Agent Planning," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1991, pp. 252-258.
- [48] Leftkowitz L. S. and Croft W. B., "Planning and Execution of Tasks in Cooperative Work Environments," in *Proceedings of IEEE Conference on Artificial Intelligence Applications*, 1989.
- [49] Lesser V. R. and Corkill D. D., "Functionally-Accurate, Cooperative Distributed Systems," *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser, Morgan Kaufmann, 1988, pp. 295-309.
- [50] Lesser V. R. and Corkill D. D., "The Distributed Vehicle Monitoring Testbed: a Tool for Investigating Distributed Problem Solving Networks," in *Blackboard Systems*, R. S. Engelmore and A. Morgan eds., Addison-Wesley, 1988, pp. 353-386.
- [51] Lesser V. R. and Erman L. D., "Distributed Interpretation: A Model and Experiment," *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser eds., Morgan Kaufmann, 1988, pp. 120-139.
- [52] Levesque H. J., Cohen P. R., and Nwnes J. H. T., "On Acting Together," in *Proceedings of AAAI*, 1990, pp. 94-99.
- [53] Malone T. W. and Crowston K., "Towards an Interdisciplinary Theory of Coordination," *Technical Report*, CCS-TR-120, Center for Coordination Science, MIT, Cambridge, 1991.
- [54] Morgenstern L., "A First Order Theory of Planning, Knowledge, and Action," in *Proceedings of the 1986 Conference on Reasoning about Knowledge*, 1986, pp. 99-114.
- [55] Nguyen T. A., et al., "Verifying Consistency of Production Systems," in *Proceedings of the Third Conference on Artificial Intelligence Applications*, 1987, pp. 4-8.

- [56] Nii H. P., "Blackboard Systems: The Blackboard Model of Problem-Solving and the Evolution of Blackboard Architectures," *AI Magazine*, Vol. 7, No. 2, Summer 1986, pp. 39-53.
- [57] Nii H. P., "Blackboard Systems: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective," *AI Magazine*, Vol. 7, No. 3, 1986, pp. 82-106.
- [58] Nirenburg S. and Lesser V. R., "Providing Intelligent Assistance in Distributed Office Environments," *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser eds., Morgan Kaufmann, 1988.
- [59] Parunak H. D., "Manufacturing Experience with the Contract Net," *Distributed Artificial Intelligence*, Vol. 1, M.N. Huhns, Pitman/Morgan Kaufman, 1987, pp. 285-310.
- [60] Polat F. and Guvenir H. A., "Knowledge Base Verification in an Expert System Shell," in *Proceedings of the Third International Symposium on Computer and Information Sciences (ISCIS) Oct 89*, Izmir, pp. 889-898.
- [61] Polat F. and Guvenir H. A., "A Unification-Based Approach for Knowledge Base Verification," *Expert Systems*, Vol. 8, No. 4, November 1991.
- [62] Polat F. and Guvenir H. A., "Coordination Issues in Distributed Problem Solving," in *Proceedings of the Sixth International Symposium on Computer and Information Sciences*, M. Baray and B. Ozguc eds., Elsevier, Vol. 1, Antalya, Turkey, 1991, pp. 585-594.
- [63] Polat F. and Guvenir H. A., "A Conflict Resolution-based Cooperative Distributed Problem Solving Model," in *Proceedings of the AAAI-92 Workshop Cooperation Among Heterogeneous Intelligent Agents*, July 1992, San Jose, CA, USA, pp. 106-115.
- [64] Polat F. and Guvenir H. A., "A Conflict Resolution Based Decentralized Multi-Agent Problem Solving Model," in *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds*

- (MAAMAW-92), Rome Italy, 1992. A longer version of this paper will be appearing in *Lecture Notes on Artificial Intelligence* to be published by Springer-Verlag.
- [65] Polat F. and Guvenir H. A., "Distributed Artificial Intelligence in Engineering Design" in *Proceedings of the Engineering Design and Analysis Conference (ESDA)*, PD-Vol. 47-7, ASME, 1992, pp. 139-142.
- [66] Polat F. and Guvenir H. A., "UVT: Unification Based Tool for Knowledge Base Verification," *IEEE Expert*, Vol. 8, No. 3, June 1993, pp. 69-75. An earlier version has appeared in *Proceedings of the European Workshop on the Verification and Validation of Knowledge Based Systems*, Cambridge, England, 1991, pp. 147-163.
- [67] Polat F., Shekhar S., and Guvenir H. A., "Distributed Conflict Resolution Among Cooperating Expert Systems," *Expert Systems*, Vol. 10, No. 4, 1993, pp. 227-236.
- [68] Polat F., Shekhar S., and Guvenir H. A., "A Negotiation Platform for Cooperating Multi-Agent Systems," *International Journal of Concurrent Engineering: Research & Applications* (to appear)
- [69] Polat F. and Shekhar S., "A Negotiation Platform for Cooperating Intelligent Systems," *Technical Report*, TR-93-38, University of Minnesota, Computer Science Department, May 1993.
- [70] Preece A. D., Shinghal R., and Batarekh A., "Principles and practices in Verifying Rule-Based Systems," *Knowledge Engineering Review*, Vol. 7, No. 2, 1992, pp. 115-141.
- [71] Pruitt D. G., *Negotiation Behavior*, Academic Press, 1981.
- [72] Rosenschein J. S., and Genesereth M. R., "Deals Among Rational Agents," in *Proceedings of International Joint Conference on Artificial Intelligence*, 1985, pp. 91-99.

- [73] Rosenschein J. S., "Rational Interaction: Cooperation Among Intelligent Agents," *Ph.D. Thesis*, Stanford University, CA, 1986.
- [74] Sathi A., Morton T.E., and Roth S.F., "Castillo: An Intelligent Project Management System," *AI Magazine*, Vol. 7, No. 5, 1986, Winter 1986.
- [75] Shekhar S. and Ramamoorthy C. V., "Coop: A Self-Assessment Based Approach to Cooperating Expert Systems," *International Journal on Artificial Intelligence Tools*, Vol. 1, No. 2, 1992, pp. 175-204.
- [76] Shoham Y., "Agent Oriented Programming," *Technical Report*, STAN-CS-1335-90, Dept. of Computer Science, Stanford University, 1990.
- [77] Shoham Y., "Agent Oriented Programming," *Artificial Intelligence*, 1992.
- [78] Shoham Y., "Micro and Macro Theories of Artificial Agents," in *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW)*, Rome, Italy, 1992.
- [79] Shoham Y., "On the Synthesis of Useful Social Laws for Artificial Agents Societies," in *Proceedings of AAAI*, 1992, pp. 276-281.
- [80] Singh M. P., "Social and Psychological Commitments in Multi-Agent Systems," in *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, November 1991.
- [81] Singh M. P., "Towards a Formal Theory of Communication for Multi-Agent Systems," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1991, pp. 69-74.
- [82] Singh M. P., "On the Semantics of Protocols Among Distributed Intelligent Agents," in *Proceedings of the IEEE International Phoenix Conference on Computers and Communications*, 1992.
- [83] Smith R. G. and Davis R., "Frameworks for Cooperation in Distributed Problem Solving," *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser eds., Morgan Kaufmann, 1988, pp. 61-70.

- [84] Smith R. G. and Davis R., "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence*, Vol. 20, 1983, pp. 63-109.
- [85] Smithers T. and Troxell W., "Design is Intelligent Behavior But What's the Formalism," *AI EDAM*, Vol. 4, No. 2, 1990, pp. 89-98.
- [86] Sriram D. and Adey R., *Applications of Artificial Intelligence in Engineering Problems*, Vols. 1-2, Springer-Verlag, 1986.
- [87] Steeb R., et al., "Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control," *Technical Report*, R-3408-ARPA, Rand Corp., 1986.
- [88] Stefik M. and Bobrow D. G., "Object-Oriented Programming: Themes and Variations," *AI Magazine*, 1986, pp. 40-62.
- [89] Suwa M., Scott A. C., and Shortliffe E. H., "Completeness and Consistency in a Rule-Based System," *AI Magazine*, Vol. 3, 1982, pp. 16-21.
- [90] Sycara K., "Resolving Goal Conflicts via Negotiation," in *Proc. of the National Conference on Artificial Intelligence*, Minneapolis, MN, Aug. 1988, pp. 245-250.
- [91] Sycara K., "Augmentation: Planning Other Agents' Plans," in *Proceedings of 11th International Joint Conference on Artificial Intelligence*, 1989, pp. 517-523.
- [92] Sycara K., "Negotiation in Design," In it Proceedings of the MIT-JSME Workshop on Cooperative Product Development, MIT, Cambridge, MA, 1989.
- [93] Tan M. and Weihmayer R., "Integrating Agent-Oriented Programming and Planning for Cooperative Problem Solving," in *Proceedings of the AAAI-92 Workshop Cooperation Among Heterogeneous Intelligent Agents*, July 1992, San Jose, CA, USA. pp. 129-137.

- [94] Tokoro M. and Ishikawa Y., "An Object-Oriented Approach to Knowledge Systems," in *Proceedings of the Int. Conf. on Fifth Generation Computer Systems*, 1984, pp. 623-631.
- [95] Tomiyama T., Kiriyama T., and Yoshikawa H., "Infrastructure for Intelligent CAD," in *Proceedings of the Engineering Design and Analysis Conference (ESDA)*, PD-Vol. 47-7, ASME, 1992, pp. 131-138.
- [96] Veerkamp P., "On the Usage of Meta-Knowledge for Reasoning about Design Processes," in *Proceedings of the Engineering Design and Analysis Conference (ESDA)*, PD-Vol. 47-7, ASME, 1992, pp. 151-157.
- [97] Velthuijsen H. and Griffith N. D., "Negotiation in Telecommunication Systems," in *Proceedings of the AAAI-92 Workshop Cooperation Among Heterogeneous Intelligent Agents*, July 1992, San Jose, CA, USA. pp. 138-147.
- [98] Weihmayer R. and Brandau R., "Cooperative Distributed Problem Solving for Communication Network Management," *Computer Communications*, Vol. 13, No. 9, November 1990, pp. 547-557.
- [99] Werkman K., et al., "Design and Fabrication Problem Solving Through Cooperative Agents," NSF-ERC-ATLSS *Technical Report*, 90-05, Lehigh University, Bethlehem, 100.
- [100] Werner E., et al., "Planned Team Activity," in *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW)*, Rome, Italy, 1992.
- [101] Winston P. H., *Artificial Intelligence*, Addison-Wesley, 1984.
- [102] Zlotkin G. and Rosenschein J. S., "Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1989, pp. 912-917.
- [103] Zlotkin G. and Rosenschein J. S., "Negotiation and Conflict Resolution in Non-Cooperative Domains," in *Proceedings of AAAI*, 1990, pp. 100-105.

- [104] Zlotkin G. and Rosenschein J. S., "Incomplete Information and Deception in Multi-Agent Negotiation," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1991, pp. 225-231.

Appendix A

A Sample Run

NEPTUNE is implemented on a network of workstations running under the UNIX operating system where each agent is modeled as a process running on a workstation. Each agent has a communication procedure written in C programming language which communicates with a server process, again written in C, keeping the shared medium. All agents are connected to the server through a stream socket in INET¹ domain. No messages are lost and the modules are designed to ensure reliable delivery of messages of any length (messages are partitioned into blocks of 1Kbytes and packed later). Each agent is implemented in SB-Prolog programming language. The abstract data types used in communication are represented in Prolog. It is the responsibility of the knowledge engineer to encode the knowledge of each agent (control, domain and conflict resolution). Agents can be distributed over geographically distant sites. Upon getting the internet address, or IP number (Internet Port number) of the server, a new agent can easily enter the problem-solving environment from a site on internet.

In the rest of the appendix, a segment of an example run is given to illustrate how agents interact in detecting and resolving a particular conflict in designing an office. The example consists of the messages exchanged between the agents and the server process. A message sent from a particular agent to the server forms a request to be accomplished by the server. After performing the request, the

¹INET stands for the InterNET domain, a wide-area network consisting of hundreds of sites.

server process sends back its reply to that particular agent. A request messages is represented as a list of two elements. The first elements denotes the message owner. The second element represents the message body which is also a list of two elements. The first element of the message body represents the request type, while the second element denotes the argument(s) passed. The second element can be an atom, or a list of atoms depending on the number of arguments to be passed. If no argument is to be passed, this field filled in with a variable name. Note that SB-Prolog represents a variable name as unique number preceded by an underscore character. The response of the server process can be represented as an atom, or as a list depending on the type of the request.

SB-Prolog Version 3.1

| ?- yes

| ?- request :

```
[agent1,
 [assert_problem_tuple,
  [[tasks([t1,t2,t3,t4,t5,t6,t7,t8,t9,t20,t21,t22,t40,t41]),
   task(t41,p_pc_locy,plocationy), task(t40,p_pc_locx,plocationx),
   task(t22,p_c_lampz,lcost), task(t21,p_c_lampy,lcost),
   task(t20,p_c_lampx,lcost), task(t9,p_lampz,lamps)
   task(t8,p_lampy,lamps), task(t7,p_lampx,lamps)
   task(t6,p_pcdesk_locy,locationy), task(t5,p_pcdesk_locx,locationx)
   task(t4,p_pcdesk,pcdesks), task(t3,p_desk_locy,locationy)
   task(t2,p_desk_locx,locationx), task(t1,p_desk,desks)],
 [po(t7,t22), po(t7,t21), po(t7,t20), po(t7,t9), po(t7,t8),
  po(t6,t7), po(t4,t6), po(t4,t5), po(t1,t3), po(t1,t2)],
 [gconstraints([c20,c21]),
  constraint(c21,[p1,p2],[],_1395456),
  constraint(c20,[p1],[],_1395624)],
 [layout([locationx,locationy,room1,door1>window1,
          eplug1,pplug1,lplug1,lplug2,lplug3]),
  value(p_c_lampz,95), value(p_c_lampy,95),
  value(p_c_lampx,95), value(p_lampz,lamp03),
  value(p_lampy,lamp02), value(p_lampx,lamp01),
  value(p_pc_locy,3), value(p_pc_locx,3),
  value(p_lampz,lamp03), value(p_lampy,lamp02),
```

```
value(p_lampx,lamp01),value(p_pcdesk_locy,3.5),
value(p_pcdesk_locx,2.3),value(p_pcdesk,pcdesk0),
value(p_desk_locy,0.5),value(p_desk_locx,0.75),
value(p_desk,desk0),
attribute(lplug3,cornery,numeric,2),
attribute(lplug3,cornerx,numeric,6),
attribute(lplug2,cornery,numeric,2),
attribute(lplug2,cornerx,numeric,4),
attribute(lplug1,cornery,numeric,2),
attribute(lplug1,cornerx,numeric,2),
attribute(pplug1,cornery,numeric,0.5),
attribute(pplug1,cornerx,numeric,6),
attribute(eplug1,cornery,numeric,0),
attribute(eplug1,cornerx,numeric,6),
attribute(window1,nparts,numeric,3),
attribute(window1,frame,symbolic,wood),
attribute(window1,height,numeric,1),
attribute(window1,width,numeric,0.1),
attribute(window1,length,numeric,2),
attribute(window1,cornery,numeric,3),
attribute(window1,cornerx,numeric,0),
attribute(window1,shape,symbolic,rectangular),
attribute(door1,made,symbolic,wood),
attribute(door1,height,numeric,2),
attribute(door1,width,numeric,0.2),
attribute(door1,length,numeric,1),
attribute(door1,cornery,numeric,7),
attribute(door1,cornerx,numeric,8),
attribute(door1,shape,symbolic,rectangular),
attribute(room1,lplugc,complex,lplug3),
attribute(room1,lplugb,complex,lplug2),
attribute(room1,lpluga,complex,lplug1),
attribute(room1,pplug,complex,pplug1),
attribute(room1,eplug,complex,eplug1),
attribute(room1>window,complex>window1),
attribute(room1>door,complex>door1),
attribute(room1,height,numeric,2.5),
```

```

    attribute(room1,length,numeric,8),
    attribute(room1,width,numeric,10),
    attribute(room1,shape,symbolic,rectangular),
    object(lplug3,eplug),
    object(lplug2,eplug),
    object(lplug1,eplug),
    object(pplug1,eplug),
    object(eplug1,eplug),
    object(window1>window),
    object(door1,doors),
    object(room1,rooms),
    attributes(lplug3,[cornerx,cornery]),
    attributes(lplug2,[cornerx,cornery]),
    attributes(lplug1,[cornerx,cornery]),
    attributes(pplug1,[cornerx,cornery]),
    attributes(eplug1,[cornerx,cornery]),
    attributes(window1,[shape,cornerx,cornery,length,
        width,height,frame,nparts]),
    attributes(door1,[shape,cornerx,cornery,length,width,height,made]),
    attributes(room1,[shape,length,width,height,door,
        window,eplug,pplug,lpluga,lplugb,lplugc])]]]]
reply : okay
request : [agent2,[get_problem_tuple,_1392488]]
reply :
    [agent1,
    [tasks([t1,t2,t3,t4,t5,t6,t7,t8,t9,t20,t21,t22,t40,t41]),
    task(t41,p_pc_locy,plocationy), task(t40,p_pc_locx,plocationx),
    task(t22,p_c_lampz,lcost), task(t21,p_c_lampy,lcost),
    task(t20,p_c_lampx,lcost), task(t9,p_lampz,lamps)
    task(t8,p_lampy,lamps), task(t7,p_lampx,lamps)
    task(t6,p_pcdesk_locy,locationy), task(t5,p_pcdesk_locx,locationx)
    task(t4,p_pcdesk_pcdesks), task(t3,p_desk_locy,locationy)
    task(t2,p_desk_locx,locationx), task(t1,p_desk,desks)],
    [po(t7,t22), po(t7,t21), po(t7,t20), po(t7,t9), po(t7,t8),
    po(t6,t7), po(t4,t6), po(t4,t5), po(t1,t3), po(t1,t2)],
    [gconstraints([c20,c21]),
    constraint(c21,[p1,p2],[],_1395456),

```

```

constraint(c20,[p1],[],_1395624)],
[layout([locationx,locationy,room1,door1>window1,
        eplug1,pplug1,lplug1,lplug2,lplug3]),
value(p_c_lampz,95),value(p_c_lampy,96),
value(p_c_lampx,95),value(p_lampz,lamp03),
value(p_lampy,lamp02),value(p_lampx,lamp01),
value(p_pc_locy,3),value(p_pc_locx,3),
value(p_lampz,lamp03),value(p_lampy,lamp02),
value(p_lampx,lamp01),value(p_pcdesk_locy,3.5),
value(p_pcdesk_locx,2.3),value(p_pcdesk,pcdesk0),
value(p_desk_locy,0.5),value(p_desk_locx,0.75),
value(p_desk,desk0),
attribute(lplug3,cornery,numeric,2),
attribute(lplug3,cornerx,numeric,6),
attribute(lplug2,cornery,numeric,2),
attribute(lplug2,cornerx,numeric,4),
attribute(lplug1,cornery,numeric,2),
attribute(lplug1,cornerx,numeric,2),
attribute(pplug1,cornery,numeric,0.5),
attribute(pplug1,cornerx,numeric,6),
attribute(eplug1,cornery,numeric,0),
attribute(eplug1,cornerx,numeric,6),
attribute(window1,nparts,numeric,3),
attribute(window1,frame,symbolic,wood),
attribute(window1,height,numeric,1),
attribute(window1,width,numeric,0.1),
attribute(window1,length,numeric,2),
attribute(window1,cornery,numeric,3),
attribute(window1,cornerx,numeric,0),
attribute(window1,shape,symbolic,rectangular),
attribute(door1,made,symbolic,wood),
attribute(door1,height,numeric,2),
attribute(door1,width,numeric,0.2),
attribute(door1,length,numeric,1),
attribute(door1,cornery,numeric,7),
attribute(door1,cornerx,numeric,8),
attribute(door1,shape,symbolic,rectangular),

```

```

attribute(room1,lplugc,complex,lplug3),
attribute(room1,lplugb,complex,lplug2),
attribute(room1,lpluga,complex,lplug1),
attribute(room1,pplug,complex,pplug1),
attribute(room1,eplug,complex,eplug1),
attribute(room1>window,complex>window1),
attribute(room1,door,complex>door1),
attribute(room1,height,numeric,2.5),
attribute(room1,length,numeric,8),
attribute(room1,width,numeric,10),
attribute(room1,shape,symbolic,rectangular),
object(lplug3,eplug),
object(lplug2,eplug),
object(lplug1,eplug),
object(pplug1,eplug),
object(eplug1,eplug),
object>window1>window),
object>door1>doors),
object>room1>rooms),
attributes(lplug3,[cornerx,cornery]),
attributes(lplug2,[cornerx,cornery]),
attributes(lplug1,[cornerx,cornery]),
attributes(pplug1,[cornerx,cornery]),
attributes(eplug1,[cornerx,cornery]),
attributes>window1,[shape,cornerx,cornery,length,
width,height,frame,nparts]),
attributes>door1,[shape,cornerx,cornery,length,width,height,made]),
attributes>room1,[shape,length,width,height>door,
window>eplug>pplug>lpluga>lplugb>lplugc]]]
request : [agent3,[get_problem_tuple,_1392488]]
reply :
[agent1,
[tasks([t1,t2,t3,t4,t5,t6,t7,t8,t9,t20,t21,t22,t40,t41]),
task(t41,p_pc_locy,plocationy), task(t40,p_pc_locx,plocationx),
task(t22,p_c_lampz,lcost),task(t21,p_c_lampy,lcost),
task(t20,p_c_lampx,lcost),task(t9,p_lampz,lamps)
task(t8,p_lampy,lamps),task(t7,p_lampx,lamps)

```

```

    task(t6,p_pcdesk_locy,locationy),task(t5,p_pcdesk_locx,locationx)
    task(t4,p_pcdesk,pcdesks),task(t3,p_desk_locy,locationy)
    task(t2,p_desk_locx,locationx),task(t1,p_desk,desks)],
    [po(t7,t22), po(t7,t21), po(t7,t20), po(t7,t9), po(t7,t8),
    po(t6,t7), po(t4,t6), po(t4,t5), po(t1,t3), po(t1,t2)],
    [gconstraints([c20,c21]),
    constraint(c21,[p1,p2],[],_1395456),
    constraint(c20,[p1],[],_1395624)],
    [layout([locationx,locationy,room1,door1>window1,
            eplug1,pplug1,lplug1,lplug2,lplug3]),
    value(p_c_lampz,95),value(p_c_lampy,95),
    value(p_c_lampx,95),value(p_lampz,lamp03),
    value(p_lampy,lamp02),value(p_lampx,lamp01),
    value(p_pc_locy,3),value(p_pc_locx,3),
    value(p_lampz,lamp03),value(p_lampy,lamp02),
    value(p_lampx,lamp01),value(p_pcdesk_locy,3.5),
    value(p_pcdesk_locx,2.3),value(p_pcdesk,pcdesk0),
    value(p_desk_locy,0.5),value(p_desk_locx,0.75),
    value(p_desk,desk0),
    attribute(lplug3,cornery,numeric,2),
    attribute(lplug3,cornerx,numeric,6),
    attribute(lplug2,cornery,numeric,2),
    attribute(lplug2,cornerx,numeric,4),
    attribute(lplug1,cornery,numeric,2),
    attribute(lplug1,cornerx,numeric,2),
    attribute(pplug1,cornery,numeric,0.5),
    attribute(pplug1,cornerx,numeric,6),
    attribute(eplug1,cornery,numeric,0),
    attribute(eplug1,cornerx,numeric,6),
    attribute(window1,nparts,numeric,3),
    attribute(window1,frame,symbolic,wood),
    attribute(window1,height,numeric,1),
    attribute(window1,width,numeric,0.1),
    attribute(window1,length,numeric,2),
    attribute(window1,cornery,numeric,3),
    attribute(window1,cornerx,numeric,0),
    attribute(window1,shape,symbolic,rectangular),

```

```

    attribute(door1,made,symbolic,wood),
    attribute(door1,height,numeric,2),
    attribute(door1,width,numeric,0.2),
    attribute(door1,length,numeric,1),
    attribute(door1,cornery,numeric,7),
    attribute(door1,cornerx,numeric,8),
    attribute(door1,shape,symbolic,rectangular),
    attribute(room1,lplugc,complex,lplug3),
    attribute(room1,lplugb,complex,lplug2),
    attribute(room1,lpluga,complex,lplug1),
    attribute(room1,pplug,complex,pplug1),
    attribute(room1,eplug,complex,eplug1),
    attribute(room1>window,complex>window1),
    attribute(room1>door,complex>door1),
    attribute(room1,height,numeric,2.5),
    attribute(room1,length,numeric,8),
    attribute(room1,width,numeric,10),
    attribute(room1,shape,symbolic,rectangular),
    object(lplug3,eplug),
    object(lplug2,eplug),
    object(lplug1,eplug),
    object(pplug1,eplug),
    object(eplug1,eplug),
    object>window1>window),
    object>door1>doors),
    object>room1>rooms),
    attributes(lplug3,[cornerx,cornery]),
    attributes(lplug2,[cornerx,cornery]),
    attributes(lplug1,[cornerx,cornery]),
    attributes(pplug1,[cornerx,cornery]),
    attributes(eplug1,[cornerx,cornery]),
    attributes>window1,[shape,cornerx,cornery,length,
        width,height,frame,nparts]),
    attributes>door1,[shape,cornerx,cornery,length,width,height,made]),
    attributes>room1,[shape,length,width,height>door,
        window>eplug>pplug>lpluga>lplugb>lplugc]]]
request : [agent4,[get_problem_tuple,_1392488]]

```


reply :

```
[agent1,
  [tasks([t1,t2,t3,t4,t5,t6,t7,t8,t9,t20,t21,t22,t40,t41]),
    task(t41,p_pc_locy,plocationy), task(t40,p_pc_locx,plocationx),
    task(t22,p_c_lampz,lcost),task(t21,p_c_lampy,lcost),
    task(t20,p_c_lampx,lcost),task(t9,p_lampz,lamps)
    task(t8,p_lampy,lamps),task(t7,p_lampx,lamps)
    task(t6,p_pcdesk_locy,locationy),task(t5,p_pcdesk_locx,locationx)
    task(t4,p_pcdesk,pcdesks),task(t3,p_desk_locy,locationy)
    task(t2,p_desk_locx,locationx),task(t1,p_desk,desks)],
  [po(t7,t22), po(t7,t21), po(t7,t20), po(t7,t9), po(t7,t8),
    po(t6,t7), po(t4,t6), po(t4,t5), po(t1,t3), po(t1,t2)],
  [gconstraints([c20,c21]),
    constraint(c21,[p1,p2],[],_1395456),
    constraint(c20,[p1],[],_1395624)],
  [layout([locationx,locationy,room1,door1>window1,
          eplug1,pplug1,lplug1,lplug2,lplug3]),
    value(p_c_lampz,95),value(p_c_lampy,95),
    value(p_c_lampx,95),value(p_lampz,lamp03),
    value(p_lampy,lamp02),value(p_lampx,lamp01),
    value(p_pc_locy,3),value(p_pc_locx,3),
    value(p_lampz,lamp03),value(p_lampy,lamp02),
    value(p_lampx,lamp01),value(p_pcdesk_locy,3.5),
    value(p_pcdesk_locx,2.3),value(p_pcdesk,pcdesk0),
    value(p_desk_locy,0.5),value(p_desk_locx,0.75),
    value(p_desk,desk0),
    attribute(lplug3,cornery,numeric,2),
    attribute(lplug3,cornerx,numeric,6),
    attribute(lplug2,cornery,numeric,2),
    attribute(lplug2,cornerx,numeric,4),
    attribute(lplug1,cornery,numeric,2),
    attribute(lplug1,cornerx,numeric,2),
    attribute(pplug1,cornery,numeric,0.5),
    attribute(pplug1,cornerx,numeric,6),
    attribute(eplug1,cornery,numeric,0),
    attribute(eplug1,cornerx,numeric,6),
    attribute(window1,nparts,numeric,3),
```

```
attribute(window1,frame,symbolic,wood),
attribute(window1,height,numeric,1),
attribute(window1,width,numeric,0.1),
attribute(window1,length,numeric,2),
attribute(window1,cornery,numeric,3),
attribute(window1,cornerx,numeric,0),
attribute(window1,shape,symbolic,rectangular),
attribute(door1,made,symbolic,wood),
attribute(door1,height,numeric,2),
attribute(door1,width,numeric,0.2),
attribute(door1,length,numeric,1),
attribute(door1,cornery,numeric,7),
attribute(door1,cornerx,numeric,8),
attribute(door1,shape,symbolic,rectangular),
attribute(room1,lplugc,complex,lplug3),
attribute(room1,lplugb,complex,lplug2),
attribute(room1,lpluga,complex,lplug1),
attribute(room1,pplug,complex,pplug1),
attribute(room1,eplug,complex,eplug1),
attribute(room1>window,complex>window1),
attribute(room1>door,complex>door1),
attribute(room1,height,numeric,2.5),
attribute(room1,length,numeric,8),
attribute(room1,width,numeric,10),
attribute(room1,shape,symbolic,rectangular),
object(lplug3,eplug),
object(lplug2,eplug),
object(lplug1,eplug),
object(pplug1,eplug),
object(eplug1,eplug),
object>window1>window),
object>door1>doors),
object>room1>rooms),
attributes(lplug3,[cornerx,cornery]),
attributes(lplug2,[cornerx,cornery]),
attributes(lplug1,[cornerx,cornery]),
attributes(pplug1,[cornerx,cornery]),
```

```
attributes(eplug1,[cornerx,cornery]),
attributes(window1,[shape,cornerx,cornery,length,
width,height,frame,nparts]),
attributes(door1,[shape,cornerx,cornery,length,width,height,made]),
attributes(room1,[shape,length,width,height,door,
window,eplug,pplug,lpluga,lplugb,lplugc]]])
request : [agent1,[signal_quit,_1392488]]
reply : []
request : [agent1,[quitted,_1392488]]
reply : []
request : [agent1,[task_status,t1]]
reply : waiting
request : [agent1,[quitted,_1392488]]
reply : []
request : [agent1,[proposal_asserted,_1392488]]
reply : []
request : [agent1,[task_status,t1]]
reply : waiting
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent1,[quitted,_1392488]]
reply : []
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent4,[signal_quit,_1392488]]
reply : []
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent4,[quitted,_1392488]]
reply : []
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent4,[task_status,t1]]
reply : waiting
request : [agent1,[proposal_asserted,_1392488]]
```

```
reply : []
request : [agent4,[quitted,_1392488]]
reply : []
request : [agent1,[task_status,t1]]
reply : waiting
request : [agent4,[proposal_asserted,_1392488]]
reply : []
request : [agent3,[quitted,_1392488]]
reply : []
request : [agent4,[task_status,t1]]
reply : waiting
request : [agent3,[task_status,t4]]
reply : waiting
request : [agent1,[quitted,_1392488]]
reply : []
request : [agent3,[quitted,_1392488]]
reply : []
request : [agent1,[proposal_asserted,_1392488]]
reply : []
request : [agent3,[proposal_asserted,_1392488]]
reply : []
request : [agent1,[task_status,t1]]
reply : waiting
request : [agent3,[task_status,t4]]
reply : waiting
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent4,[quitted,_1392488]]
reply : []
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent4,[proposal_asserted,_1392488]]
reply : []
request : [agent2,[quitted,_1392488]]
reply : []
request : [agent4,[task_status,t1]]
reply : waiting
```

```
request : [agent2,[quitted,_1392488]]
reply   : []
request : [agent1,[quitted,_1392488]]
reply   : []
request : [agent3,[quitted,_1392488]]
reply   : []
request : [agent1,[proposal_asserted,_1392488]]
reply   : []
request : [agent3,[proposal_asserted,_1392488]]
reply   : []
request : [agent1,[task_status,t1]]
reply   : waiting
request : [agent3,[task_status,t4]]
reply   : waiting
request : [agent4,[quitted,_1392488]]
reply   : []
request : [agent2,[quitted,_1392488]]
reply   : []
request : [agent1,[quitted,_1392488]]
reply   : []
request : [agent3,[quitted,_1392488]]
reply   : []
request : [agent1,[proposal_asserted,_1392488]]
reply   : []
request : [agent3,[proposal_asserted,_1392488]]
reply   : []
request : [agent1,[task_status,t1]]
reply   : waiting
request : [agent3,[task_status,t4]]
reply   : waiting
request : [agent4,[proposal_asserted,_1392488]]
reply   : []
request : [agent4,[task_status,t1]]
reply   : waiting
request : [agent1,[quitted,_1392488]]
reply   : []
request : [agent3,[quitted,_1392488]]
```

```
reply : []
request : [agent1,[proposal_asserted,_1392488]]
reply : []
request : [agent3,[proposal_asserted,_1392488]]
reply : []
request : [agent1,[task_status,t1]]
reply : waiting
request : [agent3,[task_status,t4]]
reply : waiting
request : [agent4,[quitted,_1392488]]
reply : []
request : [agent4,[proposal_asserted,_1392488]]
reply : []
request : [agent1,[quitted,_1392488]]
reply : []
request : [agent3,[quitted,_1392488]]
reply : []
request : [agent1,[proposal_asserted,_1392488]]
reply : []
request : [agent3,[proposal_asserted,_1392488]]
reply : []
request : [agent1,[task_status,t1]]
reply : waiting
request : [agent3,[task_status,t4]]
reply : waiting
request : [agent4,[task_status,t1]]
reply : waiting
request : [agent1,[quitted,_1392488]]
reply : []
request : [agent3,[quitted,_1392488]]
reply : []
request : [agent4,[quitted,_1392488]]
reply : []
request : [agent3,[proposal_asserted,_1392488]]
reply : []
request : [agent4,[proposal_asserted,_1392488]]
reply : []
```

```
request : [agent3,[task_status,t4]]
reply : waiting
request : [agent4,[task_status,t1]]
reply : waiting
request : [agent2,[get_unique_id,_1392488]]
reply : 0
request : [agent1,[proposal_asserted,_1392488]]
reply : []
request : [agent2,
          [assert_proposal,
           [0,agent2,[t1,t2,t3,t4,t5,t6],
            [assign(p_desk,desk1),
             assign(p_desk_locx,0.75),
             assign(p_desk_locy,0.5),
             assign(p_pcdesk,pcdesk1),
             assign(p_pcdesk_locx,0.75),
             assign(p_pcdesk_locy,2.75)],
            [add(relation(on(desk1,layout))),
             add(relation(on(pcdesk1,layout))),
             add(relation(location(desk1,0.75,0.5))),
             add(relation(location(pcdesk1,0.75,2.75)))]]]]]
reply : ok
request : [agent1,[task_status,t1]]
reply : waiting
request : [agent3,[quitted,_1392488]]
reply : []
request : [agent3,[proposal_asserted,_1392488]]
reply : ok
request : [agent2,
          [assert_evaluation_result,
           [0,agent2,[0.114286,0.0,0.0,0.136842,0.0404348,0.0128571],
            0.30442,nonconflicting]]]
reply : _1392288
request : [agent1,[quitted,_1392488]]
reply : []
request : [agent2,[evaluation_finished,0]]
reply : []
```

```

request : [agent1,[proposal_asserted,_1392488]]
reply : ok
request : [agent3,[get_current_proposal,_1392488]]
reply : [0,agent2,[t1,t2,t3,t4,t5,t6],
        [assign(p_desk,desk1),
         assign(p_desk_locx,0.75),
         assign(p_desk_locy,0.5),
         assign(p_pcdesk,pcdesk1),
         assign(p_pcdesk_locx,0.75),
         assign(p_pcdesk_locy,2.75)],
        [add(relation(on(desk1,layout))),
         add(relation(on(pcdesk1,layout))),
         add(relation(location(desk1,0.75,0.5))),
         add(relation(location(pcdesk1,0.75,2.75)))]]
request : [agent1,[get_current_proposal,_1392488]]
reply : [0,agent2,[t1,t2,t3,t4,t5,t6],
        [assign(p_desk,desk1),
         assign(p_desk_locx,0.75),
         assign(p_desk_locy,0.5),
         assign(p_pcdesk,pcdesk1),
         assign(p_pcdesk_locx,0.75),
         assign(p_pcdesk_locy,2.75)],
        [add(relation(on(desk1,layout))),
         add(relation(on(pcdesk1,layout))),
         add(relation(location(desk1,0.75,0.5))),
         add(relation(location(pcdesk1,0.75,2.75)))]]
request : [agent3,[assert_evaluation_result,
                 [0,agent3,[0.0,0.0,0.0,0.0,-0.145431,-0.145431],
                 -0.290862,conflicting]]]
reply : _1392288
request : [agent2,[evaluation_finished,0]]
reply : []
request : [agent3,[evaluation_finished,0]]
reply : []
request : [agent4,[quitted,_1392488]]
reply : []
request : [agent1,[assert_evaluation_result,

```



```

                [0,agent1,[0.0,0.0,0.0,0.0,-0.168478,-0.0535714],
                -0.22205,conflicting]]]
reply : _1392288
request : [agent4,[proposal_asserted,_1392488]]
reply : ok
request : [agent1,[evaluation_finished,0]]
reply : []
request : [agent4,[get_current_proposal,_1392488]]
reply : [0,agent2,[t1,t2,t3,t4,t5,t6],
        [assign(p_desk,desk1),
         assign(p_desk_locx,0.75),
         assign(p_desk_locy,0.5),
         assign(p_pcdesk,pcdesk1),
         assign(p_pcdesk_locx,0.75),
         assign(p_pcdesk_locy,2.75)],
        [add(relation(on(desk1,layout))),
         add(relation(on(pcdesk1,layout))),
         add(relation(location(desk1,0.75,0.5))),
         add(relation(location(pcdesk1,0.75,2.75)))]]
request : [agent2,[evaluation_finished,0]]
reply : []
request : [agent1,[evaluation_finished,0]]
reply : []
request : [agent4,[assert_evaluation_result,
                [0,agent4,[0.0,0.0,0.0,0.0,0.0,0.0],0.0,nonconflicting]]]
reply : _1392288
request : [agent3,[evaluation_finished,0]]
reply : ok
request : [agent4,[assert_conflict_resolution,[0,agent4,nothing,[]]]]
reply : _1392288
request : [agent3,[assert_conflict_resolution,
                [0,agent3,counterproposing,[t1,t2,t3,t4,t5,t6]]]]]
reply : _1392288
request : [agent4,[current_handled,0]]
reply : []
request : [agent3,[all_constraints_asserted,0]]
reply : []

```

```
request : [agent2,[evaluation_finished,0]]
reply : ok
request : [agent2,[conflict_or_not,0]]
reply : conflicting
request : [agent3,[all_constraints_asserted,0]]
reply : []
request : [agent2,[assert_conflict_resolution,
                [0,agent2,counterproposing,[t1,t2,t3,t4,t5,t6]]]]
reply : _1392288
request : [agent4,[current_handled,0]]
reply : []
request : [agent3,[all_constraints_asserted,0]]
reply : []
request : [agent1,[evaluation_finished,0]]
reply : ok
request : [agent2,[all_constraints_asserted,0]]
reply : []
request : [agent4,[current_handled,0]]
reply : []
request : [agent1,[assert_conflict_resolution,[0,agent1,constraining,[c6]]]]
reply : _1392288
request : [agent2,[all_constraints_asserted,0]]
reply : ok
request : [agent1,[candidates_proposed,0]]
reply : []
request : [agent4,[current_handled,0]]
reply : []
request : [agent3,[all_constraints_asserted,0]]
reply : ok
request : [agent3,[retrieve_constraints,0]]
reply : [[]]
request : [agent2,[retrieve_constraints,0]]
reply : [[]]
request : [agent4,[current_handled,0]]
reply : []
request : [agent2,[get_unique_id,_1392488]]
reply : 1
```

```

request : [agent1,[candidates_proposed,0]]
reply : []
request : [agent2,
          [assert_candidate_proposal,
           [1,agent2,[t1,t2,t3,t4,t5,t6],
            [assign(p_desk,desk1),
             assign(p_desk_locx,1.25),
             assign(p_desk_locy,0.5),
             assign(p_pcdesk,pcdesk1),
             assign(p_pcdesk_locx,0.25),
             assign(p_pcdesk_locy,0.5)],
            [add(relation(on(desk1,layout))),
             add(relation(on(pcdesk1,layout))),
             add(relation(location(desk1,1.25,0.5))),
             add(relation(location(pcdesk1,0.25,0.5)))]]]]
reply : _1392288
request : [agent4,[current_handled,0]]
reply : []
request : [agent2,[candidates_proposed,0]]
reply : []
request : [agent3,[get_unique_id,_1392488]]
reply : 2
request : [agent3,
          [assert_candidate_proposal,
           [2,agent3,[t1,t2,t3,t4,t5,t6],
            [assign(p_desk,desk1),
             assign(p_desk_locx,0.75),
             assign(p_desk_locy,0.5),
             assign(p_pcdesk,pcdesk1),
             assign(p_pcdesk_locx,3.15),
             assign(p_pcdesk_locy,0.5)],
            [add(relation(on(desk1,layout))),
             add(relation(on(pcdesk1,layout))),
             add(relation(location(desk1,0.75,0.5))),
             add(relation(location(pcdesk,3.15,0.5)))]]]]
reply : _1392288
request : [agent2,[candidates_proposed,0]]

```

```

reply : ok
request : [agent3,[candidates_proposed,0]]
reply : ok
request : [agent2,[get_candidates,0]]
reply : [[1,agent2,[t1,t2,t3,t4,t5,t6],
          [assign(p_desk,desk1),
           assign(p_desk_locx,1.25),
           assign(p_desk_locy,0.5),
           assign(p_pcdesk,pcdesk1),
           assign(p_pcdesk_locx,0.25),
           assign(p_pcdesk_locy,0.5)],
          [add(relation(on(desk1,layout))),
           add(relation(on(pcdesk1,layout))),
           add(relation(location(desk1,1.25,0.5))),
           add(relation(location(pcdesk1,0.25,0.5)))]],
         [2,agent3,[t1,t2,t3,t4,t5,t6],
          [assign(p_desk,desk1),
           assign(p_desk_locx,0.75),
           assign(p_desk_locy,0.5),
           assign(p_pcdesk,pcdesk1),
           assign(p_pcdesk_locx,3.15),
           assign(p_pcdesk_locy,0.5)],
          [add(relation(on(desk1,layout))),
           add(relation(on(pcdesk1,layout))),
           add(relation(location(desk1,0.75,0.5))),
           add(relation(location(pcdesk,3.15,0.5)))]]]]
request : [agent3,[get_candidates,0]]
reply : [[1,agent2,[t1,t2,t3,t4,t5,t6],
          [assign(p_desk,desk1),
           assign(p_desk_locx,1.25),
           assign(p_desk_locy,0.5),
           assign(p_pcdesk,pcdesk1),
           assign(p_pcdesk_locx,0.25),
           assign(p_pcdesk_locy,0.5)],
          [add(relation(on(desk1,layout))),
           add(relation(on(pcdesk1,layout))),
           add(relation(location(desk1,1.25,0.5))),

```

```

        add(relation(location(pcdesk1,0.25,0.5))))],
    [2,agent3,[t1,t2,t3,t4,t5,t6],
      [assign(p_desk,desk1),
        assign(p_desk_locx,0.75),
        assign(p_desk_locy,0.5),
        assign(p_pcdesk,pcdesk1),
        assign(p_pcdesk_locx,3.15),
        assign(p_pcdesk_locy,0.5)],
      [add(relation(on(desk1,layout))),
        add(relation(on(pcdesk1,layout))),
        add(relation(location(desk1,0.75,0.5))),
        add(relation(location(pcdesk,3.15,0.5)))]])
request : [agent4,[current_handled,0]]
reply : []
request : [agent3,[assert_evaluation_result,
  [2,agent3,[0.0,0.0,0.0,0.0,-0.155952,-0.155952],
  -0.311905,conflicting]]]
reply : _1392288
request : [agent3,[signal_candidate_evaluated,0]]
reply : _1392288
request : [agent1,[candidates_proposed,0]]
reply : ok
request : [agent2,[assert_evaluation_result,
  [2,agent2,[0.114286,0.0,0.0,0.136842,-0.0161905,0.0514286],
  0.286366,nonconflicting]]]
reply : _1392288
request : [agent1,[get_candidates,0]]
reply : [[1,agent2,[t1,t2,t3,t4,t5,t6],
  [assign(p_desk,desk1),
    assign(p_desk_locx,1.25),
    assign(p_desk_locy,0.5),
    assign(p_pcdesk,pcdesk1),
    assign(p_pcdesk_locx,0.25),
    assign(p_pcdesk_locy,0.5)],
  [add(relation(on(desk1,layout))),
    add(relation(on(pcdesk1,layout))),
    add(relation(location(desk1,1.25,0.5))),

```

```

        add(relation(location(pcdesk1,0.25,0.5))))],
    [2,agent3,[t1,t2,t3,t4,t5,t6],
    [assign(p_desk,desk1),
    assign(p_desk_locx,0.75),
    assign(p_desk_locy,0.5),
    assign(p_pcdesk,pcdesk1),
    assign(p_pcdesk_locx,3.15),
    assign(p_pcdesk_locy,0.5)],
    [add(relation(on(desk1,layout))),
    add(relation(on(pcdesk1,layout))),
    add(relation(location(desk1,0.75,0.5))),
    add(relation(location(pcdesk,3.15,0.5)))]])
request : [agent2,[signal_candidate_evaluated,0]]
reply : _1392288
request : [agent3,[assert_evaluation_result,
    [1,agent3,[0.0,0.0,0.0,0.0,-0.307692,-0.307692],
    -0.615385,conflicting]]]
reply : _1392288
request : [agent4,[current_handled,0]]
reply : []
request : [agent3,[signal_candidate_evaluated,0]]
reply : _1392288
request : [agent1,[assert_evaluation_result,
    [2,agent1,[0.0,0.0,0.0,0.0,0.0674604,-0.214286],
    -0.146825,conflicting]]]
reply : _1392288
request : [agent2,[assert_evaluation_result,[1,agent2,
    [0.114286,-0.024,0.0,0.136842,0.0534783,0.0514286],0.332035,
    nonconflicting]]]
reply : _1392288
request : [agent1,[signal_candidate_evaluated,0]]
reply : _1392288
request : [agent2,[signal_candidate_evaluated,0]]
reply : _1392288
request : [agent3,[candidate_evaluation_finished,0]]
reply : []
request : [agent4,[current_handled,0]]

```

```

reply : []
request : [agent1,[assert_evaluation_result,
                [1,agent1,[0.0,0.0135135,0.0,0.0,-0.222826,-0.214286],-0.423598,
                conflicting]]]
reply : _1392288
request : [agent3,[candidate_evaluation_finished,0]]
reply : []
request : [agent1,[signal_candidate_evaluated,0]]
reply : _1392288
request : [agent4,[current_handled,0]]
reply : []
request : [agent1,[preferred_proposal_identified,0]]
reply : []
request : [agent3,[candidate_evaluation_finished,0]]
reply : ok
request : [agent2,[candidate_evaluation_finished,0]]
reply : ok
request : [agent4,[current_handled,0]]
reply : []
request : [agent1,[preferred_proposal_identified,0]]
reply : []
request : [agent3,[preferred_proposal_identified,0]]
reply : []
request : [agent2,[signal_choose_proposal,0]]
scorefor0-0.290862
scorefor2-0.311905
scorefor1-0.615385
reply : _1392288
request : [agent1,[preferred_proposal_identified,0]]
reply : ok
request : [agent2,[preferred_proposal_identified,0]]
reply : ok
request : [agent1,[retrieve_preferred_and_score,0]]
reply : [[0,agent2,[t1,t2,t3,t4,t5,t6],
                [assign(p_desk,desk1),
                assign(p_desk_locx,0.75),
                assign(p_desk_locy,0.5),

```

```

        assign(p_pcdesk,pcdesk1),
        assign(p_pcdesk_locx,0.75),
        assign(p_pcdesk_locy,2.75)],
        [add(relation(on(desk1,layout))),
        add(relation(on(pcdesk1,layout))),
        add(relation(location(desk1,0.75,0.5))),
        add(relation(location(pcdesk1,0.75,2.75)))]],-0.290862]
request : [agent3,[preferred_proposal_identified,0]]
reply  : ok
request : [agent4,[current_handled,0]]
reply  : ok
request : [agent1,[task_status,t1]]
reply  : agreed
request : [agent2,[retrieve_preferred_and_score,0]]
reply  : [[0,agent2,[t1,t2,t3,t4,t5,t6],
        [assign(p_desk,desk1),
        assign(p_desk_locx,0.75),
        assign(p_desk_locy,0.5),
        assign(p_pcdesk,pcdesk1),
        assign(p_pcdesk_locx,0.75),
        assign(p_pcdesk_locy,2.75)],
        [add(relation(on(desk1,layout))),
        add(relation(on(pcdesk1,layout))),
        add(relation(location(desk1,0.75,0.5))),
        add(relation(location(pcdesk1,0.75,2.75)))]],-0.290862]
request : [agent3,[retrieve_preferred_and_score,0]]
reply  : [[0,agent2,[t1,t2,t3,t4,t5,t6],
        [assign(p_desk,desk1),
        assign(p_desk_locx,0.75),
        assign(p_desk_locy,0.5),
        assign(p_pcdesk,pcdesk1),
        assign(p_pcdesk_locx,0.75),
        assign(p_pcdesk_locy,2.75)],
        [add(relation(on(desk1,layout))),
        add(relation(on(pcdesk1,layout))),
        add(relation(location(desk1,0.75,0.5))),
        add(relation(location(pcdesk1,0.75,2.75)))]],-0.290862]

```



```
request : [agent4,[retrieve_preferred_and_score,0]]
reply : [[0,agent2,[t1,t2,t3,t4,t5,t6],
         [assign(p_desk,desk1),
          assign(p_desk_locx,0.75),
          assign(p_desk_locy,0.5),
          assign(p_pcdesk,pcdesk1),
          assign(p_pcdesk_locx,0.75),
          assign(p_pcdesk_locy,2.75)],
         [add(relation(on(desk1,layout))),
          add(relation(on(pcdesk1,layout))),
          add(relation(location(desk1,0.75,0.5))),
          add(relation(location(pcdesk1,0.75,2.75)))]],-0.290862]
request : [agent2,[update_value,[p_pcdesk_locy,2.75]]]
reply : _1392288
request : [agent3,[update_value,[p_pc_locy,2.75]]]
reply : _1392288
request : [agent3,[update_value,[p_pc_locx,0.75]]]
reply : _1392288
request : [agent2,[update_value,[p_pcdesk_locx,0.75]]]
reply : _1392288
request : [agent2,[update_value,[p_pcdesk,pcdesk1]]]
reply : _1392288
request : [agent2,[update_value,[p_desk_locy,0.5]]]
reply : _1392288
request : [agent2,[update_value,[p_desk_locx,0.75]]]
reply : _1392288
request : [agent2,[update_value,[p_desk,desk1]]]
reply : _1392288
```