Single Function Agents and their Negotiation Behavior in Expert Systems

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Computer Science
by

Bertram V. Dunskus, August 24, 1994

Approved:

Professor David C. Brown, Major Advisor

Professor Robert E. Kinicki, Department Head

Abstract

A Single Function Agent (SiFA) is a software agent, with only one function, one point of view, and one target object on which to act. For example, an agent might be a critic (function) of material (target) from the point of view of cost. This research investigates the possibilities and implications of the SiFA concept, and analyzes the definition language, negotiation language and negotiation strategies of the agents.

After defining a domain-independent set of agent types we investigated negotiation, analyzing which pairs/groups of agents have reason to communicate, and what the information passed between them should be, as well as what knowledge was needed to support the negotiation.

A library for the CLIPS expert system shell was built, which allows development of SiFA based expert systems from domain independent templates. We will present two such systems, one as implemented for the domain of ceramic component material selection and the other (in development) for simple sailboat design. The effect of negotiation on the design process and the results are discussed, as well as directions for future research into SiFAs.

This work is dedicated to my parents, who made my graduate studies possible and supported me all through it, as well as my Volker and Mary Ulbrich, who motivated me to go to WPI and made my life in the USA easier and more interesting.

Acknowledgments

Many people, far too numerous to be named individually, should be mentioned in this section. I would like to thank all of them, for their support and inspitaration.

However, my specific gratitude goes to Prof. David Brown, for his continous support and guidance through my entire studies at WPI; Prof. Lee Becker for inspiration and for reading this work; Digital Equipment Corp. for developing the Alpha and for the research grant to the AI group that helped me put so much effort into this work; Phil Tomlinson for his patience and constructive help in the development of SINE; Dan Grecu and Cindy Heitzmann for their support in starting the research into SiFAs; Pete McCann for keeping I3D alive and for evaluating SINE; as well as Gary Riley and Brian Donnell, developers of CLIPS and always online, without whom the platform never would have worked. Special thanks also go to the developers of XMosaic for inventing the world's best procrastination tool.

On the 'not so academic side' my special thanks go to Jonathan Kemble for his creative support, the user interface, the helpful UNIX hints, and all the fun hours (days, weeks, months?) we had together; Jenn Greenhalgh for her love and patience; Ilan Berker for bringing life into our office, and all the people at Masque for making WPI so 'dramatic'.

Finally, I would like to mention all the faculty and staff in the CS department for their support; not to forget the systems people who were always at hand, and Prof. Wills for allowing me into the graduate program. These studies were also supported by a small grant from the Dr. Jost Henkel Stiftung in Germany, which made life much easier.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Overview

This thesis is structured into seven chapters. The first, this introduction, explains the goals and motivations of the underlying work. It then proceeds to explain the basic principles of conflict resolution, negotiation and the paradigm of Single Function Agents (SiFAs), that build the foundation of this research.

The second chapter analyzes the work in this and related fields of research. Existing research into cooperation, conflict resolution, negotiation and software agent communication is presented. Also, three existing systems using Single Function Agents, i.e., Sneakers, I3D and I3D+, are discussed.

The analysis of negotiation between Single Function Agents forms the third chapter. Issues in this section are, where negotiation occurs, what kinds of conflicts lead to it and how they can be resolved. We also discuss the knowledge and functional requirements for the SiFAs.

In the fourth part, SINE, a platform for implementation of and experimentation with SiFAs is introduced and discussed. The goals of its design, architecture, communication language and knowledge representation are explained here. Also, the functionality of con-

flict detection, notification and resolution are presented, as well as the designs of the individual agent types.

The fifth chapter discusses the implementation of SINE, describing the CLIPS programming environment, the object class structure, the implementation of the individual agent types and the graphical user interface. It concludes with a description of how the SINE platform was used for the implementation of a prototype sailboat design tool.

In the sixth chapter, we evaluate the SINE system. We first shows how some of the I3D+ conflicts were simulated with the platform. Then we demonstrate a sample SiFA domain with a more complicated dependency structure. After that we describe an ongoing implementation of SiFAs for the domain of Sailboat Design. We then compare SINE to other SiFA systems, evaluate its understandability, analyze its performance and compare it to systems with larger agent sizes.

In the conclusion, we summarize the insights drawn from the research and the experiences from the development of and work with the platform. To finish up, we present opportunities for future research in the field of SiFAs.

Let us now proceed to the goals that motivated this thesis.

1.2 Goals of the Thesis

1.2.1 Basic Assumptions

The premise of this research is that Design Expert Systems can be built using many small, cooperating, limited function expert systems — referred to as SiFAs. The belief is that by using this approach we will be able to discover and investigate primitive problem-solving

components and primitive conflict resolution (CR) strategies in design systems. The approach should also lead to a deeper understanding of the types of knowledge involved. It stands in contrast to much of the current research in CR in design systems, which assumes powerful agents with relatively unconstrained functionality and knowledge.

1.2.2 Exploration of the SiFA Paradigm

This thesis was motivated by several goals which fall into the two general areas, exploration and experimentation. The first area we define in this section, the second in the following one.

- *Definition of a Domain Independent Set of Agents.*

  In order for the work to have general importance, we need to show that we can design agent types that are suitable for a larger class of design tasks, not just for an individual design problem.

- *Investigation of Agent Negotiation.*

  As there has been only a very limited amount of research into SiFA negotiation, this is probably the most important theoretical contribution of the thesis.

- *Analysis of Communication Patterns.*

  Based on the negotiation research, we want to find common patterns in the exchange between the agents, as this could give us information on how to improve the design and performance of the SiFA systems. It is also important for further research into the possibility of learning within the agents.

- *Knowledge Representation.*

  Past experience has shown that the way in which the knowledge is stored in a system can have a severe impact on its ease of design, maintenance and its performance. Hence, we want to analyze possible representations and choose the ones most suited for the task.

- *Catalog of Conflicts.*

  After investigating the agents' negotiation behavior, we will attempt to list all the theoretically known and occurring conflicts. In this catalog we want to specify when, where and why the conflicts appear, as well as what strategies might be used to resolve them.

- *Use of Design Histories.*

  As agents make decisions by themselves and communicate with one another, they exchange and gain knowledge about themselves and others. It would be interesting to see, how they could use information about their own behavior and past decisions for improving the way they perform their design tasks and their negotiation. This also seems to be an important basis for learning within the agents.

1.2.3 Building and Using a Prototype System

- *A Platform for Design System Implementation and Future Research.*

  We want to build a platform, i.e., a computer program, which would allow us to easily design SiFA systems for new tasks. Also, such a platform can be used to perform a more thorough analysis of SiFAs and extend them to suit future research.

- *Implementation of I3D+ Conflicts.*

  In order to analyze and evaluate the platform, we want to implement an expert system with it. This system will then be compared against a different design expert system, built without the platform. For this purpose, we will use the I3D+ system for comparison.

1.3 Motivation

In this section we present both the practical and the theoretical motivations, that led to the research into the negotiation behavior of Single Function Agents.

1.3.1 Practical Motivation

From a practical point of view, we found three major issues that are important to this work. The first is the growth of Concurrent Engineering (CE) and the implication that participants from different backgrounds have to work together. Due to their heterogeneous backgrounds, their knowledge, goals and preferences are different.

If we try to build expert systems that support or model the work in CE, we face the issues of inherently conflicting knowledge and interests. Modelling these is difficult in traditional expert system architectures. Usually this means that we have to build a knowledge base which is mostly conflict free, which requires us to anticipate at development time all possible situations and outcomes, in order to avoid conflicting ones.

This conflict-anticipation would have to be done not only while developing, but also during maintenance of the knowledge base. As expert systems are meant to work in domains where enumerating or trying all possible solutions is prohibitively expensive, and unrea-

sonable to do in advance, we want to find a different way of solving this system design problem.

The second practical issue is that we often want to integrate human and computer experts in one system. This is essential for computer aided engineering. We therefore need a way to provide an interface between these agents that is suitable for both sides.

The third issue is the following: if we manage to design the interface and the system in general in such a way that it supports location of the agents at different sites, then we can support another trend in engineering, the distribution of work over multiple locations. Many knowledge sharing tools already address this issue [Kuokka et al. 1993] and we will present some of them later in this thesis.

1.3.2 Theoretical Motivation

On the theoretical side, the work is stimulated by four research aspects. First, there has been some work in Generic Tasks (GT) and design system languages such as DSPL [Brown & Chandrasekaran 1989] [Brown 1992 b].

In generic tasks, an attempt is made to formulate human information processing tasks on an abstract level, e.g., selection between alternatives is a task we are often faced with. The research formulates a general model of how we perform this task. Given the model, a computer system to simulate that task can be implemented.

Design System Languages such as DSPL give developers a powerful computer language in which they can formulate the design task. The language supplies constructs, which represent the steps or tasks that would have to be executed by a human designer. When running, the system simulates that task and, if possible, comes up with a design, as requested.

Although DSPL was originally intended to be a generic task, it was later recognized as a mixture of GTs after its development [Brown 1994].

The second background against which we can view the work on SiFAs is Distributed Artificial Intelligence (DAI) [Bond & Gasser 1988] [Huhns 1987] [Huhns & Gasser 1989]. That research area focuses on splitting AI systems into multiple parts which can be executed simultaneously, and possibly even at several locations. For example, a search task could be performed by multiple search engines running on a multiprocessing machine or on multiple workstations on a computer network. SiFAs share with DAI some issues, such as the maintenance of multiple local memories, passing information between agents, splitting a task into multiple subtasks and assembling the solution from sub-solutions.We will look at these issues in the discussion on SiFAs in chapter 3.

The issue of conflict resolution that was mentioned in the introduction has been investigated by several authors. We will see more detail in the discussion on previous work in chapter 2, and in chapter 3, where we discuss how the SiFA systems can make use of this conflict resolution research.

The last theoretical issue is somewhat more implementational, but nevertheless important. Unlike traditional expert systems, the SiFA platform in this work has been developed with an Object Oriented approach in mind. We will see in the discussions on the system design and implementation how this affects the architecture and performance of the system.

1.4 Negotiation

Very commonly in our everyday situations, we discover conflicts and we solve them (at least most of the time). In some cases, we find it difficult to determine for ourselves how

we want to decide on an issue. This would then be an intra-personal conflict. However, in those cases when we have disagreements with other people (inter-personal conflict), we will try to reach a common solution with them.

Although many definitions of negotiation are possible and many are found in the literature, for the purpose of this thesis we define the following one:

> Negotiation is communication between two or more active entities (agents), with the purpose of achieving an agreement on a common topic, over which there is currently a disagreement.
>
> The disagreement is marked by possibly differing goals and/or informational states of the agents, and the process of negotiation will often lead to a change in the state or processing model for one or more of the participants.

Let us take a look at the implications of this definition. First of all, there are two or more active entities involved. Active, in a sense that they are able by some means to make decisions on their own and react to external information. Then, we need communication. There are different ways to achieve that and in section 2.5, “Communication”, we will look at different ways of implementing communication in multi-agent systems.

Davis and Smith present a good definition of the components of negotiation [Davis & Smith 1981]:

- There is a two-way exchange of information;
- Each party to the negotiation evaluates the information from its own perspective;
- Final agreement is achieved by mutual selection.

1.5 Single Function Agents

In the course of developing expert systems for support in concurrent engineering tasks, research at WPI has developed a model which involves multiple agents that cooperatively produce a solution. In particular, it has been found useful to separate the task of the entire system into many, very small subtasks and assign exactly one of these to an individual agent. Every agent now has exactly one function to perform, namely to execute this subtask. These agents are called “Single Function Agents”, SiFA for short.

As this idea proved to be so useful, it has been analyzed more carefully. The result was that it is possible, in general, to specify an agent’s task by three parameters (see Figure 1-1).

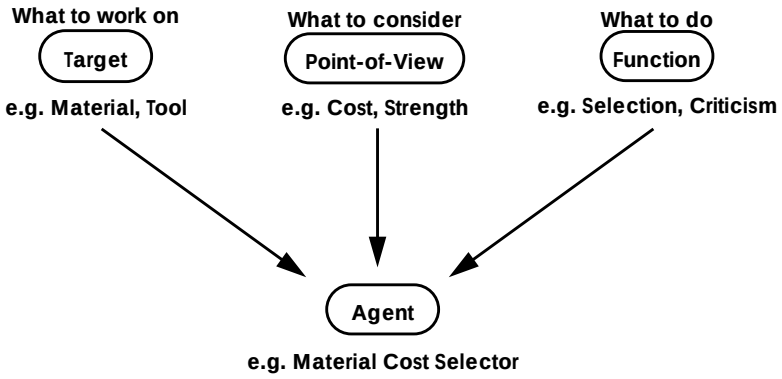

FIGURE 1-1: **SiFA Definition**

- Function

  The function of the agent defines what kind of work it performs. Possible functions are, e.g., providing Advice, Analysis, Criticism, Estimation, Evaluation, Planning, Selection and Suggestion. These functions are similar in spirit to the generic tasks described in section 1.3.2, “Theoretical Motivation”.

- Target

  The target defines on what parameter or object the agent has an immediate effect. A target could be, e.g., the material that an artifact is made from, or the process that will be employed to produce the object.

- Point-of-View

  The point of view specifies the perspective that the agent takes, as it performs its function on the target. That might be cost, strength, manufacturability etc. The agent will probably try to optimize the performance of the artifact with respect to its point of view.

Given these three parameters, we can now specify agent roles very easily. For example we could have a selector (function) of material (target) from the point of view of strength. Another role could be criticism of material from the point of view of cost. Maybe a third agent would select material from the perspective of thermal-conductivity, which would probably lead to different preferences than the first selector, and therefore to conflicts.

Figure 1-2 shows the wide range of values, that can be used to specify the parameters that define the agent roles.

We already see that many agents share some of their parameters, i.e. they have identical functions, targets and/or points of view. This will prove to be useful in the design of the system, as well as interesting during the analysis of negotiation.

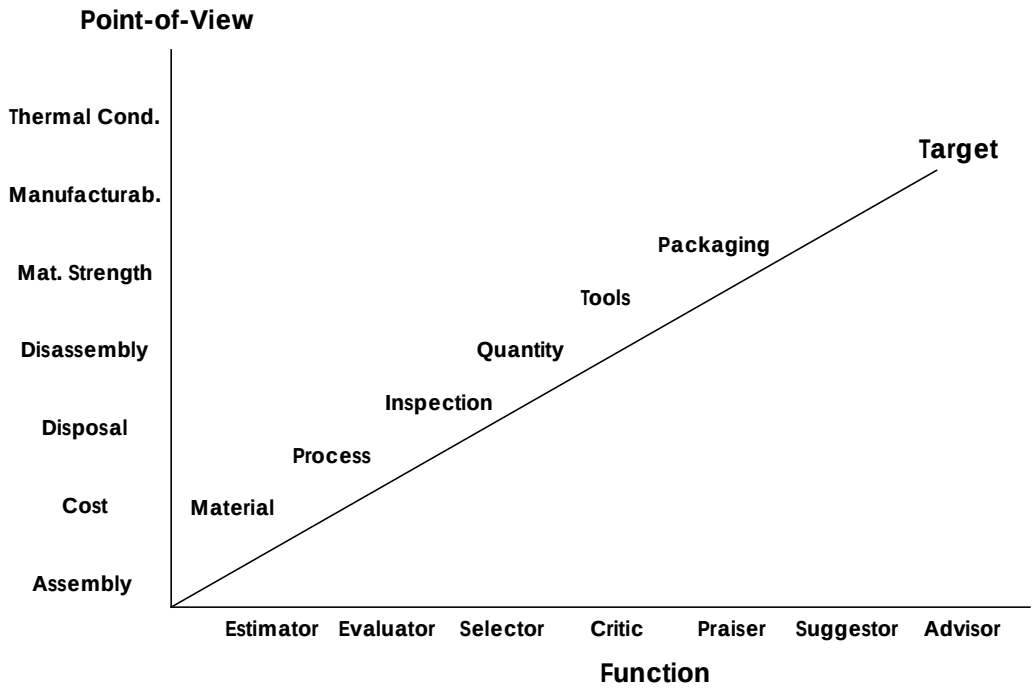

FIGURE 1-2: **SiFA Dimensions**

1.6 Conventions

For clarity, we define the following concepts:

- User: The person that uses a design expert system based on SINE.
- Developer: The designer and programmer of an expert system that uses the SINE platform and language.
- Author: The original developer of the SINE platform and author of this thesis.
- Agent: An expert system that can act and behave based on built in knowledge, possibly communicating with other agents.

Although users and developers can be of any gender, for the sake of readability, we will use the male gender in future references. This is by no means meant as a devaluation of the female.

1.7 Summary

In this chapter we presented the goals of the thesis (section 1.2), including the basic assumptions that lead to the research and the two major threads of scientific exploration of the SiFA paradigm and the more application oriented task of building a reusable platform for design systems with negotiating SiFAs.

We then investigated the motivation for this work (section 1.3). One one side there was the practical motivation from work with concurrent engineering systems, while on the other side we presented relations to the research into distributed artificial intelligence.

We explained our definition of negotiation (section 1.4), the components of it and the resulting implications for a system that works with negotiation.

The next section presented the basics of Single Function Agents (section 1.5), their three-fold specification based on function, target and point-of-view. We also showed some of the possible values for each of the three definitional parameters.

The last part of the chapter was devoted to clarifying name conventions that we use in this thesis.

CHAPTER 2

Previous Work

2.1 Introduction

In this chapter, we review the relevant research literature. We will first investigate what cooperation means and how it is being used in expert systems. Then, we will present different research that has focussed on conflict resolution and generated various models to support it.

In the third section we will see how negotiation has been used in several research applications and especially design systems. After that, we will look at what kinds of communications schemes have been developed to support agent interaction.

The last section will present three systems that have been developed using the SiFA paradigm. The first one, SNEAKERS, was developed to train users about the importance and power of concurrent engineering. It used the domain of tower design with Tinker-Toys as a demonstration of the impact of CE. A user would takes design actions and the system would react with feedback from a multitude of agents. The second one, I3D used multiple, cooperative intelligent agents to serve on a design team with a human designer. It assisted the user in decisions on selection of powder ceramic materials, production processes and

inspection planning. The third one, I3D+ was developed from I3D, in order to investigate some negotiation aspects of SiFAs and demonstrate conflict resolution in a SiFA system.

2.2 Cooperation

As mentioned in section 1.4, “Negotiation”, cooperation is found in daily life as well as in computer systems. The introduction of agent-based systems in artificial intelligence has opened a whole new field of research and exploration.

There are different definitions of exactly what cooperation means. One view is to see cooperation as the opposite of selfishness. For the agents, this would mean that they are willing to give up local goals in order to help satisfy global ones [Liu & Sycara 1993].

In another, weaker, definition, cooperation means that the agents are willing to exchange information, but no commitments to global goals are prescribed [Klein 1991]. Agents could cooperate just for the sake of satisfying their own goals. Not cooperating would mean, that they do not supply knowledge, or that they give false information on purpose.

The second definition is the one we adopt in this thesis. It reflects the fundamental willingness of the agents to work together. The behavior in goal trading is left to the area of conflict resolution.

In the literature on cooperation we find two main areas:

- Cooperation for simulation and representation of physical object.
- Cooperation in representation and modelling of reasoning of agents.

Even though there are overlaps between the two areas, the main focus of the first lies in recognition and exchange of information about the physical world. This is often researched in robotics [Connah et al.].

The second area focuses on agents that are involved in cooperative problem solving, i.e., agents share a common goal and try to achieve it by using the knowledge and abilities that they all provide. Often, the most important part of their work is building a plan. In order to do that the agents have to exchange information. They share information about their respective goals, as well as descriptions of their view of the world that they act in. The first helps them plan cooperatively, by harmonizing their decisions. The second prevents poor decisions that could occur due to the limited view of their environment that the agents might have [Grosz & Sidner 1990] [Sidner 1992].

2.3 Conflict Resolution

Whenever a decision has to be made, with multiple reasons and goals taken into account, we risk encountering a conflict. That is, the individual goals that affect the decisions would lead to different results. For example, we might like to buy a sports car, for the fun of driving, but a station wagon would be more useful when going on vacation. These goals clash, unless we find a sports station wagon (a case where the car industry helps us resolve our conflicts), but otherwise we have to find a way of resolving the problem, maybe by ranking the two goals by importance or by taking additional constraints (financial budget) into account, which eliminate some of the conflicting preferences.

If a conflict happens within one person, we call it an intra-personal conflict. If it occurs between several people, it is a inter-personal conflict. Agent-based systems can be built to have more of the first or the second kind of conflicts.

The research into conflict resolution has long been rooted in the area of social studies. As the modelling of human behavior is one of the original application fields of artificial intelligence, it is little wonder that many systems have been written to support and analyze conflict resolution. We will analyze to kinds of systems, those that perform conflict resolution in arbitrary domains and the systems that are specialized for design conflict resolution.

2.3.1 General Systems

The purpose of these systems is not restricted to performing design tasks, but they can be made to model any kind of decision making. Some of the systems use very detailed schemes for storing conflict resolution knowledge in an abstract way.

For example, Sycara's "Situation Assessment Packages" (SAPs) [Sycara 1987] are information structures that hold, among other things, the following contents:

- a description of a problem solving situation
- associated expectations about what is supposed to happen
- reasons why the expectations are violated
- responsibility for the violation
- solutions that could be achieved by a third party problem-solver
- justifications for the possible solutions
- warnings for potential failures

The purpose of SAPs is to prepackage knowledge about situations with psychological considerations of the participants. From these informations, in case of a conflict, a causal structure (graph) made from nodes, states and links can be built. The nodes describe the goals, the states the actions of the agents. The links signify relations between goals and

states. These causal structures then store generalized information about conflicts, they organize expectations and explanations of failures.

When a conflict is detected, a match of the situation against all SAPs is done. The most specific SAP is retrieved. Because the description of the conflict situation is abstract, i.e., it does not relate to any particular variable or goal in conflict, it has to be instantiated in the current situation. After that, an explanation can be derived from the reason information in the SAP. The solution method is passed to a problem-solver who then proceeds to remove the conflict.

If a new conflict type is discovered, it is abstracted and characterized, after which it is formed into a SAP and stored in memory, similar to a case base.

SAPs can be used by multi-agent systems. They support partial goals, allow blame assignment and store cross-contextual cases through abstraction.

What remains unclear is whether these ideas can be fully implemented. The matching of pre- and post-conditions of SAPs could be difficult. If implemented, it would be applicable to any kind of problem situation.

Sometimes authors only provide a model for the different kinds of information that can be exchanged and what their semantics are. Werner presents a very complete theory, with explicit representation of agents' informative, intentional and evaluative states [Werner 1989]. Messages that are passed between the agents are designed to only affect one of these three states at a time. Through this, the semantics of communication acts are defined very well. He then shows how several kinds of conflicts can be resolved with different combinations of the three kinds of messages

His model is close to the one used for planning discussed in [Grosz & Sidner 1990]. The logic is strong enough to represent other models, such as the contract net. Werner defines states, worlds/histories and roles. He defines how agents change their role and when this can occur.

Other authors provide an algorithm for solving conflicts independently of the agents. Wong's work on cooperation [Wong 1992] is interesting for the implementation of social choice as a solution paradigm. He uses a set of agents with preference operators for each one. If a conflict occurs, the system will try to maximize the preference functions for all agents. He introduces a formal model of preferences and a logic to support operations on individual and sets of preferences.

These papers are interesting, because they show what kinds of knowledge can be exchanged between agents and how task and domain-independent conflict resolution can be supported.

2.3.2 Design Systems with Conflict Resolution

The second category of research focuses specifically on conflict resolution in design systems. Here we find the work of Liu and Sycara, where Constraint Partition and Coordinated Reaction as a methodology is presented [Liu & Sycara 1993]. The idea is to separate the design task into subtasks according to variable types. Agents are responsible for certain variable types. In the sample domain (job shop scheduling) there are Order and Resource agents. Agents try to resolve conflicts through application of heuristics. These will try to minimize ripple effects and the need for further CR.

Important for our work is the research by Klein [Klein 1991] [Klein & Lu 1990]. In their system, a double hierarchy is introduced (see Figure 2-1). The first one consists of a tree of conflicts, the most abstract one at the top and concrete conflicts at the bottom. If a con-

flict is detected, then it is matched against that tree to recognize it and find the most specific conflict type. At the same time as the conflict is analyzed, the path to the root of the tree provides a series of abstractions that can be used during conflict resolution.

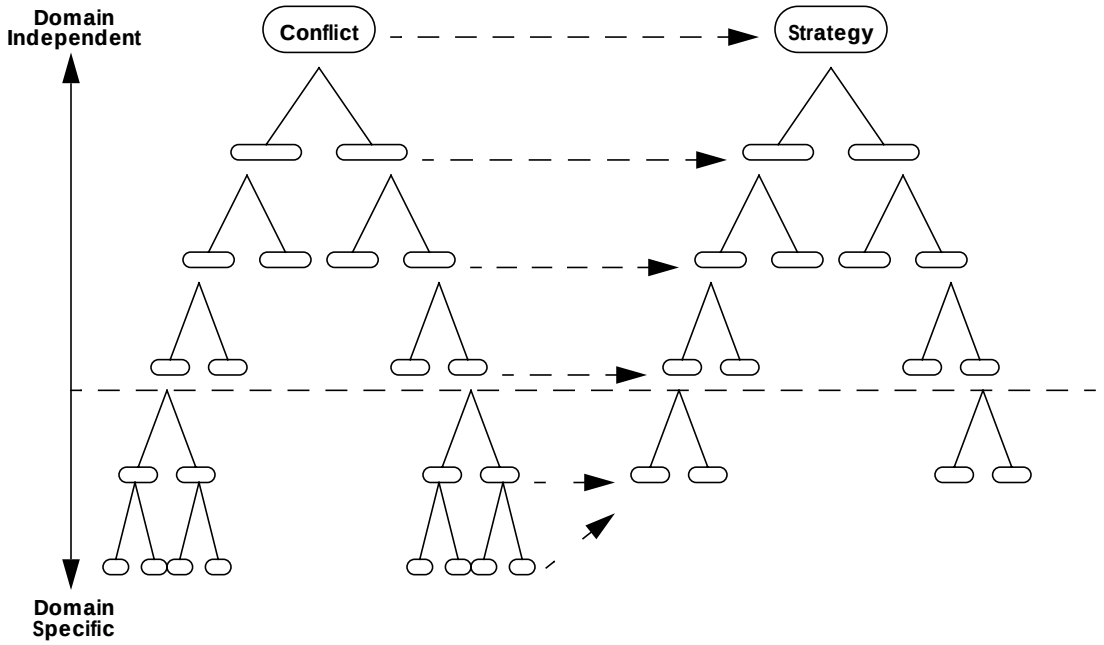

FIGURE 2-1: **Klein's Conflict Resolution Approach**

The second tree consists of conflict resolution strategies. After the conflict type and its abstractions are determined, the system tries to find the most specific strategy that is appropriate for this conflict and solve the conflict by applying it. If this fails to solve the conflict, a more general method is used, and so forth, until all the strategies up to the root of the strategy tree have been applied. If that one fails too, then the conflict definitely cannot be solved and the design fails.

The remarkable aspect of this model is that it supports both domain dependent and domain independent CR, and that there is a smooth transition from one to the other. It is also able to recover from failures in the CR, by applying the more general approaches, if the more specific ones fail.

2.4 Negotiation

For negotiation, we found generic and design-oriented research, just as we did for conflict-resolution research.

2.4.1 Generic Negotiation Systems

Khedro and Genesereth present a strategy, “Progressive Negotiation”, which minimizes backtracking and guarantees the consistency of distributed solutions [Khedro & Genesereth 1994]. An agent is assumed to have Knowledge K (a set of predicate logic axioms), Criteria Constraints C (also a set of predicate logic axioms) and a database D (a set of ground predicate logic atoms). Furthermore, a facilitator agent manages the communication and translates between the different vocabularies of the agents.

The authors now distinguish three types of conflicts:

- Critical Conflict:

  One agent **R** has knowledge K that is inconsistent with a solution proposal that it received from the facilitator. This is solved as follows: R sends its problem statement P (the failing axioms) to the sender **S**. **S** in turn checks whether it can find a solution that satisfies all the axioms, i.e., make $P \cup K$ consistent. If so, it sends the updated proposal to **R**, and the conflict is resolved. This may involve constraint relaxation. If **R** cannot find an appropriate solution, it fails.

- Non-Critical Conflicts with Authority:

  Again **R** sends its problem statement P back to **S**. If **S** can accommodate the additional constraints, it will send out a new solution proposal. If $P \cup K$ is inconsistent in the knowledge of **S**, i.e., **S** cannot find a solution, and **S** has authority, it sends a reply to **R**, which in turn relaxes its constraints.

- Non-Critical Conflicts without Authority:

  Again **R** sends its problem P to **S**, which now checks whether $P \cup K \cup C$ is consistent (i.e., it can find a solution that satisfies its knowledge and its constraints). The solution process is the same as in the second case.

In essence, these are instances of a subset of the strategies that Klein has, i.e., accommodate additional constraints if possible, or relax constraints.

Davis and Smith coin the term ‘distributed problem-solving’ (DPS) and present a comprehensive analysis of DPS compared to distributed computing [Davis & Smith 1981]. They define DPS as problem solving performed as a cooperative activity of a group of decentralized and loosely coupled knowledge-sources. They mention several reasons why DPS is useful, among them that it promises an increase of speed, reliability and extensibility, as well as increased tolerance to uncertainty in data and knowledge.

Furthermore, they address design issues of DPS systems. First, coherence in the actions of the agents is difficult to control, since every agent only has a limited and local view. This limited perspective, on the other hand, simplifies the problem solving, as the agents only have to look at their local subproblems. In order to reduce the potentially negative impact of the decomposition into subproblems, Davis and Smith suggest to provide three elements in a DPS framework:

- The concept of negotiation as a mechanism for interaction

- The network of tasks that result from the problem decomposition
- A common language shared by all nodes

We will later see how these issues receive support in the SINE system.

Laasri et al. present a ‘Generic Model for Intelligent Negotiating Agents’ [Laasri et al. 1992]. Among with a well founded theoretical background, they present the ‘Recursive Negotiation Model’, which serves as a basis for classifying and specifying where conflict resolution among multiple experts is needed. They also emphasize that negotiation can be used in both domain-level and control-level problem solving.

Lander and Lesser present the TEAM system, which implements the negotiated search paradigm [Lander & Lesser 1991]. Here, several agents iteratively search for a solution (currently in pairs). They can use any approach to finding a point of agreement (on a scale), although the linear compromise strategy usually gives the best efficiency.

In TEAM, the conflict resolution works in two steps: *Search* and *Compose Solution*. Search can produce sets of alternative solutions. A selected solution will have to be *complete*, i.e., acceptable by all agents, and *acceptable*, as defined by a user function, which gives a threshold for quality.

Communication is established through a central blackboard (BB). A central controlling agent, the Framework Controller, receives messages from the other agents and updates the BB. The system works in cycles: First the agents process their data and post their messages, then the controller propagates all the changes to the BB. Agents can be ‘heterogenous’ because there are no assumption about *how* the agents work.

Blackboard mechanisms are used in most centralized systems, i.e., systems that have a single point at which they store all or a majority of the shared information. This is due to

the fact that they allow for easy transfer of common data, and no coherence problems occur. Distributed systems however need a different way to maintain global data, unless they are centralized with an information server at one place.

2.4.2 Negotiating Systems in the Design Domain

Sycara is among the most prominent developers of negotiating design systems. In [Sycara 1990], she presents a model that integrates several kinds of design knowledge, with emphasis on case-based design. The idea is to build an abstraction ontology of design parameters and a dependency graph, using qualitative description (sign, amount, importance, feasibility). The agents have some understanding of the dependency and they can share a vocabulary at some level of abstraction. All goals have an importance ranking.

She presents four conflict situations:

- Conflicting recommendations for values from different agents.
- A proposed value makes selection of a dependent attribute's value impossible for another agent.
- A design decision made by one agents negatively affects the optimality of another agents' choice.
- Plan conflict: alternate approaches exist for similar results.

Agents can exchange different kinds of information:

- design proposals
- justifications and reasons
- agreements
- requests

- utilities and preferences

The idea of a shared vocabulary can be realized with an object hierarchy and inheritance of communication skills (ability to produce and understand messages) from more abstract agents. The richness of possible communications makes the design of negotiation schemes very difficult for this system. The agents have to be very complex, because they have to be able to find out, what language they share with a negotiation partner, every time they encounter a conflict. Nevertheless, the agent functionality and the richness of the communication promise to provide very powerful CR abilities.

Werkman presents ‘Designer Fabricator Interpreter (DFI)’, a system for cooperative evaluation of beam-to-column connections from multiple points of view [Werkman & Barone 1991]. In the DFI system three agents assist the user in selecting the best connection, by evaluating and suggesting a connection type. As they have to agree on the suggestion, a central arbitrator is used to trade off the different preferences.

In section 2.6.3, “I3D+”, we will see another negotiating design system, which was developed at WPI.

2.5 Communication

The transport of messages from agent to agent is an issue that has been heavily researched and still receives considerable attention in artificial intelligence research. As is so often true, the background stems from the effort of replicating human interaction. Major efforts are designing computer languages to capture all the needed information and formulating clear semantics for the information exchange.

On the other hand, distributed computing and distributed artificial intelligence have brought about many more issues, such as the need for a transport protocol and meta-services, such as locating another agent or brokering information.

2.5.1 Internal-only Communication

The first kind of communication we find has the sole purpose of allowing the agents in the machine to exchange messages. No specific effort is made to make the communication understandable by humans, hence we refer to it as “internal-only”. We can distinguish the models by their explicitness, i.e., how directly an agent addresses a message to another agent. In the work of Liu and Sycara [Liu & Sycara 1993], we find only indirect communication between the order and the resource agents. They tag activities that they try to schedule with coordination information that other agents can read.

More explicit, but not necessarily more understandable by humans, is the messaging scheme that Agha and Hewitt adopt in their object-oriented model [Agha & Hewitt 1985]. Their agents communicate in a variation of LISP (Apiari), and their messages are executable pieces of code.

Cromarty discusses various forms of agent connection topologies [Cromarty 1987], and then formulates a model that facilitates communication between them through ports, which are an abstraction of the topology.

2.5.2 Speech Acts

Most often, we find models that are based on speech acts [Austin 1962] [Searle 1965]. Speech acts, also called language acts or linguistic acts, are minimal units of linguistic communication. Performing a speech act means to engage in a rule-governed form of behavior, producing output that has been intentionally designed. Models based on speech

acts use short messages that are similar to human sentences for the exchange of information and knowledge between the agents. The expressions used in the acts have a predefined meaning that both sender and receiver agree upon.

A simple example of a negotiating system with speech acts can be found in [Bussmann & Mueller 1993]. They present agents that try to build a certain word from letters that they have. The bargaining starts, after every agent has received a specific task, i.e., a word to build and a set of letters to start with. The agents plan to build a word, then try to exchange letters with other agents. Their partners may accept or reject the exchange deal. If an agent cannot achieve his goal, he will select a different plan.

The Communication acts include: inform, request, accept, deny, propose, answer. Agents can request information and actions from one another. Goals get expanded into action steps according to a fixed grammar. A history of all speech acts is kept locally by every agent, for retrieval of information about which agent has which letters.

2.5.3 Hybrid Communication Architectures

In [Taleb-Bendiab & Oh 1993], a cooperative design system is presented. It uses a hybrid communication architecture, i.e., the agents can communicate directly with one another, or post public messages on the blackboard.

Their language is made from communication primitives, which include: request, analyze, calculate, compare, query, inform, accept, disagree, evaluate, comment. The speech acts can contain a single instruction, or a set of actions (a plan), that one agent expects the other agent to perform.

The examples are based on the mechanical design of a fluid coupler. Participating agents are the Cost, Assembly, Manufacturing and Design Agents. A design Manager helps coor-

dinate, while a Geometric Reasoner, a Constraint Processor and a Truth Maintenance subsystem are supporting the agents. This work is based on the SIMAD (System for Improving Mechanical Assembly Design).

2.5.4 KQML Knowledge Query and Manipulation Language

In order to provide a separation of message transport, semantic information and content, the DARPA Knowledge Sharing Initiative External Interfaces Working Group has developed the KQML standard. It defines a language that can be used as a wrapper around message contents, so that messages can be handled in a uniform way, independent of the information content. It also specifies some aspects of how the messages are to be interpreted, as well as where they come from and to whom they are addressed. Here are some of the specific aspects of KQML:

- KQML adds on to domain content languages, such as KIF (see section 2.5.5, “KIF”) It wraps around the contents and allows the exchange of information in the content language.
- It facilitates the communication between autonomous agents by providing primitives (performatives).
- Agents are seen as programs manipulating a database. KQML helps other agents mutually access these databases and transfer requests etc.
- It makes no assumptions about the way in which the messages are transported, except that the transport is reliable and that it passes messages First-Out-First-In.
- Agent data are called virtual knowledge bases (VKB), because it is not important in what form the knowledge is stored and manipulated within the agent.
- VKB’s are supposed to hold at least two kinds of information: Beliefs and Goals

They consist of a performative name (e.g., ask-about, tell, evaluate, broadcast, broker) and some named arguments (:content :in-reply-to :language :ontology :receiver :sender, and more). Performatives are keywords that specify how the content is to be interpreted by the receiving agent. They fall into different categories:

- basic informative (tell, deny, untell)
- database (insert, delete)
- basic responses (error, sorry)
- basic query (evaluate, reply, ask-if, ask-about)
- multi-response query (stream-about, stream-all, EOS)
- basic effectors (achieve, unachieve)
- generators (standby, ready, next, rest, discard, generator)
- capability-def (advertise)
- notification (subscribe, monitor)
- networking (register, unregister, forward, broadcast)
- facilitation (broker, recommend, recruit)

Ontologies define the meanings of the things that get talked about. They are an explicit specification of a conceptualization of an abstract simplified view of the world. A common ontology defines the vocabulary with which queries and assertions are exchanged among agents. An ontological commitment is an agreement to use the shared vocabulary in a coherent and consistent manner [Gruber 1993].

Ontologies are external to KQML and can be defined in KIF, or by special help from Ontolingua, an ontology maintenance tool. An ontology describes all of the objects that an agent can communicate and reason about. We can define several ontologies, specific to

certain domains, and large unified ontologies that attempt to represent everything that can be communicated about in a software agent system.

Example 1: A simple query and response

Agent A asks for the value of the motor torque at the simulation time 5:

```
(evaluate   :language KIF
            :ontology motors
            :reply-with q1
            :content (val (torque motor1) (sim-time 5)))
```

and B replies with the appropriate value:

```
(reply      :language KIF
            :ontology motors
            :in-reply-to q1
            :content (scalar 12 kgf))
```

Example 2: Requests to achieve states

Agent A asks agent B to achieve a state such that the torque of the motor is of 2kgf at simulation time 5:

```
(achieve    :language KIF
            :ontology motors
            :in-reply-to q1
            :content (= (val  (torque motor1) (sim-time 5))
                              (scalar 2 kgf)))
```

Agent B tells agent A the achieved torque:

```
(tell       :language KIF
            :ontology motors
            :in-reply-to s1
            :content (= (val  (torque motor1) (sim-time 5))
                              (scalar 12 kgf)))
```

Developers may decide to support only a subset of the performatives. They may also decide to use additional performatives, but they should use the existing ones as far as possible.

KQML has been used in a handful of systems, including PACT (Palo Alto Collaborative Testbed), with message passing via LISP, TCP/IP, email and CORBA (Common Object Request Broker) (see 2.5.6, “SHADE”, and [Kuokka et al. 1993]).

2.5.5 KIF

KIF is a formal language for the interchange of knowledge among disparate computer programs. It is not intended for human computer interaction, though it is readable by humans. It is also not intended for internal knowledge representation, although it can be used for that purpose. The origins of the syntax was adapted from the LISP programming language, and KIF still is very close to the Common-LISP.

KIF allows expression of first order predicate logic. It uses objects such as constants, variables, sets, lists, functions and relations. Variables can be quantified (existentially and universally), functions allow the definition of implications and rules, e.g.:

```
(material moon stilton)
(flies bird)
(believes john ?p)
```

It allows the expression of meta-knowledge, i.e., KIF can make statements about other KIF statements, by putting them into KIF expressions.

```
(=> (believes john ?p) (believes mary ?p)) ; implication
```

KIF supports non-monotonic reasoning, through ‘consis’ which is a predicate that evaluates to true if the situation, that its objects produce, would be consistent, e.g.:

```
(<<= (flies ?x) (bird ?x) (consis (flies ?x)))
```

which is a backward chaining rule that says, the bird flies, if that is consistent with other knowledge. This allows for default rules, closed world assumptions etc.

KIF allows a constant to be specified by an analytic definition. The specification can be partial or complete, e.g.:

```
(defobject id (= (f ?x id) ?x) (= (f id ?x) ?x))
```

This defines the constant `id` to be a left and right identity for the binary function `f`.

KIF was developed by Michael R. Genesereth and Richard E. Fikes at Stanford in conjunction with KQML and Ontolingua, to support knowledge sharing between agents.

2.5.6 SHADE

Kuokka describes a set of systems that have been implemented using KQML and KIF [Kuokka et al. 1993]. The main thrust lies in building cooperative design systems that will support knowledge sharing between several agents (human and computer) and software tools.

Apart from design agents, there are facilitation agents that help in the communication by brokering information, i.e., they receive offers from agents to supply information, and requests from other agents to receive the same. The brokers try to locate information sources and sinks, and provide the appropriate connections to the other agents. This proved very useful in a distributed system, such as the internet.

Shared design knowledge is represented by KIF (first order predicate logic) using an ontology specified by Ontolingua. Contents are marked with attitudes (belief, interest, ability, expectation, fulfillment). Both LISP and C-Language Application Program Inter-

faces for KQML are available. Message transport can be performed by any means, but Service Mail (email) has been used most of the time for security reasons.

2.6 SiFA Systems

2.6.1 Sneakers

SNEAKERS was the first SiFA system developed at WPI. A few lines from its introduction clarify its purpose:

> "SNEAKERS is a single user demonstration system. Its task is to design towers composed of pieces similar to Tinker Toys ™. The screen is set up in a windowed environment, with which the user interacts using a mouse. Various expert systems run in the background and offer criticisms and suggestions to the user concerning recent design decisions. SNEAKERS is easy to use, and helpful in demonstrating the value of CE." [Douglas 1992]

As a SiFA system, it has several agent types, which it adopts from [Brown 1992 a]:

- Advisor — makes recommendations for the next decision;
- Critic — compares the design to standards, offers criticism on the last user action;
- Suggestor — takes the criticism and offers suggestions for satisfying them;
- Analyst — offers numerical analysis to derive attributes, such as strength cost or size;
- Evaluator — evaluates the whole design from one perspective, determines how well the design meets the needs of a certain perspective.

It also has a set of points-of-view:

- Design — Rules about structure, design process, symmetry, stress and buckling analysis;
- Manufacturing — Knowledge of the manufacturing process, connection of pieces on site and in house, rules on composition of connectors and materials;
- Assembly — Rules on tower height, number of pieces and connector type selection;
- Cost — Cost estimation for materials and processes, suggestions for cost reduction.
- Marketing — Knowledge of customer preferences for certain tower shapes
- Safety — Rules about safety for tower visitors (bracing, hand holds, minimum platform size, tower height).
- Disposal — Concerned with disassembly and recycling.
- Packaging — Preferences for smaller pieces, that are easier to get on site.

As this work was based on the original version of the SiFA idea, it only separated the agents by function and point of view. There was no distinction by target.

When a user designs a tower, the agents in SNEAKERS give feedback from their domain. All the output from the agents goes to the user, there is no communication between agents and also no negotiation. The system produces comments, evaluations and recommendations. As the recommendations and evaluations are coming from different points of view, with different intentions, they are often conflicting. This is useful, because it teaches the users an important aspect of CE.

2.6.2 I3D

The second SiFA system was I3D [Victor et al. 1993]. It interacts with a designer sitting at a workstation. As the designer moves through requirements specification, conceptual design and detailed design of a part to be made from powder ceramic material, the system graphically displays the state of the design on the screen. It makes appropriate assumptions about design decisions not yet made in order to be able to continuously display the component during both the conceptual and detailed stages of design.

In both rough and detailed design phases, the system provides feedback from different agent types and points of view [Victor 1993].

The agent functions found in I3D are:

- Advisor — providing information about what to do next, what parameter to decide;
- Critic — comments on possible problems with existing design decisions;
- Planner — produces a choice of actions and their sequencing;
- Selector — picks one item from a list;
- Estimator — estimates values derived from the current design parameters.

The points of views are

- Material
- Process
- Manufacturing
- Inspection
- Cost

- Reliability
- Durability

Unlike SNEAKERS, the agents in I3D are partially dependent on one another, i.e., the output from some agents is needed for others. This is the reason, why a rigid sequencing of the agents execution was adopted.

I3D too, was developed with a two dimensional agent role model, based on agent function and design aspect. The system did not deal with conflicts, those were eliminated at development time, by not allowing more than one agent to select a value for the same design parameter. This makes addition of new agent knowledge difficult.

2.6.3 I3D+

In the thesis work on I3D+, an attempt was made to remove some of the restrictions that were inherent in I3D [Victor 1993]:

- First, the potential for conflicts was allowed and a mechanism was implemented to demonstrate how conflict resolution might be handled.
- Second, the rigid agent sequencing was replaced by a flexible agenda mechanism, which schedules the agents, based on tasks that they announce.
- Since the agents demonstrate conflict resolution by negotiation, they have to exchange information. This is implemented by using speech acts and message files. The agents communicate directly among themselves, without the use of a central communication manager.

Similarly, the negotiation is done between pairs of agents, without help from a central arbitrator. The agents are responsible themselves for conflict detection and notification.

The negotiation is produced by a sequence of rules that react to a specific conflict situation (which, for testing and demonstration, is asserted as a fact before run-time).

Victor classifies conflicts into 6 situations, depending on the relation between the agents' local goals and the global goal, as shown in Table 2-1. (The grey areas have not been implemented.)

Type	Agent1 Goal	Agent2 Goal	Global Goal	Result
1	X	Y	Z	Either one will win
2	X	Y	X	Agent1 will win
3	X	Y	Y	Agent2 will win
4	X	Y	X,Y	Either one will win
5	X	X	Z	Either one will win, depending on their expertise
6	X	X	X	Either one will win, depending on their expertise

TABLE 2-1: **Conflict Situations in I3D+**

Let us now summarize some of the aspects of I3D+:

- I3D+ simulates the handling of selection conflicts based on goals. It doesn't differentiate between preferences that are allowed to fail, and constraints that must not fail for agents.
- It has no mechanism for constraint relaxation.
- The conflict resolution is domain dependent, it is encoded in the agents knowledge base. There is no support for more abstract, domain independent resolution.
- Agents have no facility to talk about constraints, and therefore cannot learn about other agents' reasons for failure.
- No design history is kept for lookup in later conflicts.
- The simulation only allows one negotiation per target for every run.

- The model has no means of preventing loops.
- Agents have to detect conflicts themselves, by checking the values on the blackboard. This is expensive, as it has to be done every time an agent is considered for scheduling.

2.7 Summary

In this chapter we focused on previous work in negotiation and SiFAs. We started by presenting the four founding aspects of this work.

First, we investigated Cooperation (section 2.2), with its two definitions of ‘non-selfishness’ and ‘willingness to exchange information’.

Second, the definition of Conflict Resolution and the different ways of implementing it were presented in section 2.3. We showed different research models, some of which were focused on design in particular, and some were not geared to any particular task or domain. We described Situation Assessment Packages, Mediator based models, Constraint Partition and Coordinated Action, and Klein’s hierarchical approach.

Third, we presented Negotiation (section 2.4), both for general problem solving and for design. We described the works by Khedro & Genesereth, Davis & Smith, Laasri et al., Lander & Lesser, Sycara and Werkman.

Fourth, we discussed Communication (section 2.5), with internal only communication, speech acts, hybrid models, and two important standards, KQML and KIF. We finished the section by describing the SHADE research, which uses both of these standards.

The last part of this chapter (section 2.6) described three SiFA systems, SNEAKERS, I3D and I3D+, explaining their agent types, targets and points of view. For I3D+ we also discussed the limited amount of negotiation in it.

In the next chapter, we investigate SiFAs and their negotiation behavior.

CHAPTER 3

SiFAs and Negotiation

3.1 Introduction

In this chapter, we will first describe a set of single function agents that we think are widely applicable. Then we analyze where conflicts might occur in a SiFA system.

After classifying the conflicts into categories, we then proceed to describe possible resolution methods for the conflicts. We then show some of the general strategies that can be used by agents in negotiation.

The last two sections focus on the requirements that we can derive for the knowledge and the function of the agents. These will be used to guide the implementations.

3.2 Agent Types

Although we have found a slightly different set of agent types in SNEAKERS and I3D/ I3D+, most of the functions were recurring. We present each individual agent type in the following sections.

3.2.1 Selector/Advisor

A Selector picks one item from a set of alternatives (see Figure 3-1). In doing so, it can

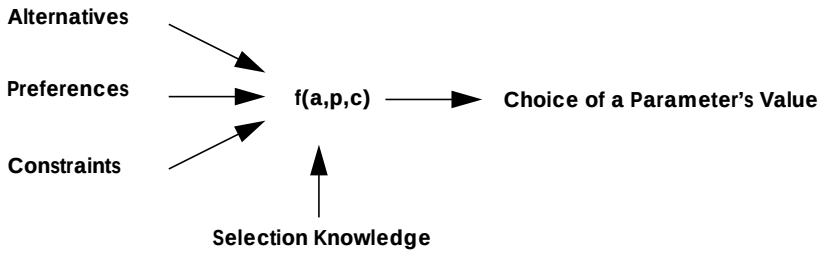

FIGURE 3-1: **Functionality of the Selector Agent Type**

use preferences to rank alternatives, and constraints restrict the alternatives to only valid choices. Some possible items could be choice of a material from a list, suggested next steps, or some parameter for an attribute of the design object

The advisor is a more general form of the selector. It is not bound to an enumerated list of possible choices, but it has an abstract description of what it can choose from. For example, the options could be described by a set of constraints on a numeric parameter, requiring the advisor to use linear optimization to produce a specific value.

3.2.2 Estimator

An Estimator produces approximate values derived from values of attributes of the design object (see Figure 3-2). These values are “approximate”, because not necessarily all the input parameters are available, or some of the input values are by themselves approximated or statistical. In addition to that, the agent can use a function that is only close to, but is not exactly the function that applies in real life. This is often done to save time and effort in early design stages.

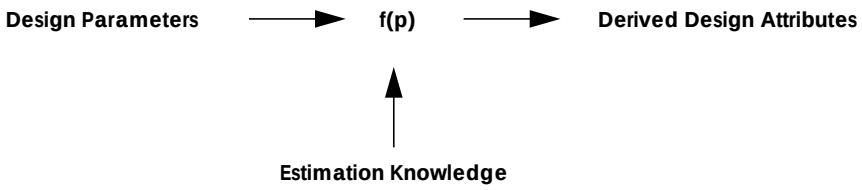

FIGURE 3-2: **Functionality of the Estimator Agent Type**

3.2.3 Evaluator

The Evaluator uses design goals and attributes of the artifact to evaluate the design from one point of view. The result is an information about the quality of the design, i.e., how well the design meets the requirements or some explicit goal (see Figure 3-3). The evalua-

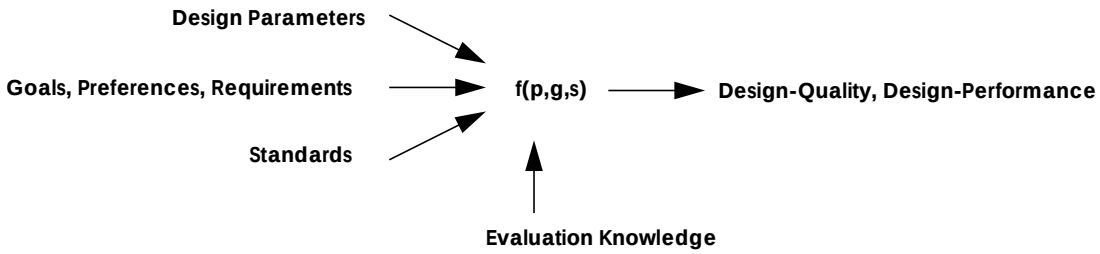

FIGURE 3-3: **Functionality of the Evaluator Agent Type**

tion can be expressed numerically (e.g., in percent), or in an abstract way (e.g., ‘good’, ‘average’ or ‘poor’). The evaluation does not make any statements about whether the achieved quality is satisfactory or not.

3.2.4 Critic/Praiser

A Critic points out potential problems, suboptimal decisions, poor choices that stand in conflict with the preferences that are expressed in the requirements. Positive criticism is generated by Praiser agents.

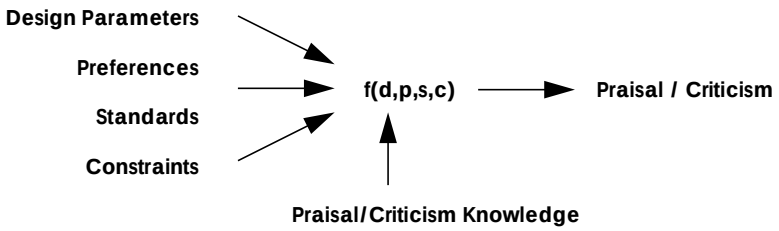

FIGURE 3-4: **Functionality of the Critic and Praiser Agent Type**

3.2.5 Suggestor

The Suggestor takes a criticism and the context and suggests alternative solutions or recommended actions to achieve a solution.

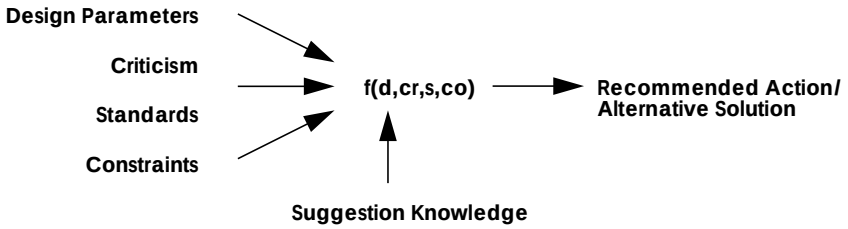

FIGURE 3-5: **Functionality of the Suggestor Agent Type**

3.3 Conflict Occurrences

In Table 3-1, we show a matrix of conflicts. In the rows, we list the agents that initiate the conflict, by discovering it, and in the columns we have the partners, that they go into negotiation with. For example, one of the most common conflicts, is the Selector-Selector conflict. The square in the intersection between the selector row and the selector column describes a conflict cause, then an arrow and a possible resolution step. This table only

Initiators (below):	Estimator	Evaluator	Selector	Critic	Suggestor
Estimator	Conflicting Estimations ⇒ Modification of Estimation Basis		Poor Estimation Basis ⇒ Select different alternative, or give more details		
Evaluator	Incomplete Evaluation Basis, Poor Estimation ⇒ Modify Estimate, add Attributes to Estimate	Conflicting Evaluations ⇒ Modification of Evaluation Basis	Poor Evaluation Basis ⇒ Select different alternative, or describe selection in more detail		
Selector	Poor Estimation Quality ⇒ Request Modification of Estimate	Undercritical Evaluation ⇒ Request Refinement of Evaluation	Constraint Failure, Preferences ⇒ Removal and Reordering of Alternatives		No selection between several suggestions possible ⇒ Refine Rationale
Critic	Complaint about Quality of Estimation ⇒ Modify Estimation Procedure	Complaint about Quality of Evaluation ⇒ Modify Evaluation Criteria	Disqualification of Alternatives ⇒ Additional Constraints	Conflicting Criticism (e.g. too high and too low) ⇒ Agreement on Cr.	
Praiser			Approval of Choices ⇒ Reordering of preferred Selection Method	Conflicting Statements ⇒ Re-evaluate or retract	Unwanted Change Suggestion ⇒ Receive Reason for Improvement
Suggestor	Improve Estimation ⇒ Add/Remove Item Types, Methods	Improve Evaluation ⇒ Add/Remove Attribute Types, Methods	Improve Selection Process ⇒ Intensify Search; Extend Search Space	Enable Selection ⇒ Relax Constraint	Conflicting Suggestion ⇒ Agreement on common suggestion

TABLE 3-1: **Conflicts: Partners, Causes and Resolution**

gives an overview, but it shows how rich the interactions between the different agent types are. The grayed squares show conflicts that are not yet supported by the SINE system. The outlined squares, however, show places where multiple types of conflicts exist between the same pair of agents.

Let us now investigate individual pairs of conflicting agents. For each conflict, we describe the object of the conflict, the cause, the strategy that could be used to solve it and an explanation of why this conflict can occur. We also mention whether the SINE platform currently supports the conflict type or not.

3.3.1 Estimator related Conflicts

- Conflicts with Evaluator type agents:

Object:	Not enough information available for Estimation.
Cause:	The evaluator's output doesn't provide the kinds of data that the estimator needs.
Strategy:	Try to come up with richer results from the evaluator.
Explanation:	The evaluator produced an evaluation, which the estimator wants to use. The evaluation, however, does not provide the right kind of information. This could happen, for example, if the evaluator specifies the result in the wrong units.
State:	Not supported yet.

Object:	Estimator cannot produce an accurate or reliable enough output, due to non-accurate output from the evaluator.
Cause:	The evaluator was using only a rough model, only some of the information available, or estimated values which are not reliable or exact enough.
Strategy:	Switch to a more detailed model or use better quality information.
Explanation:	When an evaluator uses a rough approach with default values, instead of the final design values, this might not be good enough for an estimator.
State:	Not supported yet.

- Conflicts with Selector/Advisor type agents:

Object:	Not enough information available for estimation.
Cause:	Some information to base the estimation or evaluation on is not available for the current choice.
Strategy:	Select an object which can be evaluated better, i.e. which has that information.
Explanation:	The selector or advisor decided on a value. The estimator does not have enough information to perform its estimation. This could happen, for example, if the estimator does not know the material that got selected.
State:	Supported.

3.3.2 Evaluator related Conflicts

- Conflicts with Estimator type agents:

Object:	Not enough information available for Evaluation.
Cause:	The estimator's output doesn't provide the kinds of data that the evaluator needs.
Strategy:	Try to come up with richer results from the estimator.
Explanation:	An evaluator might need more information than just a single value result from the estimator. For statistical values in particular, a range/ probability pair is usually needed. If the estimator fails to provide that kind of information, the evaluator might not be able to perform its function.
State:	Supported.

Object:	Evaluator cannot come up with an accurate or reliable enough output, due to non-accurate output from the estimator.
Cause:	The estimator was using only a rough model or only some of the information available, not all.
Strategy:	Switch to a more detailed model or use more information.
Explanation:	When an evaluator has to produce a detailed evaluation, it might need detailed estimates from the estimator.
State:	Supported.

- Conflicts with Selector/Advisor type agents:

Object:	Not enough information available for evaluation.
Cause:	Some information to base the estimation or evaluation on is not available for the current choice.
Strategy:	Select an object which can be evaluated better, i.e., which has that information.
Explanation:	The selector or advisor decided on a value. The evaluator does not have enough information to perform its estimation. This could happen, for example, if the evaluator does not know the material that got selected.
State:	Supported.

3.3.3 Selector related Conflicts

- Conflicts with other Selector/Advisor type agents:

Object:	Different Selection.
Cause:	Use of a model that is too shallow.
Strategy:	Abandon recommendations from the selector that uses the shallow model.
Explanation:	If two selectors with the same target and point of view clash, and if one of them uses a shallow model, while the other one uses a detailed model, it is advisable to withdraw the recommendations that are only supported by the selector with the shallow model. This conflict type has been adapted from [Klein & Lu 1990].
State:	Not supported yet.

Object:	Different Selection.
Cause:	Use of default values, lack of information.
Strategy:	Replace invalid default values with real values and fill in knowledge gaps.
Explanation:	If the difference in choice is based on the fact that one of the selectors has wrong information, or lack of information, then this can be resolved by updating his knowledge. This conflict has been adapted from [Werner 1989].
State:	Not supported yet.

Object:	Differing Selection.
Cause:	Differing Knowledge of Alternatives.
Strategy:	Inform other agent of additional alternatives.
Explanation:	This is a subclass of the previous conflict type, where the reason for the conflict is that the selectors have different lists of alternatives to begin with. This could happen in a distributed system, due to lack in information transport, or because one of the agents has a different alternative-query method. Usually, this conflict can be resolved by mutually informing the other agent of the additional choices.
State:	Not supported yet.

Object:	Different Selection.
Cause:	Differing Preferences.
Strategy:	Agree on common value, or convince other agent to abandon choice in favor of more important/global goals.
Explanation:	If two selectors differ, because they have different preferences, even when all the information is the same, they can only try to agree on a common value, or try to get the other agent into accepting their own. This conflict is the one most commonly found in the literature, e.g., in [Lander & Lesser 1991] and [Victor 1993].
State:	Supported.

Object:	Different Selection.
Cause:	Different Constraints.
Strategy:	Build intersection of both agent's possible choices.
Explanation:	If the reason for the disagreement is that the agents choose from different supersets of alternatives, then it is advisable to build the intersection of the two sets and resolve it the same way as the previous conflict type.
State:	Supported.

- Conflicts with estimator and evaluator type agents:

Object:	Not enough information available for selection.
Cause:	Some criteria pertinent to the selection process are not available for that object.
Strategy:	Add needed information into the evaluation/estimation result.
Explanation:	The estimator or evaluator decided on a value. The value alone might not provide enough information for the selector (e.g., for statistical values: an estimated value, a reliability factor and an error tolerance always go together).
State:	Not supported yet.

Object:	Alternatives don't show enough difference in quality.
Cause:	Evaluation or estimation model is too weak to produce significantly different results for the input, maybe due to too high tolerances.
Strategy:	Improve estimation or evaluation contrast.
Explanation:	The selector / advisor can use feedback from evaluators or the estimators to rank its possible choices. If the ranking is not possible, because the values do not show enough difference, then the selector or advisor may request a more critical evaluation or estimation.
State:	Supported.

3.3.4 Critic/Praiser related Conflicts

- Conflicts with estimator and evaluator type agents:

Object:	Not enough information available for Criticism.
Cause:	The estimator's / evaluator's output doesn't provide the kinds of data that the critic needs.
Strategy:	Try to come up with richer results from the agents.
Explanation:	The estimator or evaluator decided on a value. The critic does not receive enough information to perform its criticism. This can happen for example with statistical estimates, when they do not provide the needed all of the value, the statistical reliability and the error margin.
State:	Not supported yet.

Object:	Not enough information gets used in estimation / evaluation.
Cause:	The critic wants to make sure that all the pertinent information is used.
Strategy:	Try to add additional design parameters, goals, preferences or standards into the estimation / evaluation data.
Explanation:	Critics can be used to ensure that a certain level of quality of work is performed by the other agents. If an agent does not use all the pertinent information, this can prompt the critic to request from the agent in question to use more information in his approach.
State:	Not supported yet.

Object:	Poor processing model.
Cause:	Agents were only using a rough model to come up with a quick answer.
Strategy:	Switch to a more detailed model.
Explanation:	Critics can be used to ensure that a certain level of quality of work is performed by the other agents. When an agent uses a quick and shallow approach (e.g., during rough design), this might not be acceptable in later design stages. The critic can then request the other agent to use a more thorough approach.
State:	Supported.

- Conflicts with selector type agents:

Object:	Unsatisfactory Selection.
Cause:	Constraint violated.
Strategy:	Inform Selector of constraint, try to select again, with the additional constraint.
Explanation:	When a specific constraint fails for a particular selection, and the critic has reason to believe that the selector could interpret the constraint itself, then the critic can decide to request from the selector to incorporate his failing constraint into the selector's design knowledge.
State:	Supported.

3.3.5 Suggestor related Conflicts

- Conflicts with other suggestors:

Object:	Different Suggestion.
Cause:	Model too shallow or too general.
Strategy:	Abandon recommendations from the Suggestor that uses the shallow or general model.
Explanation:	Similar to the selector-selector conflicts. Two suggestors might differ on their suggestions about what to do next. If one of them uses a shallow or general model and the other one employs a more detailed or more specific model, then the simpler or less specific suggestor's solution should be abandoned in favor of the more specific one.
State:	Not supported yet.

Object:	Different Suggestion.
Cause:	Default Values or lack of information.
Strategy:	Replace invalid default values with real values and fill in knowledge gaps.
Explanation:	If the difference in suggestion is based on the fact that one of the suggestors has wrong information, or lack of information, then this can be resolved by updating its knowledge.
State:	Not supported yet.

Object:	Different Suggestion.
Cause:	Different Preferences and/or different goals.
Strategy:	Agree on common values, or convince other agent to abandon goal in favor of more important/global goals.
Explanation:	If two suggestors differ, because they have different preferences, even if all the information is the same, they can only try to agree on a common value, or try to get the other agent into accepting their own.
State:	Not supported yet.

- Conflicts with other agent types:

Object:	The suggestion cannot be implemented.
Cause:	The other agents do not know how to apply the suggestion to the situation.
Strategy:	Inquire possible alternatives from agent and operationalize suggestion accordingly.
Explanation:	If the suggestor makes a proposal which cannot be implemented, because the agents in concern are unable to instantiate the proposal, then the suggestor can be asked for a more specific description or a more appropriate suggestion.
State:	Not supported yet.

Object:	The suggestion can be implemented but produces problems.
Cause:	There are constraints that fail when other agents try to apply the suggestion.
Strategy:	Find a different plan that avoids the conflicts or modify plan to satisfy constraints.
Explanation:	A suggestor cannot always anticipate all the possible difficulties in implementing the suggestion. It might be prompted to refine or adjust its proposal, if not to withdraw it and suggest a different solution.
State:	Not supported yet.

3.4 Conflict Types

Although there is the possibility for a multitude of conflicts between SiFAs, they fall into a limited number of categories. The conflict types recur in several agent type combinations:

- *Not enough Information in Output*

  This conflict type occurs, when one agent needs information from another agent and the information provided is less descriptive than the first agent needs. The conflict can be found with all agents that use output of another agent, especially the pairs Evaluator-Estimator, Evaluator-Selector, Selector-Estimator, Selector-Evaluator, Critic-Selector.

- *Information Quality not sufficient*

  Among those situations where an agent uses another agents' output, the agent that uses the information might have quality requirements, e.g., reliability or error tolerance values. Agent pairs where this type of conflicts is found are Evaluator-Estimator, Selector-Evaluator, Critic-Estimator, Critic-Evaluator.

- *Poor Processing Model*

  This conflict type can only occur when an agent has knowledge about another agents' different processing models. An agent might have a rough and a detailed model for devising a value. Although knowledge about these processes can be acquired through communication, it is mostly used by critics, suggestors and selectors. The selector for example might request the evaluator to use the detailed model, so that the evaluations are more usable for the selection purpose.

- *Differing Preferences*

  When several agents compete for assignment of the same design parameter or general issue, they can have differing local optima. Two selectors can have different first choices, but also two suggestors may want to suggest a different solution approach.

- *Design Constraint Violation*

  Violations of constraints relating to values of parameters are most commonly discovered by critics and are produced by selectors and advisors, because they are the agents making the design decisions. But a suggestion can also violate an agents' constraints.

3.5 Conflict Resolution

Conflicts can be solved in several ways, depending on the conflict type:

- *Add needed Information.*

  When an agent needs more information in the output, the output producing agent can try to fill in the kinds of data that have been requested. Often, the agent will not be able to produce the values. In those cases, the producing agent will have to supply default values, or the information using agent will have to make default assumptions.

- *Improve Information Quality.*

  Information quality is often dependent on the amount of research done. For estimators especially, statistical information can sometimes be improved by performing a more thorough analysis of the data. This will produce a higher reliability and/or less error margin in the estimate. If a better information quality cannot be achieved, the agents that use the data might have to be more careful in their use. For evaluators this can mean that they produce a less drastic difference between their best and worst evaluations. For selectors this can mean that they do not select alternatives that are characterized by a high degree of uncertainty.

- *Change to a Better Processing Model.*

  In many cases a poor or shallow model has only been used to allow quick prototyping or rough design. In the detailed design phase these models should be abandoned in favor of better models, which use the additional information available at that point.

- *Agree on Common Value.*

  Depending on the strategy (see 3.6, “Negotiation Strategies”) an agreement can be reached through different means. However, in the course of the design, revision and further agreements might be needed. Agents should be flexible enough to react to changed situations, but intelligent enough to avoid redundant negotiation and vicious circles. This means that they might have to keep historical information, which will avoid asking another agent to perform what it cannot do or what it already has done.

3.6 Negotiation Strategies

For any conflict with more than one possible outcome, some way of driving the agents’ behaviors will have to be available. These strategies are different in the behavior that they produce and they can have an impact on the portability to a new domain.

Agents can have a different strategy, depending on behavioral aspects, which we can call ‘character traits’. For example, the agents might give in, whenever they can suit the requesting agents needs, insist on their previous behavior/value, or negotiate and adapt to requests, possibly by trading knowledge and resources as they do.

Strategies can be domain dependent or domain independent. Usually, the domain dependent strategies allow a direct solution of the conflict, but the strategy cannot be transferred to a new design domain, without modification. The reason is that the strategy might include knowledge about domain specific agent-interactions, dependencies of values, and/or aspects of parameters.

Another aspect of strategies is whether they can be applied to any kind of agent-pair or whether they are specific to a certain grouping, for example selector-estimator.

3.7 Knowledge Requirements

Agents need mostly two kinds of problem-solving knowledge: Design Knowledge and Negotiation Knowledge. The design knowledge allows them to perform their function and it is described in the section 3.2, “Agent Types”, and section 4.4, “Agent Design”.

In order to solve conflicts, the agents need the negotiation knowledge. It consists of several parts. One part has to analyze what other agents have lead to the current inconvenience. This is known as blame assignment, and in SiFA systems it consists of tracing origins of values or using knowledge about value dependencies to find the ‘culprit’.

The information from the first part is used in the second part, analyzing the conflict type. This usually involves tracing historical information and/or asking the other agent questions, in order to classify the conflict into one of the conflict types in section 3.4, “Conflict Types”. In SINE, agents can also use their knowledge about agent structure to gather information about the agent type that the stand in conflict with.

3.8 Functional Requirements

For the negotiation, the agents need:

- *Communication Functions*

  These features facilitate the locating of agents and transmitting messages to them. In distributed systems the messaging can be layered on top of existing communication channels (TCP/IP, RPC, IPC, email). In communication environments based on KQML, information brokering functions might be available. The agents need functionality to use these for their negotiation purposes.

- *Functionality for finding Conflict Solutions*

  In negotiation the agents need to be able to find possible points of agreements, choosing the best one of them and signaling them to the other agents. These functions will take into account the strategies and conflict types. They also involve recognizing what proposals are possible solutions and making proposals to the other agents in a way that keeps the negotiation short and efficient.

- *Functions for Implementation of Solutions*

  Once a common solution has been agreed on, the agents will need functionality to implement it. For some agents that might involve redeciding their values and often it includes adding, relaxing or removing constraints, or activating different rule-sets and functions.

3.9 Summary

In this chapter we investigated Single Function Agents. We started by defining the individual agent types in terms of their functions, inputs and outputs (section 3.2). We then continued with an analysis of where conflicts could occur between the different agent types, by building a matrix of all agent types and investigating each combination (section 3.3). In the same section, we described the possible conflicts for each agent type, with the causes, strategies and explanations pertinent to the conflict.

We then abstracted the conflict occurrences into a set of five conflict types (section 3.4) and built a similar set of negotiation strategies (section 3.5). We concluded the chapter by deriving the knowledge and functional requirements (sections 3.6 and 3.7, respectively) for the agents from these two sets.

In the next chapter, we present SINE, a platform for research into negotiating SiFAs.

CHAPTER 4

SINE: A Platform for Negotiating SiFAs

4.1 Introduction

In this chapter we will present SINE, a platform for research into negotiating SiFAs. We first describe the goals that are to be fulfilled with this platform and explain the reasons for its development.

In the second section, we present the architecture that SINE uses, including the knowledge representation, the data and control flow, as well as the language and communication that we use. We also present the methods used for conflict detection and notification, the conflict resolution model and the scheduling system.

The third section explains the design of the individual agent types.

4.2 Goals

After the development of several SiFA systems at WPI, there was considerable evidence that some of the structure and functionality was recurring. In SNEAKERS and I3D, for example, we find many identical agent types. Also, with the implementation of negotiat-

ing SiFA systems it became obvious that it is inefficient to design an entire negotiating SiFA system from scratch. We therefore decided to build a platform with the following goals:

- *Build a reusable SiFA system for the future*

  The system should allow easier development of SiFA based systems, especially for parametric design systems. A large amount of domain-independent design functionality should be made available. Also, the system should provide a programming environment to the developer which allows the development on an abstract, high level. For example, it should provide support for explicit data representation and writing the development of the design knowledge independent of the data that get used in the system. This would prevent hard coding data in the design knowledge, which has been found to be difficult for software maintenance and extension.

- *Implement domain-dependent and domain-independent CR knowledge*

  In I3D+, only domain dependent negotiation knowledge was implemented. A universal platform would have to supply some general CR rules and strategies that could be used as a background for implementing more specific techniques in critical areas. Then the domain independent part could be used for the bulk of the occurring conflicts.

- *Experiment with conflicts and their resolution in a CE system*

  As not all the aspects of negotiation in SiFA systems had been researched in previous work, the platform should allow experimenting with different knowledge, strategies and conflict situations.

- *Design and implement a negotiation language for SiFAs*

  A reusable language for negotiation would allow present and future agents to be compatible and set a standard for SiFA communication, which up to now did not exist.

4.3 Architecture

One of the challenging tasks in developing SINE was to devise an architecture that would suit the needs of present and future SiFA systems in multiple domains. As building a totally universal system would have been difficult, the SINE platform is geared mostly toward routine parametric design. This means that the structure of the design object, its features and the parameters influencing it are known in advance.

The design task will consist of making decisions on values for design attributes, which can be either parameters (lengths, widths, colors, materials etc.) or general design decision (e.g., selecting a structure type for the artifact or determining the class of material to use). This also allowed a more function specific and efficient design for the agents, while still allowing routine conceptual and configurational design.

4.3.1 Agent Topology

The agent topology defines what agents exist and where interaction between them occur. Many kinds of topologies are possible, such as star, ring, multi-star and hierarchy. The SINE system adopts a flat, totally connected structure. This means that all agents are able, if necessary, to communicate with all others, and no agent has a hierarchical priority over any other agent.

The advantages are that this most closely represents a human team at work. While some agent could take on a controlling function, the system does not enforce any kind of behavior on the agents. Also, agents can be developed, tested, added and removed without considerations about the rest of the system. They could in fact randomly join and exit the design process.

A possible disadvantage is that this structure produces an exponential number of possible conflicts and negotiations for very large design tasks. As analyzing all the possible connections was one of the goals for the research, this is an advantage for research. The flat agent topology could be extended to support some agent hierarchy in future research.

4.3.2 Data Flow and Negotiation Links

For efficiency and coherency reasons, the agents design the artifact on a centralized blackboard, i.e., all the information about the design parameters is located at one site. When an agent decides on a design parameter or when it retrieves information about the design, the information is sent to or retrieved from this blackboard.

Figure 4-1 shows how the agents store and retrieve information about a design parameter

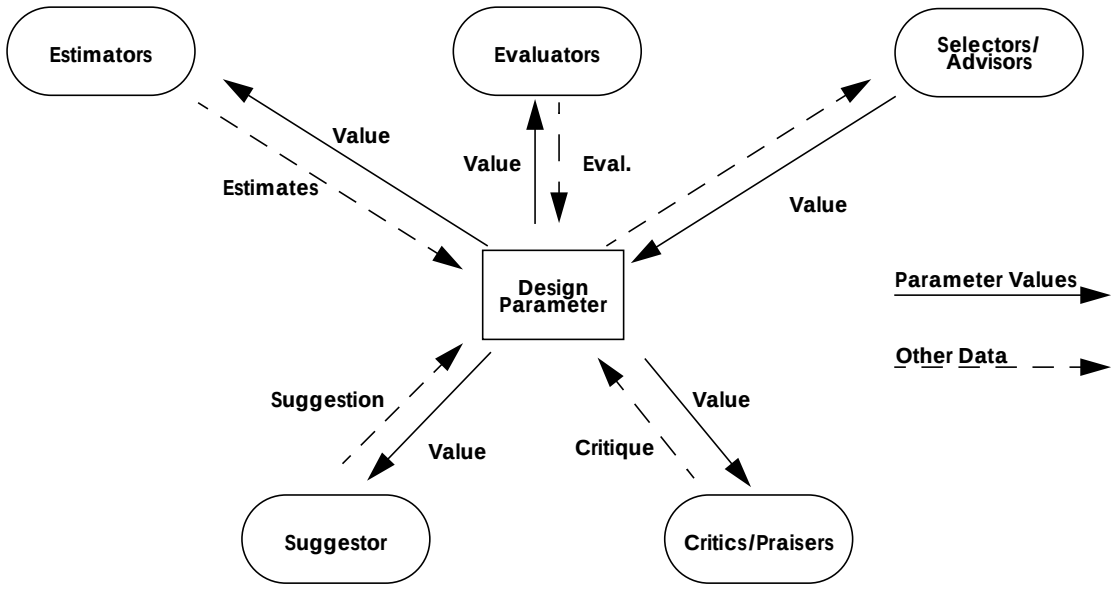

FIGURE 4-1: **Data Flow in SINE**

from the central blackboard. If no negotiation occurs, the data flow is star shaped.

Even though this removes some of the possibilities that a fully distributed system would offer, the advantages of having a shared location for the design description outweighed the disadvantages in the design of SINE. This is due to the fact that a shared memory prevents knowledge inconsistencies due to propagation latencies, and it also makes retrieval of information about the artifact more efficient. A large portion of research in DAI has analyzed the implications of knowledge distribution, so there is no goal for SINE to replicate these insights [Bond & Gasser 1988] [Huhns 1987] [Huhns & Gasser 1989].

The interactions during negotiation are much more complex. Apart from using the central blackboard, agents also use their communication features to request and send information to other agents. Figure 4-2 shows all the possible negotiation links between agents. A key

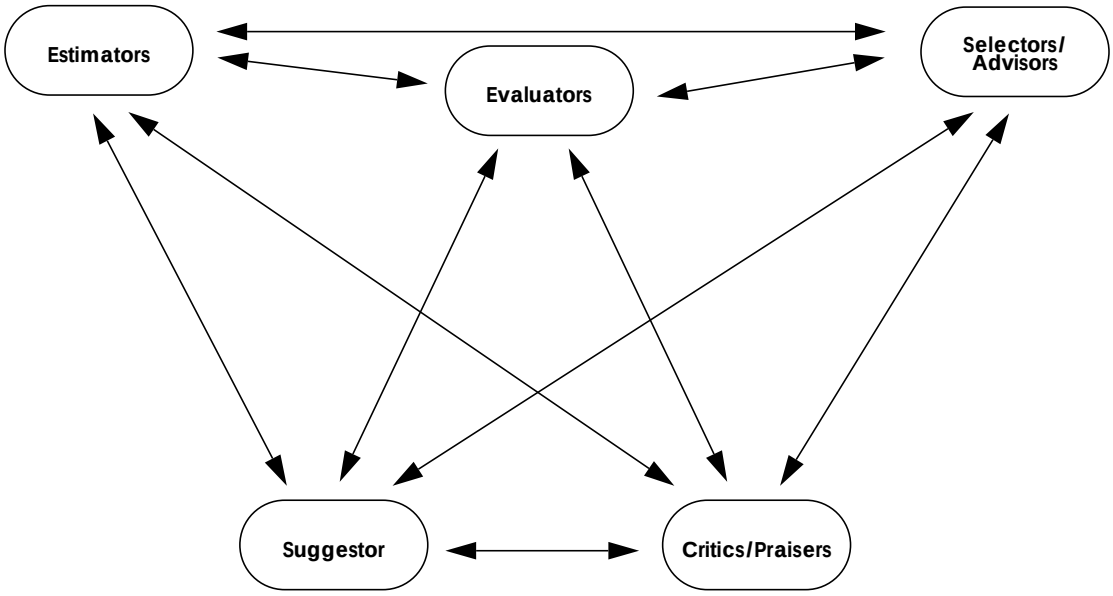

FIGURE 4-2: **Negotiation Links in SINE**

part of the research is to investigate these links and their use. The platform supports communication between any pair of agents.

The data flow and the negotiation occur intermittently, with agents using the blackboard to retrieve information and post the results of the negotiation.

4.3.3 Knowledge Representation

The SINE system uses a combination of rules and reactive objects to store knowledge. Reactive objects, like all objects in an object oriented design, have some data content, e.g, a simple or structured value, and some functions that allow access and usage of the data. The objects are generated as instances of classes. The classes are structured in a hierarchy. Functions that the objects provide, as well as structure and default values can be inherited.

A *design parameter,* for example, has a ‘value’ slot and an ‘origin’ slot. The ‘value’ slot stores the current value of the design parameter, the ‘origin’ slot holds the source of the parameter, i.e., which agent decided it. The parameter class also provides three functions to access its objects: one read-access function, and two write-access functions, one with and one without restriction.

The value of a design parameter could be a numeric value, such as a length in millimeters, weight or volume. It could also be a reference to a structured value, such as a material that includes information about density, thermal conductivity, cost etc., in attribute value pairs.

Agents can use *constraints*, which are represented as objects of a constraint class. There are three kinds of constraints:

- Constraints on values, e.g., (> length 3), meaning “the length has to be larger than 3”.
- Constraints on the existence of values, for example (≠ cost nil) to express “The cost of the material has to be known.”
- Constraints on the interaction of values, e.g., (> length width), which could mean that the length of the artifact has to be larger than the width.

For agents that need to make a choice between different alternatives, there is a class called *preference*. Its instances store an abstract relation. These can be used to produce a (partial) ordering on a set of values or choices. For example, an agent could have a 'low_cost_preference' with an ordering expression stored in it. The agent can then send the names of two alternatives to the preference object, with the request to have them ordered. The preference object can use an inherited mechanism to order the alternatives by increasing cost. It can also use a built in iterative mechanism, which will sort a list of alternatives by applying the preference operator repeatedly.

Requirements are specified in two parts. The first part consists of the rules, constraints and preferences. It is thought to change only occasionally, with the addition of new knowledge into the system, or modifications of the design task or design artifact structure.

The second part of requirement specifications consists of filling slots in a requirement structure, similar to design parameters, with values. These values, together with the constraints, rules and preferences, drive the agents in the design.

For a material cost critic, for example, the maximum total cost is specified as a value in the material requirement structure, and can be changed before any run or derived from user input, while the 'lower-than'-constraint is part of the agents knowledge base.

4.3.4 Communication

The communication structure adopted by SINE reflects the flexibility necessary for research into negotiation. As all agents can negotiate with all other agents, the communication supports direct addressing of agents.

The communication mechanism implements synchronous messages, i.e., when agent A sends a message to agent B, B is given control to process the message immediately, while

A has to wait for B to return control to it. Synchronous messaging might seem like a restriction, but since the receiving agents are free to simply store the message, without doing anything else, an asynchronous messaging system can easily be built on top of the existing scheme.

The message format is based on speech acts (see section 2.5.2, “Speech Acts”), i.e, it cap-

Example

- primitive tell
- subject proposal-count
- parameter1 nil
- parameter2 nil
- in-reply-to [msg14]
- language sifa_language
- ontology sifa_ontology
- receiver [mts]
- reply-with [msg15]
- sender [mbs]
- timestamp 12

FIGURE 4-3: **Speech Act Example**

tures a complete utterance in one message. The utterances are similar to a human sentence, which allows a straightforward adaptation of sentences found in human discussion. The format is similar to the KQML message structure, with slots for storing the primitive, sender, receiver, time stamp, language and ontology information, as well as references to previous and future messages [Finin et al. 1993]. However, the KQML content slot has been split apart into a more specific structure, consisting of a subject and two parameter slots. These can be filled with standard names of design parameters and other names of objects and attributes.

How the primitive, subject and parameters can be filled to form meaningful sentences is defined by the SINE communication language, which we present in the following list. We will use some elements of BNF (Backus-Naur-Form) for representing logical relationships in the description, so that | stands for logical ‘or’, ‘{’and ‘}’ mark the beginning and end of a set, ‘*’ means zero or more elements, ‘+’ means one or more elements, ‘::=’ means ‘is defined as’, and ‘<*name*>’ represents a placeholder.

- *Alternatives:*

  All agents that produce values by selection, or by elimination from a set of possible choices (e.g. Selector, Advisor, Suggestor), should be able to communicate their alternatives (i.e., their different possible choices). Selectors have a finite input set of alternatives, which can then be filtered through constraints and sorted by preferences. Other agents might request a selector to consider an additional option or to refrain from using an alternative.

“Add the following alternative to your list of considered choices.”

primitive	insert
subject	alternative
parameter1	<alternative-name>
parameter2	nil

“Remove the following alternative from your list of considered choices.”

primitive	delete
subject	alternative
parameter1	<alternative-name>
parameter2	nil

- *Constraints*

  Many agent types use constraints. Selectors restrict their set of choices with them, Critics use them to trigger specific messages and other agents can check their preconditions with constraints.

  Communication about constraints can help agents understand reasons for conflicts. It can also help them avoid conflicts in later stages, if they manage to incorporate other agents' constraints into their considerations.

  Constraints can have one of three states: failed, satisfied or unknown.

  <c-status> ::= {failed | satisfied | unknown}

"What are your (all, satisfied or unsatisfied) constraints?

primitive	ask-about
subject	constraint
parameter1	<all | <c-status>>
parameter2	nil

"My constraint is/are"

primitive	tell
subject	constraint
parameter1	{<constraint-name> <c-status>}$^{+}$
parameter2	nil

"What is the status of the following constraint?"

primitive	ask-about
subject	constraint_status
parameter1	<constraint-name>
parameter2	nil

“The constraint is satisfied/unsatisfied”

primitive	tell
subject	constraint_status
parameter1	<constraint-name>
parameter2	<c-status>

“Add the constraint to your list of things to consider”

primitive	insert
subject	constraint
parameter1	<constraint-name>
parameter2	nil

“Remove the constraint from your list”

primitive	delete
subject	constraint
parameter1	<constraint-name>
parameter2	nil

- Preferences

  Agents that have to search a set of choices for the best option can use preferences for the purpose. Passing information about preferences can be used to locate reasons for differing choices, in situations where the same data are known to both agents.

“What are your preferences?”

primitive	ask-about
subject	preferences
parameter1	nil
parameter2	nil

“My preferences are...”

primitive	tell
subject	preference
parameter1	<preference-name>$^{+}$
parameter2	nil

- Proposals

  A value that an agent asserts for a design parameter is considered a ‘proposal’, since it can be contested by other agents through negotiation. This is probably the most often used type of subject. The language reflects this importance by offering a large variety of speech acts for proposal communication, including agreement, disagreement, counter-proposals, queries as to the number and the values of possible proposals, etc.

“I agree with your proposal for design-value x.”

primitive	tell
subject	proposal
parameter1	agree
parameter2	nil

“I don’t agree with your proposal for design-value x.”

primitive	tell
subject	proposal
parameter1	disagree
parameter2	<reason>

“Here is my counter-proposal”

primitive	tell
subject	proposal
parameter1	<value>
parameter2	nil

“How many different proposals do you have?”

primitive	ask-about
subject	proposal_count
parameter1	nil
parameter2	nil

“I have x proposals.”

primitive	tell
subject	proposal_count
parameter1	<value>
parameter2	nil

“Give me another/all possible proposals(s)”

primitive	ask-about
subject	proposal
parameter1	{one | all}
parameter2	nil

“Here is another proposal / are all other proposals”

primitive	tell
subject	proposal
parameter1	<object-name>$^{+}$
parameter2	nil

“I accept your counter-proposal x”

primitive	tell
subject	proposal
parameter1	accept
parameter2	<value>

“I reject your counter-proposal x”

primitive	tell
subject	proposal
parameter1	reject
parameter2	<value>

- Data

  In order to decide whether a conflict exists due to the lack of knowledge, agents can inquire what data other agents are using.

“What data are you using?”

primitive	ask-about
subject	data
parameter1	nil
parameter2	nil

“Do you have knowledge of x?”

primitive	ask-if
subject	data
parameter1	<data-name>
parameter2	nil

“I use the following data: ....”

primitive	tell
subject	data
parameter1	<data-name>*
parameter2	nil

“What is your value for data-item x?”

primitive	ask-about
subject	data-value
parameter1	<data-name>
parameter2	nil

“Try to get information about data x (with attribute <data-value>)!”

primitive	achieve
subject	data
parameter1	<data-name>
parameter2	<data-value>

“I have the value y for data-item x”

primitive	tell
subject	data
parameter1	<data-name>
parameter2	<data-value>

- Processes

  Agents may have different ways of performing their functions. These can be tagged with names. Then, they can exchange information about their processes (e.g., with a Critic).

“What process are you using?”

primitive	ask-about
subject	process
parameter1	current
parameter2	nil

“I use the following process:....”

primitive	tell
subject	process
parameter1	<process-name>$^{+}$
parameter2	nil

“What process can you use / could you use?”

primitive	ask-about
subject	process
parameter1	{all-applicable | all}
parameter2	nil

“Do you have knowledge of process x?”

primitive	ask-if
subject	process
parameter1	<process-name>
parameter2	nil

“Please use the process x”

primitive	tell
subject	request-process
parameter1	<process-name>
parameter2	nil

“I accept your request to use process x”

primitive	tell
subject	request-process
parameter1	accept
parameter2	<process-name>

4.3.5 Conflict Detection and Notification

Selection-Selection Conflicts are detected when an agent tries to assert a value for a design parameter. When it sends a request to the parameter to store a value in a parameter which is different from the previously stored one, it will get a negative reply.

Value-Usage Conflicts can be detected when an agent checks its preconditions before it gets run. If a design value does not satisfy all its precondition constraints, the failure of the appropriate constraint is its indication of conflict.

Value-Criticism Conflicts can also be detected through constraint failure If the critics have a set of constraints that they evaluate during their run and one of them fails, this is an indication of a conflict. In addition to that, the critics can have domain dependent knowledge (e.g., in rules) that indicates conflicts in other situations.

After recognizing the fact that a conflict exists, the agent uses a built-in function to assert the fact that a conflict exists. The conflict information gets posted in the global database and can be seen by all the agents. It holds information about what type of conflict has been detected, which agent discovered it, what agents are participating in the conflict (e.g. the agent that proposed the original value) and when it was discovered.

4.3.6 Conflict Resolution

After posting the conflict information in the central database, the initiating agent starts the conflict resolution. It is possible that the agent has a separate CR module, which concentrates all the CR knowledge in one place. These modules can be inherited from the agents' parent classes. A material selector, for example, can inherit CR knowledge from the selector agent class.

How exactly the agent solves its conflict, what questions it asks and messages it sends, is up to the developer of the agent, as long as the agent uses the SINE language for the communication. It is not necessary that all agents have the same CR knowledge.

However, a convenient way to solve a problem is to have the initiating agent iterate through several questions, in order to determine the current state of its partner in the conflict. It may then proceed to requesting the other agent to produce a new value or, if it receives an indication that this is the only value that the partner can accomplish, try to adapt to the given value itself.

The agent behavior can be guided by strategies and general behavioral attributes. An agent may be set up to give in to other agents' request very quickly or very slowly. It might be designed to share only a minimal amount of information or as much as it has at any point.

Apart from these behavioral issues, there are also some more technical issues that have to be decided. For example, the amount of information that is retrieved from previous confrontations with other or the same partner has to be determined. Reusing this information can reduce the amount of conflicts that occur, as well as the message traffic, but the information might also be outdated, due to intermediary design changes. Reusing too much knowledge can therefore overconstrain the design.

The agent is also responsible for recognizing that the conflict has been solved or that no solution is possible. Although this might seem difficult, in many cases one of the two agents will make a proposal to which the other one agrees. This agreement message can be used to automatically trigger a functionality in the agent which removes the conflict information from the blackboard.

Otherwise, if the agent has tried all its possible solution steps without success, there is apparently no feasible solution. In that case, a different strategy or plan might have to be used, the CR might have to be postponed, or other agents might have to be taken into the resolution process.

4.3.7 Agent Scheduling and Control Flow

The SINE platform has a scheduling mechanism which maintains a list of agents that should be run next. This agenda manager is the top level controller of the system. It goes through a loop of steps, until no agents can be scheduled anymore:

1	Request all agents to propose agenda entries
2	Evaluate all the proposals with their reasons and tag them with a numerical value indicating the importance
3	Sort all proposals by importance
4	Execute the most important task

TABLE 4-1: **Agenda Manager**

An entry is proposed by an agent by sending a message to the agenda manager. This message will contain the agent's name, a reason, explaining why the agent wants to be scheduled and a textual description of the task the agent wants to perform, for reference by the user or developer. The reason can be one of the following (in increasing importance):

- usage (An agent is ready to use values)
- selection (An agent is ready to select a value)
- criticism (An agent wants to criticize) or
- conflict (An agent has encountered a conflict)

When no agent proposes an entry, the agenda manager assumes that the design is either finished or stuck unrecoverably, and it terminates, returning control to the user.

In the case of a conflict, the agenda system is inactive. Instead, the communication mechanism that passes the information back and forth between the agents ensures that each agent receives control to be able to evaluate and react to the message (see Figure 4-4).

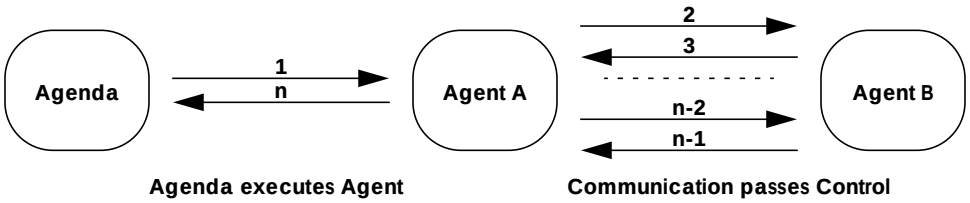

FIGURE 4-4: **Control Flow during Negotiation in SINE**

4.4 Agent Design

An agent in SINE consists of four major parts (see Figure 4-5):

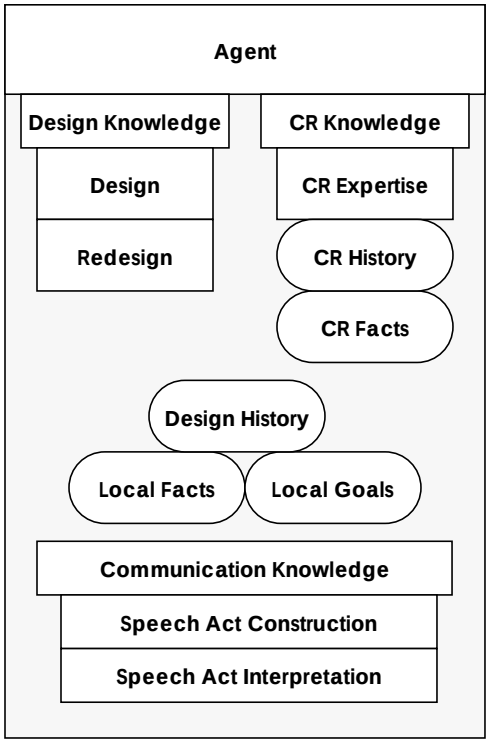

FIGURE 4-5: **SINE Agent Structure**

- Design Knowledge
- Conflict Resolution Knowledge
- Communication Knowledge
- Local Memory

The design knowledge contains all the functionality that the agent needs to perform its design task, if no conflicts arise. The knowledge can be encoded in rules or in a more functional programming language. In addition to that, there is also a small portion of knowledge devoted to redesigning, for use after or during conflicts.

The conflict resolution knowledge contains all the methods that are used to detect and classify conflicts, as well as the strategies for conflict resolution. It also contains a memory for storing historical information about previous CR interaction with other agents, such as, their goals, previous proposals that failed, preferences of other agents, etc.

The communication knowledge consists of two parts:

- Speech Act Construction
- Speech Act Interpretation

The first is used to produce and dispatch messages to other agents, the second retrieves messages that are addressed to the agent and provides immediate reaction to the message. The agent can be designed so that reactions to some messages are directly triggered through this interpretation knowledge, without involving the CR knowledge. This can improve message turnaround time.

It is generally desirable to keep actions during CR as slim as possible and to perform most of the action during the work phases, when the agent is scheduled by the agenda. Only then can the agenda system ensure a fair distribution of processing between the agents.

The connections between the different parts of the system and the agents can be seen in Figure 4-6.

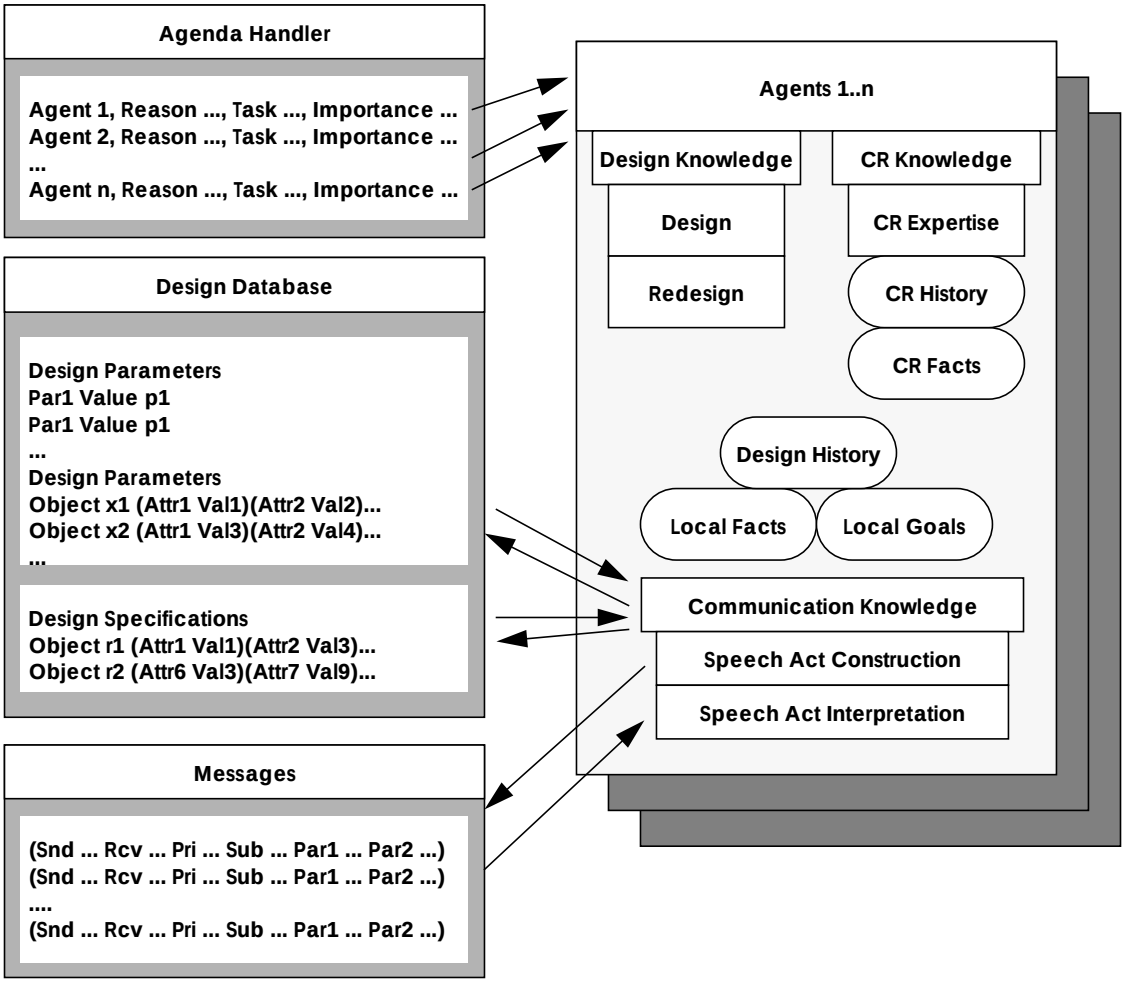

FIGURE 4-6: **The Components of SINE**

4.4.1 Selector/Advisor

The selectors and advisors hold a key role in a SiFA system, as they are the agent types that directly select values for design parameters, which involves decision making. On the other hand, the selectors are the agent type which have the most rigidly defined function. That is why the SINE platform provides a strong supporting structure for implementing selectors.

The general task of the selector type is to:

- Produce a list of all types of values that the design parameter can hold
- Remove from the list all values/object that do not satisfy the set of constraints that the selector knows about.
- Use preference operators to choose among the set of valid-choices the locally best value.
- Negotiate with other agents in case of a conflict.

For example, let us consider how a selector for material using thermal-conductivity considerations, can be designed through inheritance. The material-thermal-conductivity-selector class inherits from the following classes:

- selector function class
  This provides the agent with selection functions and agenda scheduling, as well as conflict resolution knowledge.
- material target class
  The material class provides knowledge about how to find possible choices.
- thermal-conductivity point-of-view class
  The thermal-conductivity class specifies what aspect of the target to consider, in this case it is the thermal-conductivity attribute.

Selection Process. Given only these three inheritances, the agent can already perform its function, i.e., select a value for the material parameter. It will use its materials knowledge to find all the materials and then apply its selection knowledge to pick one of them and store it in the design parameter.

However, without a preference function, the selector will not be able to optimize the choice with respect to its point-of-view. Therefore we need to add a preference object to the selector, which orders alternatives by decreasing thermal-conductivity. Now the selector can find its locally optimal choice. Figure 4-7 shows the steps that the selector takes in order to decide on a value. We will see in section 5.4, “Agent Implementation”, how this is programmed.

If the design parameter object accepts the value that the selector requests, then the agent is done, and it goes into a ‘waiting’ state until the design parameter is modified by another agent or until some agent sends it a speech act.

The selector also provides a scheduling function. When the agenda polls the agents for scheduling requests, the selector automatically requests an agenda entry, when its preconditions are satisfied. If the selector has never been run before in the current session, it will request to be scheduled for ‘selection’.

Otherwise, it remembers what its previous choice for the parameter was. If the current value for the parameter is different from its previous choice, it detects the conflict and requests to be scheduled for ‘conflict’, which will give it a higher priority.

Conflict Resolution. When the selector encounters a conflict during the selection process, it categorizes the conflict and activates a resolution strategy. For selection-selection conflicts, the selector module provides CR functionality:

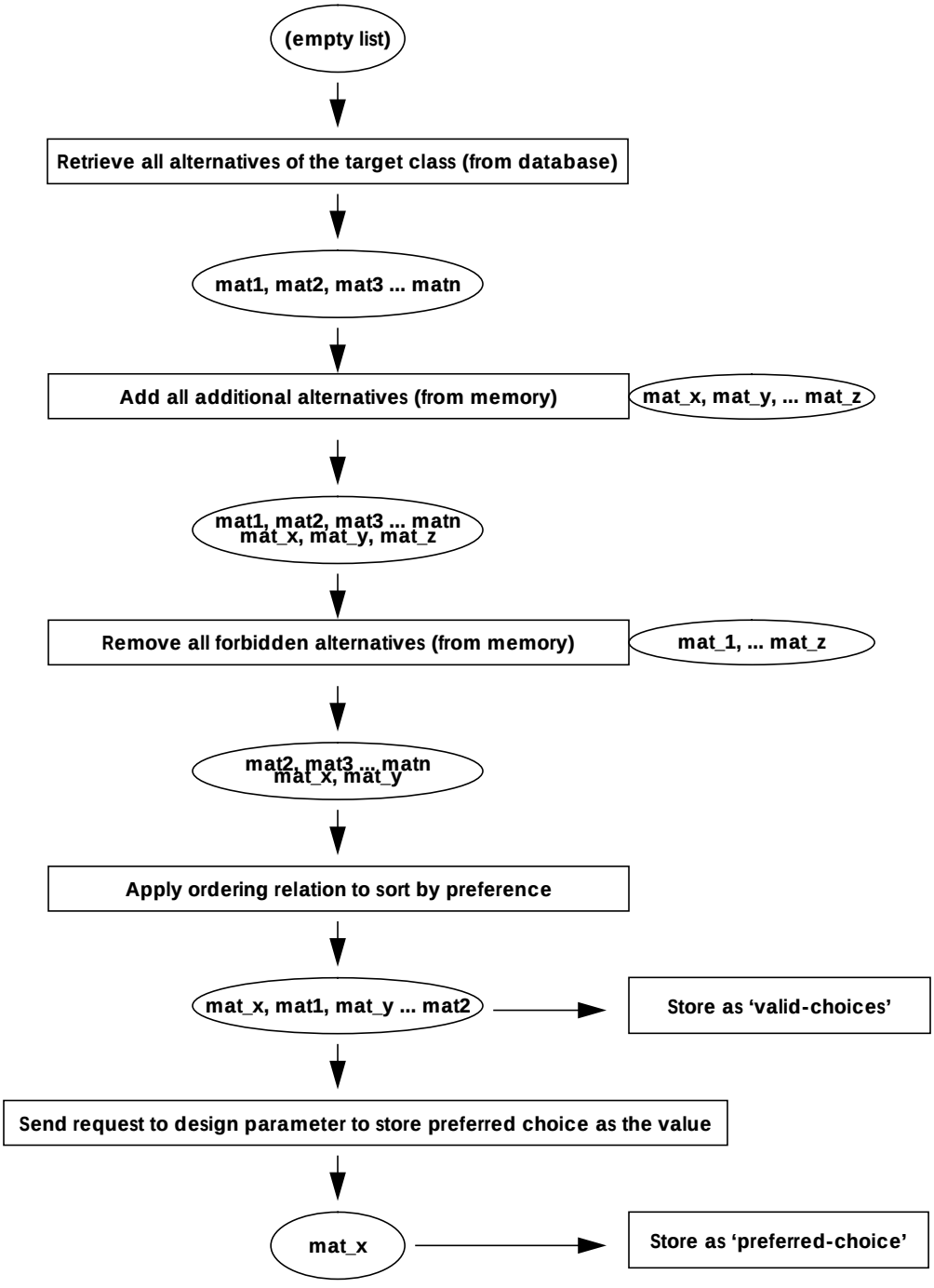

FIGURE 4-7: **Selection Process**

- The selector can ask the other selector whether it has a different proposal to offer. If both of them can agree on a proposal, the design parameter is updated to hold the new value.
- Otherwise one of the agents will have to adapt to the design value or a counter-proposal by relaxing some of its constraints.
- If this is not possible, the agents will have to block the design process, because it is over-constrained, unless their difference is due to lack of information, which can be resolved by exchanging information between the selectors.

Conflict Avoidance. The selector has three mechanisms that can help prevent conflicts:

- *Additional Constraints*

  The selector can accommodate additional constraints from other agents, that it learned through negotiation. These will help it refine its options to prevent further conflict with other agents due to that constraint. For example, if a material evaluator needs density information about the material, then this constraint can be passed to the selector to prevent proposal of materials that do not have density information associated with them.
- *Forbidden Alternatives*

  The selector can hold a list of forbidden choices. This list can be built from requests by other agents, not to use certain alternatives. It will avoid selecting those alternatives.
- *Additional Alternatives*

  A list of additional alternatives, built from descriptions passed to it from other agents, allows it to consider choices that it was not able to find by its standard method (as inherited from the target class).

4.4.2 Estimator

The estimator functionality is much more domain dependent than the selector. Our experience has shown that the estimation process need specific knowledge, such as rules, or functions, such as statistical methods. This is probably due to the fact that there is no estimation function that can be used in all cases.

However, the design of the estimator supplies some basic structures which help during the development. All estimators can have a list of precondition constraints. Unless all those constraints are satisfied, the agent will not perform its estimation. If any of the constraints fail, the estimator stores their names in memory, for use during conflict resolution. If it can blame a particular agent for the failure (e.g. a selector), it may proceed to requesting that agent to incorporate the failed constraint to prevent further conflicts. Alternatively it may choose to request from the other agent not to propose the value that lead to the failure anymore.

4.4.3 Evaluator

An evaluator can use evaluation functions to derive the quality of the current design choices. These functions can be stored in objects of the quantified_preference class. A quantified preference is a preference, with the added possibility of numerically or symbolically expressing the quality of any possible choice.

For example, we can build an absolute preference for material strength, such that any material strength can be compared to a scale from 0 to the maximum required strength (from the design specifications). If the strength is 1/2 of the maximum, the function returns 0.5. If the strength is at the maximum, it returns 1, if it is closer to 0 it returns smaller values.

Given a list of such preferences, the evaluator can iterate through them, evaluating each individual preference and posting the results on the blackboard. Figure 4-8 shows the pro-

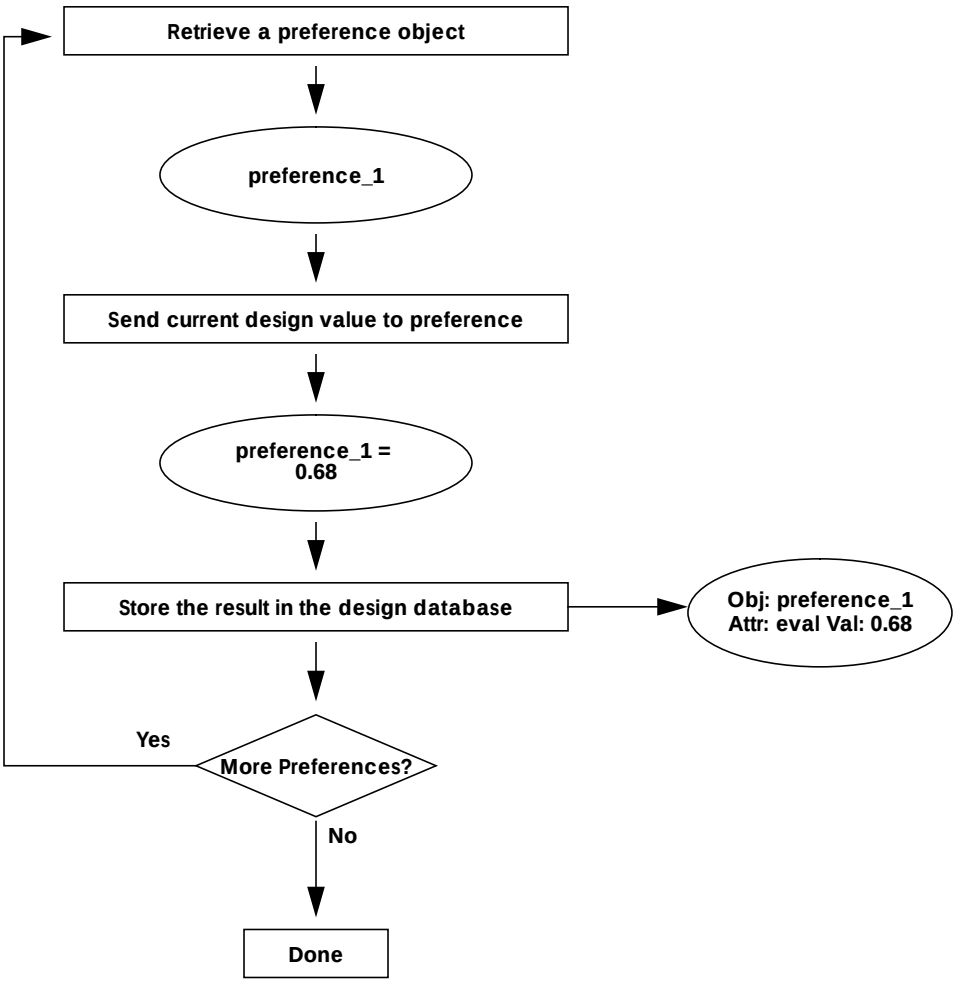

FIGURE 4-8: **Evaluator Functionality**

cess. In addition to that, the evaluators have precondition constraints just like the estimators.

Conflict Resolution. The basic CR that the evaluators provide is equivalent to the estimator CR, with negotiation about failed preconditions and requests to add constraints, in the case of selector-evaluator conflicts. In addition to that, it can negotiate with estimators about missing information and required information quality.

4.4.4 Critic

The critics have a set of constraints, similar to the evaluators set of preferences, which they analyze in turn (see Figure 4-9). If any of the constraints fail, the critic can start a negotiation with the originator of the design value. Depending on the agent's point-of-view (POV) several things can happen:

- If the originator has the same POV, the critic might be able to send the failing constraint to the agent. For example, when a material critic has a failing constraint on the thermal-conductivity, it can request the material thermal-conductivity selector to add the constraint into the selection constraints.
- If the POV is different between the critic and the originator, it is not guaranteed that such a constraint passing would be successful. In a situation where the selector makes a decision, e.g., based on thermal conductivity, and the critic has a complaint about the price of the material (which has been derived from the material selection by the material cost estimator), the failing constraint on the material cost cannot be used by the selector, since it has no knowledge about cost.
- A situation where constraint passing is useful, even with different POVs, occurs when a critic has a constraint failure about a value or attribute which the selector can evaluate immediately, i.e., without involvement of a third agent. For example, a critic for material bend-strength might be able to send the failing constraint on the bend strength to the material-thermal conductivity selector. In following selection steps, the selector will incorporate part of the knowledge from another point-of-view into its decisions.

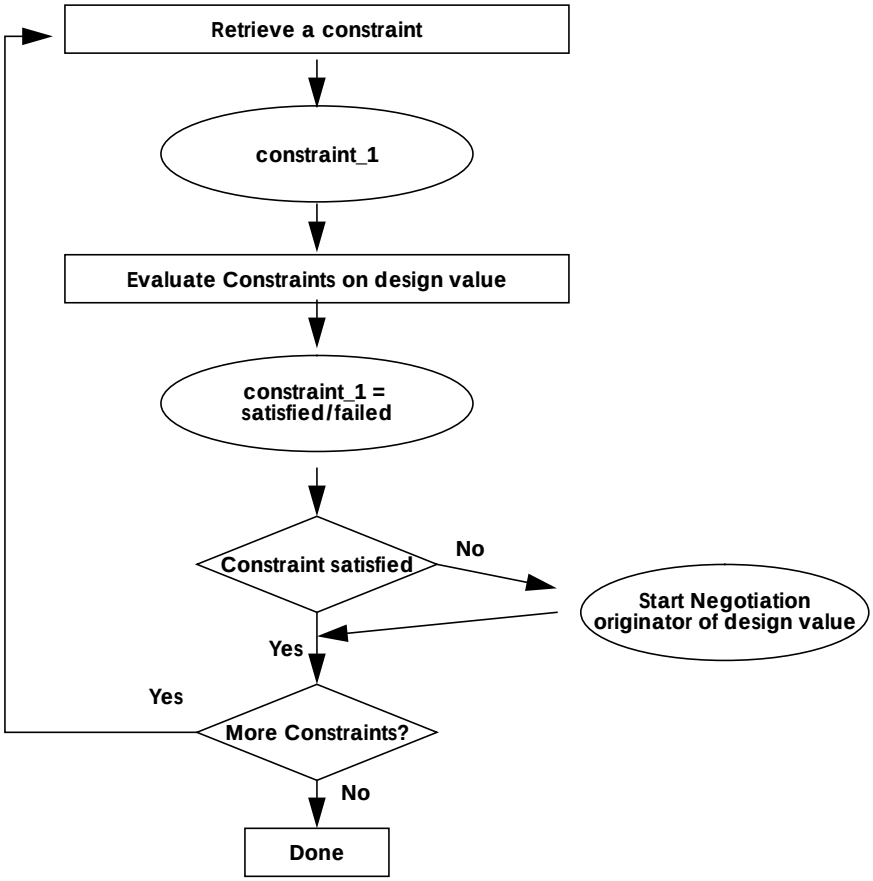

FIGURE 4-9: **Critic Functionality**

In other words, constraint-passing is only useful, if the other agent can directly evaluate and use the constraint. Tracing the interactions between these design decisions is not a trivial task, so that constraint-passing between different POVs can probably only be applied in a domain-dependent way.

In addition to the standard constraint-complaint cycle, the developers are free to implement additional, more complex criticisms in the critics knowledge base.

4.4.5 Suggestor

At present, no support for suggestors has been built into the SINE system. Unlike the suggestors in a user-interactive system, which suggest to the user what steps to take next, the suggestors in a non-interactive system, such as SINE, would suggest to other agents what steps to take. This could be used in cases where no agreement can be reached in a negotiation.

4.5 Summary

In this chapter we discussed the SINE platform. We started out by defining the goals of the platform: building a reusable system, implementing domain-dependent and domain-independent CR knowledge, experimenting with conflict, and designing and implementing a negotiation language for SiFAs (see section 4.2).

In section 4.3, we describe the architecture, by presenting the agent topology, data and negotiation links, knowledge representation, communication, conflict handling and scheduling.

Section 4.4 presents the agent design, describing in detail the functionality of every agent type.

In the next chapter we describe the implementation of SINE.

CHAPTER 5

Implementation of SINE

5.1 Introduction

This chapter focuses on the implementational issues of SINE. First we describe the programming environment in which SINE has been developed. In section 5.3, “Classes and Inheritance”, we show how inheritance is used to develop the library of agents and design information objects. Section 5.4, “Agent Implementation”, describes the implementation of each agent type.

5.2 Programming Environment

The SINE system has been developed using the CLIPS (C Language Integrated Production System) language [Giarratano & Riley 1993] [Giarratano & Riley 1994]. CLIPS was developed by the NASA at the Software Technology Branch of the Johnson Space Center. It is a forward chaining rule-based inference engine, which is entirely programmed in the C language. CLIPS is distributed for a nominal fee, including the documentation and source code. This allows users to write rules in a language that is close to LISP, for use in a rule base system that is similar to the OPS5 rule base language. It also allows developers to modify and enhance the CLIPS system, by adding functionality through the CLIPS lan-

guage or additional C functions. The CLIPS environment is available on UNIX™/X-Windows™, Macintosh™ and MS-DOS™/MS-Windows™[1]. For this project, the UNIX version of CLIPS was used. The system was developed on a DEC Alpha™[2] 3000-300, using GNU C.

CLIPS offers multiple ways of programming:

- *Rules*

  For a rule based system, the most common way is to design rules that get activated by facts, produce output and generate new facts. The inference engine analyzes the facts at all times and activates all the rules whose ‘if’ condition can be satisfied by the facts. Of the activated rules, one is selected and ‘fired’ (executed). Different strategies, such as most-specific-first, least-recently-used-first etc., can be used to decide which rule to fire.

- *Functions and Methods*

  CLIPS has strong support for functional programming. It implements an interpreter for a recursive language, similar to LISP. These functions can be used to manipulate facts, global variables and objects, and to activate rule-sets.

- *C Language Interface*

  CLIPS is written in C and it provides an interface through a library of functions for developers to compile with their own code. These functions can use any of the CLIPS or user-level C functions. Functions that the developers declare in CLIPS are made available in the CLIPS language and can be used just like any other CLIPS function.

CLIPS supports three kinds of data:

1. UNIX is a trademark of ATT Unix Systems Labs. X-Windows is a trademark of MIT. Macintosh is a trademark of Apple Computer Inc. MS-DOS ans MS-Windows are trademarks of Microsoft Inc.
2. Alpha is a trademark of Digital Equipment Corp.

- *Facts*

  A fact is an assertion, e.g. (is-a herring fish), meaning that herring are fish. A rule can be built to match this kind of fact, produce some output and/or derive new facts.

- *Objects and Classes*

  A data object is a combination of a data structure, holding information, and functions that act on the data. A developer can build a hierarchy of classes, where data members and functions can be inherited from the higher classes to the more concrete classes. CLIPS allows objects to be used on the left hand side (the ‘IF’ side of rules). It can pattern-match against any object description, i.e., by object class, object name, or any kind of slot value pattern.

  Objects are not restricted in complexity or size and therefore allow very powerful functionality to be realized with them. CLIPS supports multiple inheritance for classes, i.e. a child class can inherit the features of more than one parent class. This possibility is exploited by the agent classes.

- *Global Variables*

  This kind of data is very limited in use, as it cannot be employed to control the activations of rules, unlike objects and facts. It is useful though, for maintaining internal counters, such as for counting the current cycle of agent activity.

As the objects have functions attached to them, they become active entities in the program. Communication to objects is realized with messages. Message-Handlers in objects are functions that respond to messages and perform some kind of action on the data in the object. The actions can be anything from storing a value in the object over generating new objects to running an entire rule-base to produce new results.

The SINE system makes extensive use of these methods, both for rule base activation and for data manipulation. Sections 5.3, “Classes and Inheritance” and 5.4, “Agent Implementation” describe the methods used in more detail.

5.3 Classes and Inheritance

All classes in SINE are derived from one top-level class called ‘sifa-object’(see Figure 5-1). This allows the developer to attach functions for tracing information and general object

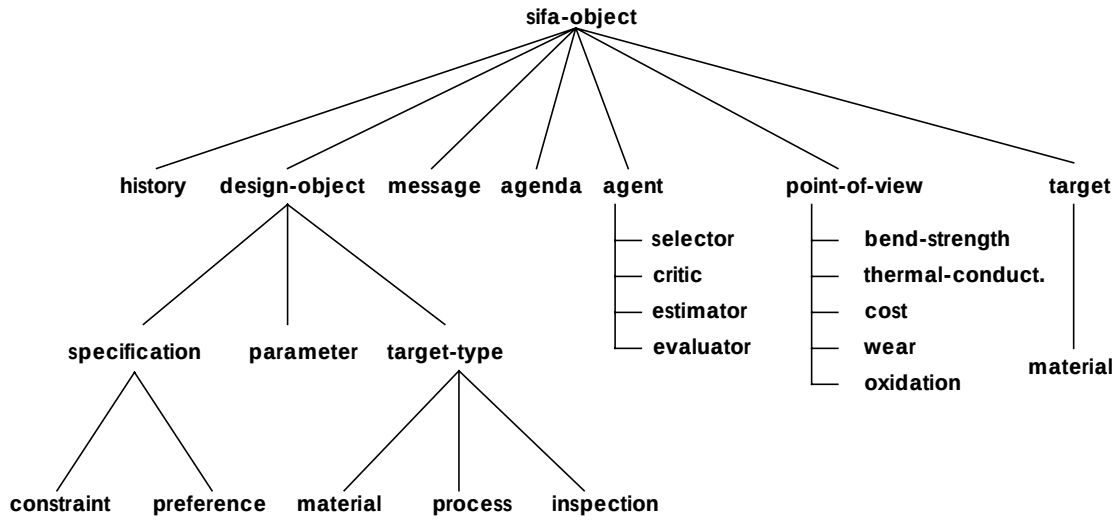

FIGURE 5-1: **SINE Class Hierarchy**

management. The class provides slots for storage of a location, i.e., where, or to what agent, the data belongs. It also has a slot for storage of a full, verbose name of the object, for output to the users.

5.3.1 History Class

The history class holds information that the agents can generate whenever they want to, especially after decisions or positive/negative outcomes of negotiations. It stores the name of the agent, a time stamp, the name of the action that the agent has taken, the objects that are affected by the action, the partners that were involved in the decision and an explanation for output to the user.

5.3.2 Message Class

The message class stores all the information for a speech act, i.e., the KQML slots and the message content, made from the subject and the two parameter slots. It also contains a time stamp slot containing information about when the message was created.

5.3.3 Design Object Class

All objects that directly relate to the design are derived from the design_object base class. This class provides the ‘object_type’ slot for tagging the instances as being either requirements, alternatives or parameters. This is used by agents to distinguish between parameters in the design and possible choices for parameters, among other purposes.

Design Parameter Class. The design parameter class is derived from the ‘design_object’ class. It stores the value of a parameter or a reference to an instance that fills in for the value. For the material parameter, for example, the parameter stores the name of a particular material in the value slot. This is a reference to an instance of a material class object.

The origin slot contains the name of the agent that stored the value, or nil if the value has not been stored yet. The time stamp slot is updated every time that the value is changed.

The callback slot stores a list of agents that will automatically be notified when the parameter is updated. Agents can request entries to be added and removed from this list through add and delete messages. The list is used for agent activation and conflict detection by the agent class.

The design_parameter class also provides three message-handlers. The first one, ‘request’ is used by agents to request the parameter object to accept (i.e., store) a certain design value for its value slot. There are several situations that can occur, depending on whether the value that the agent requests is the same as the value stored in the parameter object, and whether the origin, i.e., the agent requesting the value, is the same as the origin stored in the object, i.e., the agent that originally stored the previous value:

Value (in Object)	Origin (in Object)	Action
nil	nil	Value is stored and the origin slot is updated with the name of the requesting agent. Agent receives positive acknowledgment.
<value> = <request>	<origin> = <agent>	Value is updated to reflect most recent request. Agent receives positive acknowledgment.
<value> = <request>	<origin> ≠ <agent>	Value and origin remain the same. Agent receives positive acknowledgment.
<value> ≠ <request>	<origin> ≠ <agent>	Modification is refused by the parameter. Agent receives negative acknowledgment.

TABLE 5-1: **Parameter Update Requests and Reactions**

An agent can use the ‘force’ message-handler to unconditionally update a design parameter, but this is not necessary in most cases, since the new value should be decided in negotiation with the original agent. In that case, the original agent can update the value with the ‘request’ message.

The design parameter class uses a specially designed ‘put’ function for storing new values. This function automatically notifies all the agents in the callback list of the updated value, by sending them a ‘callback’ message.

Specification Class. This is the base class of all object that contain design specifications, such as constraints and preferences.

Constraint Class. As constraints are probably the most important objects in design expert systems, SINE provides a large number of different kinds of constraints for use in building design systems. In essence, a constraint has a list of object attribute pairs that it constrains. A predicate is stored and used to evaluate for success or failure. The ‘check’ message handler retrieves this predicate and evaluates it with the values of the object attributes in the design database.

In order to allow constraint relaxation, the constraint object contains a list of predicates, in increasing order of relaxation. By default, the first predicate will be used, but whenever the object receives a put-relaxation message, it retrieves the predicate that is associated with the level specified in the parameter of the message and stores it in the ‘current_predicate’ slot. Only the ‘current_predicate’ is used to evaluate the constraint.

While the ‘check’ message request the constraint to evaluate whether it is satisfied under the given design choices, the ‘check_alt’ message handler allows agents to evaluate alternative choices. For example, if a constraint that restricts the material cost fails given material A, it can be sent a message to check material B. If it is satisfied, the agent that sent the message might consider material B a better choice than A.

The ‘filter’ message-handler receives a list of alternatives and iterates through them, returning a restricted list, of which all members would satisfy the constraint.

There are different subtypes of constraints:

Type	Description	Logical Equivalent
exist	A value has to exist for the specified slot in the named object	value ≠ nil
larger	The first attribute has to be larger than the second.	value1 > value2
smaller	The first attribute has to be smaller than the second.	value2 < value1
ref_range	The value of the first object's attribute has to be in the range specified by the second object's attribute.	value2a < value1 < value2b
alt_range	The range specified by the first object has to include the value specified in the attribute of the second object.	value1a < value2 < value1b

TABLE 5-2: **Constraint Types**

Preferences. Whenever a relation between objects has to be stored, such as for ordering alternatives by their quality, objects of the 'preference' class can be used. A preference stores the name of an attribute and a function. There are two types of preferences:

- *Comparative Preferences*

  The 'compare' message-handler retrieves the values of the two objects that are passed as parameters and applies the function to them. The comparison function acts as a predicate, returning TRUE if the relation holds, FALSE otherwise.

  The 'sort' message-handler takes a list of alternatives and returns them sorted by decreasing preference.

  The 'min' and 'max' message-handlers search for the objects with the highest/lowest preference value in a list.

- *Quantified Preferences*

  A more powerful type is the 'quantified_preference' class, which uses quantitative measures to perform the evaluation. It stores a minimum and a maximum value and computes a linear index for any alternative.

  The 'evaluate' message-handler returns the index, which can be used by evaluator

agents, by selectors, but also by other agent types. The ‘compare’ message-handler computes the indices of the alternatives and compares those. The ‘min’ and ‘max’ handlers perform equivalent functions to the previous class.

There are two types of preference for each of the ‘quantified_preference’ and the ‘comparative_preference’ classes, namely the ‘smaller’ and the ‘larger’ type. They prefer smaller or larger attribute values, respectively. This makes preference specification one step easier, as they use predefined comparison functions and only the attribute has to be specified.

5.3.4 Target-Type Class

For every kind of target, an appropriate class has to be created. It should have slots for all the kinds of information that are relevant to the target. In the domain of the demonstration only the material target is used. The material class has slots for bend-strength, thermal-conductivity, wear performance, oxidation performance, unit-cost, density, mass, part-cost and total cost.

5.4 Agent Implementation

Every agent in SINE is an instance of some kind of class. There is a specific class for every agent, where the class is based on features inherited from a specific agent type class, a target class and a point of view class. Figure 5-2 shows this multiple inheritance. The three letter abbreviations in the agent blocks are the names of the agents in the implementation, see in 6.3, “Simulation of I3D+ Conflicts”.

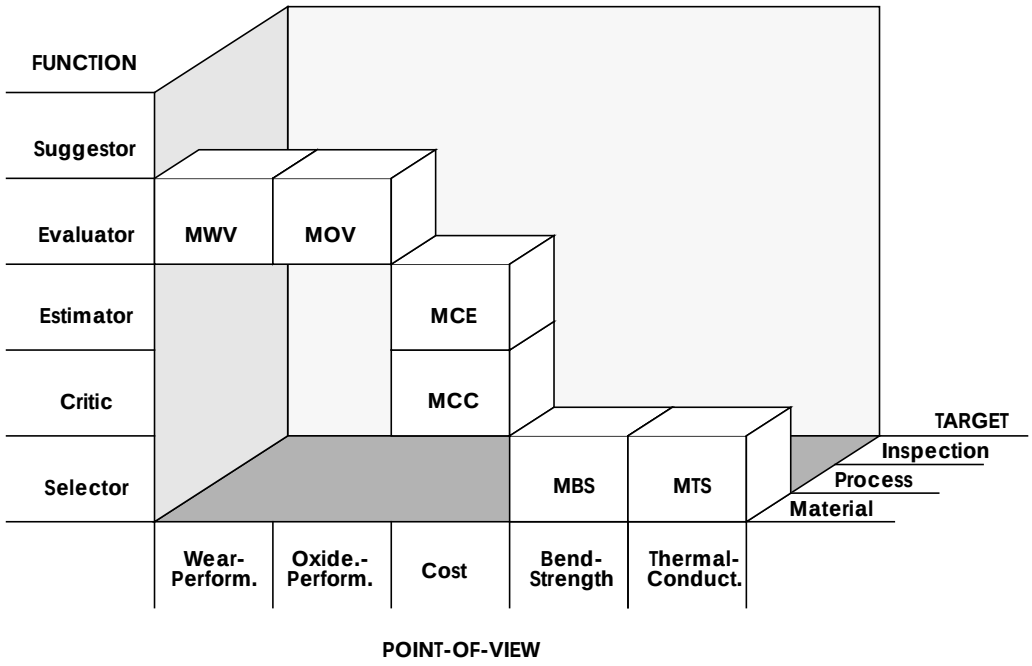

FIGURE 5-2: **SINE: Multiple Inheritance for Agents**

The agents all inherit from a common base class called ‘agent’. This class provides many features that are used by the majority of the agents, such as storing the names of the constraints and preferences that the agents use, as well as the names of the rule modules (see below) that the agents employ. The agent base class can also handle the activation of agents through callback messages from the agenda handler, and through checking of an agent’s precondition constraints.

Many other message-handlers are included in the agent class (see appendix B, “Developers’ Guide”). There are facilities to add and remove constraints, modify the current process that the agent uses, and a method that displays the decision history of the agent.

Since CLIPS does not provide any way to inherit rules in object classes, the SINE system provides modules with rules for every agent type. These can be used for each instance of

an agent, by filling a slot value with the name of the rule-module. The rules get activated from message-handlers. Figure 5-3 shows how the rule module gets activated.

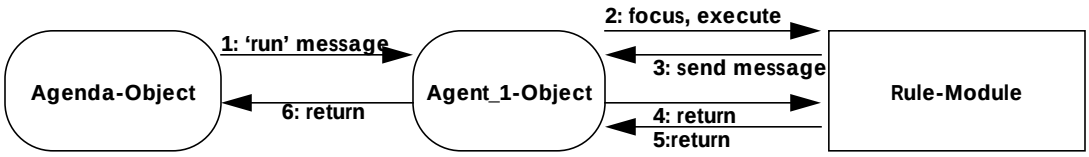

FIGURE 5-3: **Activation of Rule Modules in SINE**

The module, in turn, can send messages to objects, especially to the agent instance that called it. This is used in the selector and critic agents to activate the message-handlers which perform their major functions.

In the case of a conflict, the CR-Module is activated. It then sends speech act messages which are dispatched by a communication function. This function automatically locates the CR module of the destination agent. It activates that module, so that the rules can fire and produce a reaction. A reply is sent back.

There can be multiple messages sent back and forth between the CR modules, until the conflict gets resolved. Once that is the case, the first CR module returns control to the agent that activated it. The agent terminates its main function and control is returned to the agenda manager object. Figure 5-4 tries to capture the complexity of the recursive calls in graphical form.

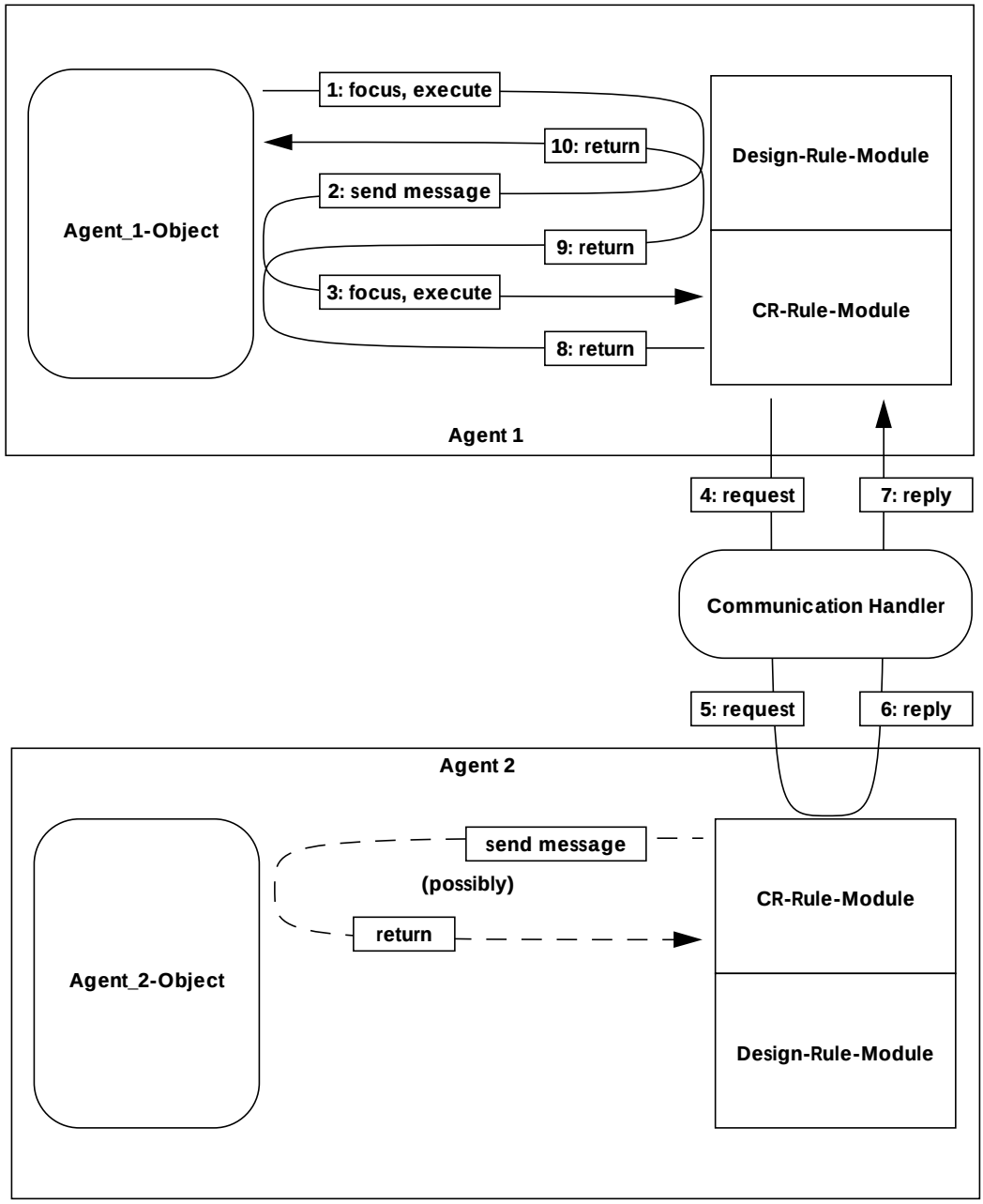

FIGURE 5-4: **Activation of Modules and Objects during CR**

5.5 User Interface

Design expert systems developed with the SINE platform can use a multi-windowed user interface, developed by Jonathan Kemble, a Masters student in CS at WPI. The CLIPS process and the windowing program interface act as client and server through a TCP/IP connection. Router functions in CLIPS allow the platform to work with or without window interface, without change in the code.

The window interface provides individual output for every agent, a main window and an agenda window. Figure 5-5 on page 102 shows a screen dump of the interface running with the I3D+ simulation. Since the agents usually produce more output than can be displayed in the limited space available on the screen, each window has a horizontal and a vertical scrollbar, to allow random scrolling through the message text.

5.6 Summary

This chapter began with a section about the programming environment used for implementing SINE, which was developed using CLIPS and GNU C on a UNIX system.

In section 5.3 we presented the object classes and the inheritance relations that are used in SINE. We described the history class, the message class, the design object class and the target type class, postponing the discussion of the agent class until the following section.

Section 5.4 was entirely devoted to the description of the agent implementation and the threefold multiple inheritance that SINE uses.

The user interface that SINE offers (section 5.5) concluded the material in this chapter.

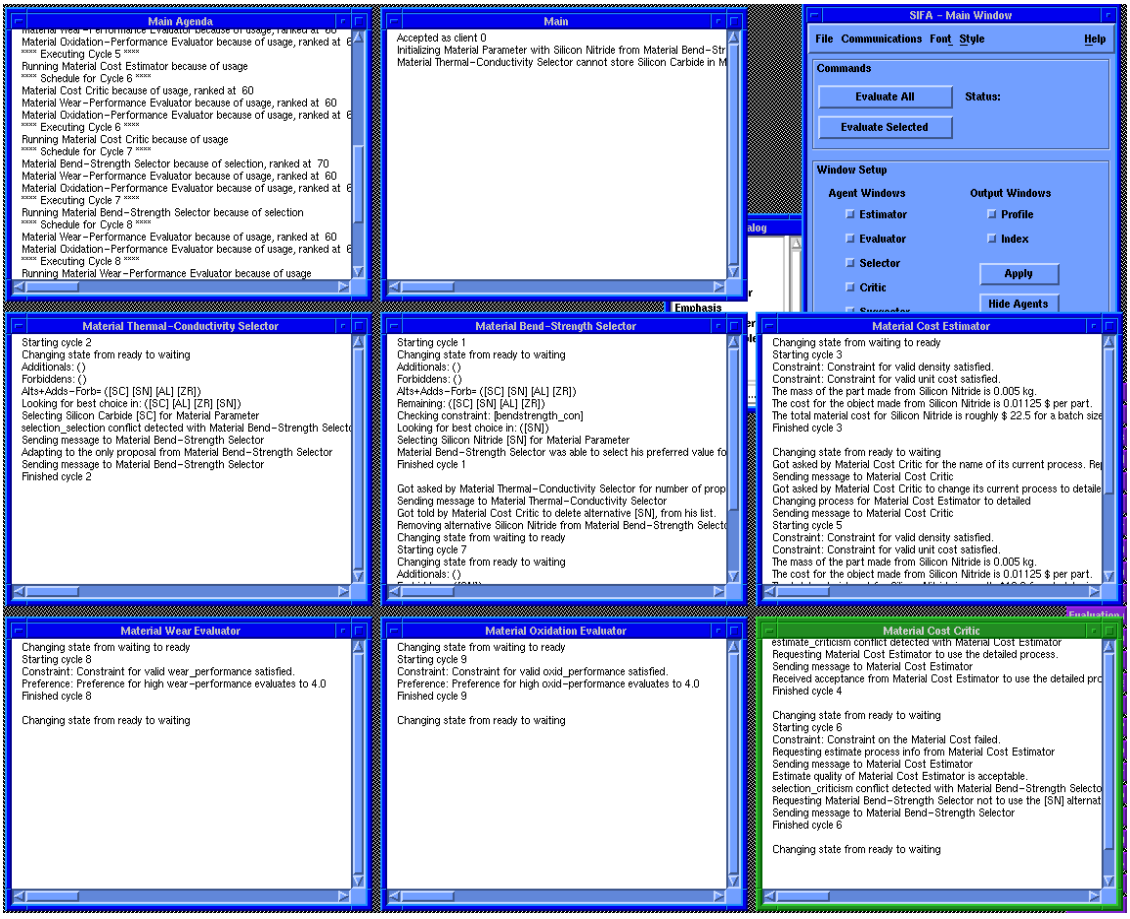

FIGURE 5-5: **SINE Multiwindow Interface**

In the next chapter we evaluate the work described in this thesis, by analyzing the theoretical and practical contributions of it.

CHAPTER 6

Evaluation

6.1 Introduction

In this section, we will attempt to evaluate the work in this thesis. We will first investigate the theoretical achievements, related to the exploration of the SiFA Paradigm. We will compare the goals that were defined in section 1.2.2 to the results from chapter 3, “SiFAs and Negotiation”.

In order to analyze how well the practical goals of the thesis, as defined in section 1.2.3, were met, we evaluate the SINE platform by analyzing three design system examples that were developed with it. Sections 6.3, “Simulation of I3D+ Conflicts”, 6.4, “Design System Example with derived Attributes” and 6.5, “Adaptation to new Domain — Sailboat Design” present these systems. We also compare the platform to other SiFA systems and systems with larger agents (sections 6.6 and 6.7 respectively) by contrasting their key features.

In section 6.8 we evaluate the performance through measurements and by pointing out key aspects of SINE. We finish the chapter by assessing the understandability of the platform and the negotiation it produces with the help of an experienced I3D user who was an I3D co-developer.

6.2 Theoretical Achievements

The first theoretical goal described in section 1.2.2 was to define a set of domain independent agents. This was achieved by analyzing the agents in SNEAKERS, I3D and I3D+. We described the resulting set of agents in section 3.2, “Agent Types”. Section 6.5 presents a sample design system which has been developed independently of the previous SiFA systems for an entirely new domain by using the SINE platform. This suggests that the set of agent types is at least sufficient for simple design systems.

Two of the goals were to the investigate the agent negotiation and to generate a catalog of conflicts. Sections 3.3, “Conflict Occurrences”, 3.5, “Conflict Resolution” and 3.6, “Negotiation Strategies” describe the individual conflicts and the negotiation that could result from them, while section 3.4, “Conflict Types”, presents an abstraction of the conflicts into more general categories.

An important goal was to develop a communication language for the SiFAs. Section 4.3.4, “Communication”, describes the language that was developed and the speech acts that the agents can exchange.

Finding an adequate knowledge representation that would be suited both for design and conflict resolution was another goal. We presented the knowledge objects that are used for design (constraints and preferences) and the use of reusable rule modules for conflict resolution in section 4.3.3, “Knowledge Representation”.

The last theoretical goal, analyzing the use of design histories, has received a limited amount of study. The platform supports the generation of history objects which can be used both by the agents and by humans, but SINE does not employ this functionality for conflict resolution or design at this point.

6.3 Simulation of I3D+ Conflicts

The first practical goal of the thesis was to implement the conflicts that were generated in the I3D+ platform. I3D+ was used as a test case, since it was the first and only negotiating SiFA system. Although I3D+ used domain specific conflict resolution knowledge, SINE, being a domain-independent platform, should ideally be able find solutions for the same conflicts. For this purpose the design domain of the I3D+ material selection process was implemented [Victor 1993], by building two selectors, one estimator, two evaluators and one critic which are equivalent in terms of function, point of view and target to the ones used in I3D+. The following sections describe the agents.

6.3.1 Selectors

TABLE 6-1: Material Bend-Strength Selector

Name	MBS MaterialBend-StrengthSelector
Task	Select a Material such that the range of bend strength that the material offers include the value of the bend strength parameter in the requirements. If several materials apply, use the one with the highest bend strength
Inheritance	Selector, Material-Target, Bend-Strength POV
Preferences	Preference for high bend-strength
Constraints	Alt_range_constraint on the range of the bend-strength

TABLE 6-2: Material Thermal-Conductivity Selector

Name	MTS MaterialThermal-ConductivitySelector
Task	Select a Material such that the thermal conductivity is as high as possible.
Inheritance	Selector, Material-Target, Thermal-Conductivity POV
Preferences	Larger_comparative_preference for high Thermal-Conductivity
Constraints	none

6.3.2 Estimators

TABLE 6-3: Material Cost Estimator

Name	MCE MaterialCostEstimator
Task	Estimate the material cost, using either a rough or a detailed process
Inheritance	Estimator, Material-Target, Cost POV
Preferences	none
Constraints	Existence_constraint for valid density, Existence_constraint for valid unit cost

6.3.3 Evaluators

TABLE 6-4: Material Oxidation-Performance Evaluator

Name	MOV MaterialOxidPerformanceEvaluator
Task	Evaluate the oxidation performance of the material, with respect to the requirements
Inheritance	Evaluator, Material-Target, Oxidation-Performance POV
Preferences	Quantified_preference for high oxidation-performance
Constraints	Existence_constraint for valid oxidation-performance

TABLE 6-5: Material Wear-Performance Evaluator

Name	MWV MaterialWearPerformanceEvaluator
Task	Evaluate the wear performance of the material, with respect to the requirements
Inheritance	Evaluator, Material-Target, Wear-Performance POV
Preferences	Quantified_preference for high wear-performance
Constraints	Existence_constraint for valid wear-performance

6.3.4 Critics

TABLE 6-6: Material Cost Critic

Name	MCC MaterialCostCritic
Task	Criticize high costs, ensure accuracy of cost estimation
Inheritance	Critic, Material-Target, Cost POV
Preferences	
Constraints	smaller_constraint on the Material Cost, domain specific rules

6.3.5 Design Process

The agents interact to produce a value for the material parameter that satisfies all agents. They use requirements that the user can specify in a file before execution. The requirements are specified as a partial description of the material to be obtained. For example:

```
(req_material of material_type
      (object_type requirement)
      (full_name “Requirement on Material”)
      (bend_strength 500)
      (wear_performance 2)
      (oxid_performance 4)
      (cost 50))
```

The sequence of the agents is only partly defined by dependencies, in that the estimators and evaluators have to wait for the selectors to make a decision on the material, before they can run. The critic has to wait for the estimator to produce an estimation before it can start criticizing. The evaluators and estimators can only start deriving values when their input constraints are satisfied. However, they will start negotiation with the selectors whenever the selectors assert a value which is not suitable for the precondition constraints in the estimators or evaluators. Figure 6-1 shows all the data dependencies and the possible negotiation links, of which some or all can appear at runtime.

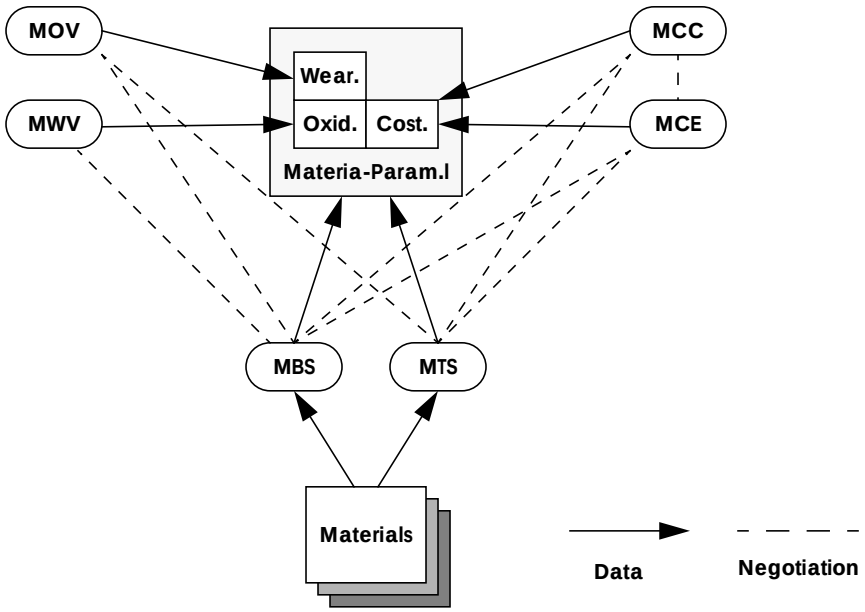

FIGURE 6-1: **Data Flow and Negotiation Links in the Material Selection Example**

Although the general structure of agents and design parameters was preserved in the I3D+ simulation, some the resulting conflicts and the conflict resolution are noticeably different. First, I3D+ only simulated conflict resolution between the selectors and between MCC and MCE. It did not have any evaluators.

Furthermore, I3D+ used global and local goals to decide which agents should win in conflict. SINE is situation driven, i.e., the agents have no goals and plans, but rather try to solve conflict situations by compromising or adapting as well as possible. The selectors do not try to override the value asserted by the other selector, but they try to find a value that both of them can agree on.

Due to these differences in structure, SINE will not produce the same negotiation as I3D+. However, it is able to solve conflicts between the same pairs of agents, and it implements CR for more pairs of agents and with more possible outcomes than I3D+. Due to this, the CR abilities in SINE can be seen as a superset of those implemented in I3D+, which proves the quality of the CR knowledge. The design decisions that each individual agent takes if no conflict occurs are equivalent to the decisions taken by the agents in I3D and I3D+.

6.4 Design System Example with derived Attributes

In the I3D/I3D+ systems, and the I3D+ simulation using SINE, only alternatives with predetermined attribute values are used. In order to evaluate the support of the platform for design problems with more complex dependencies we designed a sample case in which attributes of material alternatives are estimated and evaluated.

In the example, an estimator derives a thermal-conductivity value for each material, based on the material's density.

From the thermal-conductivity estimates, the evaluator computes a thermal-conductivity index, and it stores the values in the materials. The index is called 'thermal-conductivity evaluation', and it is the target for both the evaluator and the selector.

The index is used by the selector agent to choose the one material with the highest value (see Figure 6-1). Sections 6.4.1 through 6.4.3 describe the agents that were implemented.

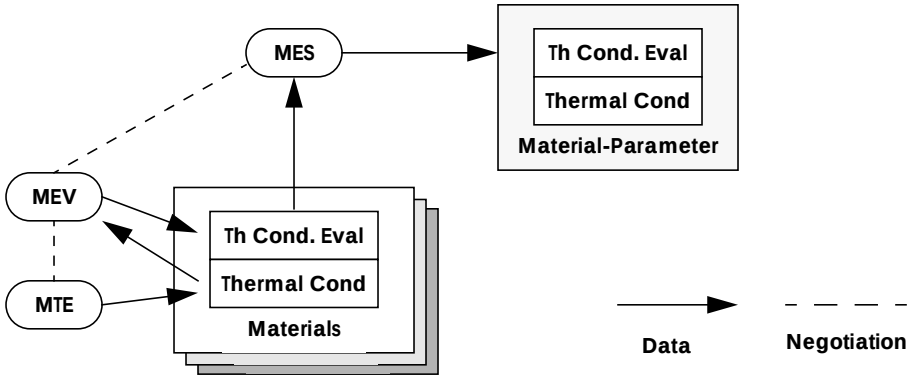

FIGURE 6-2: **Data Flow and Negotiation Links in the Derived Attribute Example**

6.4.1 Selector

TABLE 6-7: Material Thermal-Conductivity Evaluation Selector

Name	MES MaterialThermalCondEvaluationSelector
Task	Select the material with the highest thermal-conductivity evaluation (index).
Inheritance	Selector, Material-Target, Thermal-Cond-Eval POV
Preferences	Preference for high thermal conductivity evaluation
Constraints	none

6.4.2 Estimator

TABLE 6-8: Material Thermal Conductivity Estimator

Name	MTE MaterialThermalCondEstimator
Task	Estimate the thermal conductivity, using either a rough or a detailed process
Inheritance	Estimator, Material-Target, Thermal-Cond POV
Preferences	none
Constraints	none

The estimator can use a rough or a detailed estimation method. The rough method simulates an estimation method that can produce a statistical reliability of up to 60% and a statistical error margin of 20%. The detailed method can produce 80% and 5% respectively. The agent will at first only assert the estimates and provide no reliability information. At the request of other agents, it will also assert reliability and/or error margin information. If other agents request higher reliability or lower error margins, the estimator will switch to the detailed model.

6.4.3 Evaluator

TABLE 6-9: Material Thermal Conductivity Evaluation Evaluator

Name	MEV MaterialThermalCondEvalEvaluator
Task	Evaluate the thermal conductivity and produce an index for it
Inheritance	Evaluator, Material-Target, Thermal-Cond-Eval POV
Preferences	Quantified preference for high thermal conductivity
Constraints	none

6.4.4 Negotiation

There is a possibility for negotiation between the selector and the evaluator when the selector cannot find enough difference in quality between the alternatives. This will prompt the evaluator to produce a more critical evaluation.

Also, the evaluator can request the estimator agent to produce statistical information (reliability, error-margin) and it can request information in a certain quality (x% reliability, x% error).

Please refer to the traces in Appendix C for details of the negotiation.

6.4.5 Conclusion

This example shows that the SINE platform is able to handle design tasks in which the attributes of alternatives are produced by the agents themselves. Since this shows that the platform can handle more complex interactions than the ones presented in the I3D+ simulation, it proves that it is usable for a wider range of design tasks.

6.5 Adaptation to new Domain — Sailboat Design

In order to test the portability and usefulness of the SINE platform, a visiting student, Phil Tomlinson (now at the Key Center of Design, University of Sidney), implemented a design system for sailboats using SINE. The task of the system is to perform a routine type selection and parametric design, involving several design attributes and parameters (Hull Design Type, Sail Design Type, Sail Material, Hull Material, Number of Sails). For every one of the parameters, several values are possible but only one can be selected.

6.5.1 Design Parameters and Attributes

The values are multi-attribute structures, similar to the material_type values in the I3D simulation. Each numeric slot can take a value in a range from 0 to 100. For instance, a Hull Design can be of the following kind:

```
Hull Design Type
      name “Sloop”
      cost 70
      draft 30
      floatation 20
      durability 60
```

The system uses the following targets:

- Sail Boat
- Hull Design Type
- Sail Design Type
- Hull Material
- Sail Material
- Num Sails

These agent functions are used:

- Selector
- Estimator
- Suggestor
- Critic
- Evaluator

And these Points of View are taken into account:

- Cost
- Speed
- Durability
- Cargo

6.5.2 Conflicts

The developer of the Sailboat design system expects interesting conflicts about different issues between agents. One area of the conflicts appears to be where cost versus performance is an issue. This is a typical area for trade-offs between goals. The following conflicts have been identified:

- Sail Material Cost Selector - Sail Material Durability Selector
- NumSails Cost Selector - NumSails Speed Selector
- Hull Design Cost Selector - Hull Design Cargo Selector

Another set of conflicts are located where suboptimal values are chosen for two or more design parameters, so that the parameters together offer an improvement over the optimal decision chosen for either one of the individual design parameters.

- Num Sails Speed Selector - Sail Design Type Durability Selector - Boat Speed/Durability Evaluator:
  The number of sails affects the speed and the sail design influences the durability. An evaluator could be designed to produce an evaluation that represents a preference for a combination of these points of view.

- Num Sails Speed Selector - Hull Design Speed Selector - Boat Cost Critic:
  In this case the globally optimal solution might be where the Num Sails Speed Selector and the Hull Design Speed Selector select suboptimal values from their individual point of view.

A third range of conflicts appears where one value can be traded for another:

- Hull Design Cargo Selector - Num Sails Speed Selector:
  A conflict where the preference is for a fast boat which holds a large amount of cargo. In this case it may be necessary to go down from the FRIGATE hull design to the SLOOP hull design, giving up cargo capacity in exchange for speed.

6.5.3 State of the Implementation

In the current state of the implementation, about 30 agents have been designed, with their basic functionality implemented. Most of the computational tasks have been designed and coded with an algorithmic approach, rather than rules. This, again, is similar to what was done for the I3D simulation.

The domain independent part of SINE allows the system to perform some very basic negotiation, but more development and adaptation is needed here, to gather the full support from SINE for the design task.

6.5.4 Impressions from the Sailboat Designer Developer

Even though the developer has not yet finished the basic implementation of the Sailboat Designer, he was already able to provide the author with some feedback about his experiences:

- *Agent Classes*

  The object oriented approach to expert system design was perceived as very useful, especially for a developer with no previous experience in expert system development, since the agent classes provide the developer with an outline of the functionality and knowledge that needs to be implemented in the agents.

- *Domain Independent Conflict Resolution*

  The developer expressed faith in the ability of SINE to solve basic conflicts without the addition of domain dependent CR knowledge. For example, agent precondition negotiation and selector-selector conflict types can be handled by the existing CR knowledge in SINE.

- *External Data and Preferences*

  According to the developer, the use of preference objects and data (alternatives) stored separately from the agents has been found to be an unusually abstract but very practicable approach. He noted that compared to standard expert systems, the SINE platform had a strong ability to make relative choices using only minimal situational information, instead of being built around situation-action rules, because of the preference functions which can order alternatives under any circumstances.

- *Object Oriented Architecture*

  Even though the Sailboat Designer will possibly include a large number of agents, built from all the possible combinations of function, target and point-of-view, the developer noted that the inheritance of rules and functionality drastically limited the number of necessary rules and specific functions for each individual agent.

One of the goals of developing the SINE platform was to reduce the time needed to develop a negotiation based SiFA system. From the experience with the Sailboat Designer, the following statistics have been derived:

- About 50 hours of work were needed by the developer to implement the system to its current point.
- Of that time, around 40% were spent on learning about expert systems in general, and the CLIPS environment and syntax.
- About 30% of the time was spent understanding the functioning of the SINE platform and the I3D+ simulation. In particular, the workings of the objects with their multiple inheritance was found to be challenging, since no documentation was available yet, and the source code is written at a very abstract level. The developer noted that some higher level development environment would be useful for further development with the SINE platform.
- The last 30% were used for reviewing papers about SiFAs and SINE, and helping the author in developing the platform through questions and recommendations.

It should be noted that at the time of development of the Sailboat Designer, neither the platform nor the documentation were finished. For future developers, the Programmers' and the Users' Guide in this thesis should help reduce the effort needed in understanding the platform. These should be revised in the future as more experience is gained with the SINE platform.

6.6 Comparison to other SiFA Systems

We now present a comparative analysis of SINE and previous SiFA based design systems, I3D+ in particular, as SNEAKERS had no support for negotiation. We first compare the general aspects of the systems, including the solution quality and the negotiation support, and then, in the second part, we compare development issues.

6.6.1 General Aspects

- *Solution Quality*

  SINE finds solutions for more combinations of requirements than I3D/I3D+, since these systems are not able to make a material choice, if the problem is undercon-strained. In SINE, the selectors will make a proposal from the list of all possible choices and allow other agent types to agree or disagree with the value.

- *Conflict Types*

  SINE currently implements support for five different types of conflicts (see section 3.4, “Conflict Types”), each of which can appear between different sets of agent types. I3D+ only supported a limited number of situations for conflict between two selectors.

- *Knowledge Representation*

  In SINE, agents use explicit representations of constraints, preferences and alternatives. This allows formulation of the design task on a more abstract level. Traditionally, expert systems encode this knowledge in the rule base. Although rules are very useful for encoding knowledge about specific actions to take under certain circumstances, they prevent the design knowledge from being handled and manipulated like data. In SINE, agents can converse and reason about the design knowledge, they can even learn this knowledge during the design process.

- *Support for Constraint Relaxation*

  The constraint object class in SINE has built in functionality for constraint relaxation by simple message passing. This allows reusing the same rules and functions, indepen-dently of the current level of relaxation, which reduces the amount of code.

- *Templates*

  The uniform structure of agents allows both easy development of design systems using agent templates (classes) and domain-independent functionality, as well as easier understanding and modifying of an existing system.

In other SiFA systems, understanding the functionality of the agents is more difficult, due to the heterogenous form of the agents and the low-level and implicit representation of knowledge and data.

- *External Data Files*

  The use of knowledge databases avoids data-representing rules and simplifies the design, which also makes maintenance and reuse more efficient than in the other SiFA systems, where the data are stored in the rules.

6.6.2 Development and Implementation

In the following we will contrast some of the rule based agents from the I3D+ implementation with the equivalent SINE agents.

- *Selectors*

  A typical material selector in I3D+ needed about 20 to 30 of rules. In the I3D+ simulation with SINE, each of the two selectors only had 3 rules, two of which displayed begin and end messages and one which activated the domain independent selector function. The two agents themselves were defined with two and three instance-declarations respectively, of which one for each agent declares the agent object, one is used for defining the preference operators, and one selector uses a constraint object attached to it.

- *Critics*

  The material cost critic in I3D+ uses 9 rules, whereas the same function is achieved in the SINE implementation without any rules, using only constraints expressed as objects.

- Estimators

  For the estimators, the difference is less obvious. The material cost estimator in I3D and I3D+ is made from 13 rules, whereas the one built on the SINE platform uses 9 rules.

In addition to that the conflict resolution supported by the SINE implementation is more elaborate than the original I3D+ system's, since it supports more types of conflicts and more adequate solution-strategies.

6.7 Comparison to Systems with larger Agents

In this section we attempt to contrast the SINE platform and SiFA based expert systems designed with it to systems built with a larger agent size. We have to use a feature analysis, since there are no implementations available which would allow a direct comparison of the two approaches:

- SINE makes building negotiating systems easier. SiFA Agents usually have a simple function and share something in common, e.g. function, target or point-of-view.
- Larger agents have more complex negotiation, as functionality is less restricted.
- SINE provides a large amount of domain independent CR knowledge as a basis for negotiation and thus reduces the size of the domain-dependent CR knowledge needed. In systems with larger agents, the complexity of the agents prevents some of the abstraction that lead to such domain independent knowledge.
- Larger agents can less easily anticipate what the other agent knows, perhaps leading to a more complex communication language, or the increased need for a system to act as a converter between vocabularies and/or ontologies.

- Design system development is made easier, due to inheritance and existing supporting functions.

6.8 System Performance

Performance is less of an issue in research, but very important for full-fledged implementations in repeated use. Although the author developed SINE with considerations particularly for run-time performance, testing this performance is difficult, since the sizes of the demonstration problems were too small to produce run times over 1.5 seconds.

In the future, the design and implementation of larger size SiFA systems should give a better insight into the performance aspects of SINE.

In the following sections we evaluate usability of the SINE platform by analyzing the effect it has on the development process, the runtime performance and the maintenance of the resulting expert system.

6.8.1 Development Process

We can evaluate the quality of the platform by analyzing the effect it has on the development process. Ideally, the net effect of designing an expert system with SINE should be that it makes development easier and/or quicker. Also, in comparison to implementing negotiation in other ways than SiFAs, using SINE should make the design of the negotiation scheme simpler and possibly more effective. We find the following effects of SINE on design:

- The application can be developed without data files, i.e., the data can be acquired at the same time that the system is designed and implemented. This allows easier testing with “fake” data and prototyping with small data sets.
- The built-in basic negotiation knowledge and communication functionality remove a major burden from the application developers and allow an incremental implementation of domain specific CR knowledge. We can see from the implementation of the I3D+ simulation, how only a small number of rules had to be designed in a domain dependent fashion (see Figure 6-3).

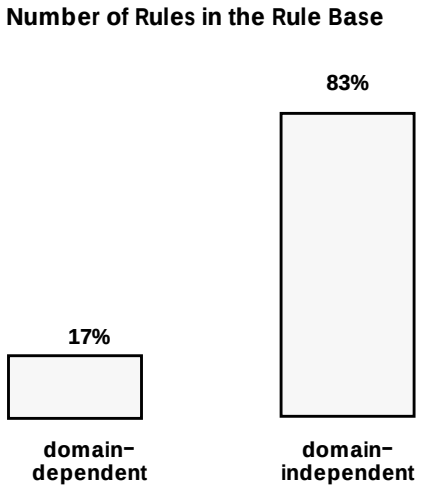

FIGURE 6-3: **Analysis of the Rule Base for I3D+ Simulation**

- The negotiation itself is simpler than with non-SiFA systems, due to the function, target and/or point-of-view that many of the negotiating partners share. The agents can make more assumptions about the other agents’ knowledge and abilities. Also, since the agents’ tasks are very limited, blame assignment is nearly trivial.
- As we already noted in 6.5, “Adaptation to new Domain — Sailboat Design”, the agent templates simplify the design of the expert systems.

- The available functions and methods reduce the amount of functionality that has to be implemented by the developer. In the I3D+ simulation, the number of domain dependent computational functions and class definitions was less than 10% (see Figure 6-4),

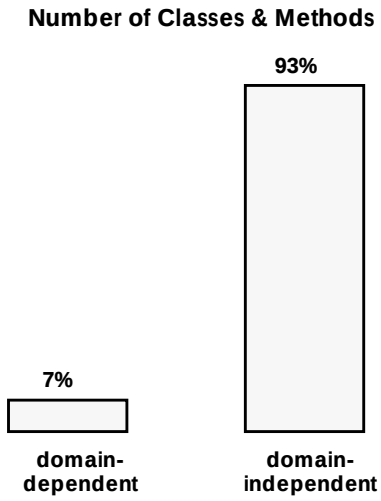

FIGURE 6-4: **Analysis of the Algorithmic Parts of the I3D+ Simulation**

6.8.2 Runtime Performance

Using SINE should produce performance at least as good as with other approaches. Ideally, it will speed up the problem solving task. The following are the runtime effects of SINE:

- The algorithmic support functions in SINE produce runtime speedup, especially during operation on larger data sets. This is due to the fact that operations such as search for optima and the sorting of choices is expensive in rule based systems, but a simple task for an algorithm to perform.

- One way of implementing agents involves running one agent at a time, as a CLIPS process by itself. I3D+ uses this scheme, which produces expensive context switches and generation of data files on disk. The use of CLIPS 6.0 rule modules produces a higher execution speed, because it avoids stopping and restarting the CLIPS inference engine, thus reducing the time needed to switch between agents from around 50 milliseconds to about 10 microseconds(!).
- Conflict avoidance through learning of constraints (see sections 4.3, “Architecture”, and 4.4.1, “Selector/Advisor”) reduces the amount of negotiation needed, especially for complex problems. It breaks the exponential number of possible combinations of values down to a very limited set of potential candidates after only a few negotiation cycles.
- External data can be included dynamically in the system. This allows more interactive applications, where the system assists the user in refining design decisions. It can also be used to make design decisions based on live data from other external sources.
- Figure 6-5 shows selected runtimes for the I3D+ simulation. It shows the runtime in dependence of the number of agents and the resulting number of design and negotiation cycles for different constellations of requirements. The first two bars represent the simulation of I3D+ both with and without the evaluators, for a run with the original 4 material types. The last two bars show the times and cycles needed to come to a decision, when 7 additional materials with missing information (unit-cost, density, etc.) are introduced. The difference in complexity between the last two runs is due to the differences in the requirements, which lead to different conflicts.

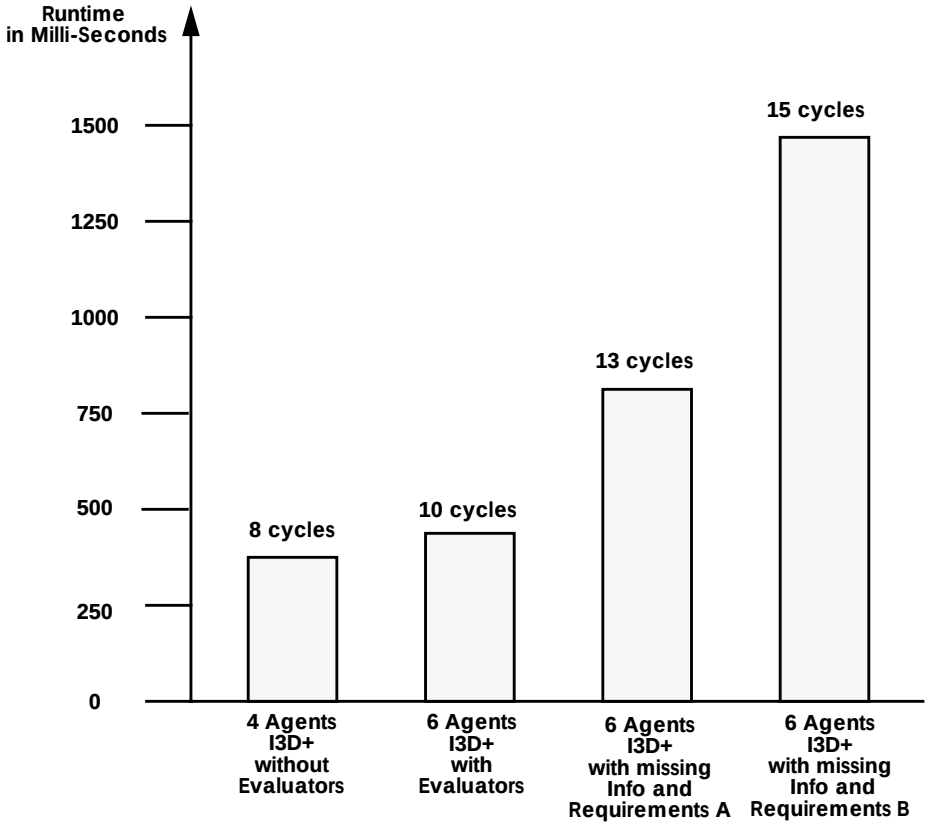

FIGURE 6-5: **Runtimes of the I3D+ Simulation**

6.8.3 System Maintenance

- Probably the most important maintenance issue is that the knowledge base can be kept the same, even when modifications to the data are made, due to the fact that the data are stored separately from the design knowledge.

6.9 Understandability

An important aspect of negotiation in AI systems is whether the resulting conflict resolution is understandable by a human. This leads to two requirements:

- A human should be able to read the output produced by the negotiating agents.
- The conflict resolution performed by the agent should appear reasonable and understandable for a human.

SINE has features that support both of these requirements:

- The system automatically generates human- and machine-readable output. During the negotiation, the rules that generate speech acts automatically produce a user readable form of the same information. In addition to that, the speech acts themselves are so close to human language, that even the pure data structures can be read with little experience.
- The explicit representation of constraints and preferences makes knowledge and communication understandable, since they are able to more accurately describe the intentions and constraint issues of the agents, compared to the usual non-descriptive rules.

The design and output of the I3D+ simulation has been presented to several faculty members and graduate students. Among them was a student from the Manufacturing department (Peter McCann), who had participated in the original development of I3D. He is now designing a graphical user interface and integration of I3D into a new CAD system. His comments support the claim that SINE had achieved the two most important goals:

- The negotiation introduced into SINE is understandable to potential users. In addition to that, the additional complexity introduced through the negotiation between the agents is acceptable to the user, because it reflects the conflicting nature of the CE design task.
- The implementation with abstract agent functions acting on high-level objects, such as preferences and tasks, was commented on as not only being understandable, but also being easier to comprehend than the original rule-based implementation.
- The separation of the system design into design knowledge and data was perceived very positively, since it achieves a maintainability which is important to the I3D developers and users, but which was never achieved with the original design.

6.10 Summary

This chapter presented an evaluation of the SINE platform. The first two parts (section 6.3 and 6.4) described a simulation of the I3D+ system using the platform and an example that uses derived attribute values.

The second section (6.5, “Adaptation to new Domain — Sailboat Design”) presented work that has used the platform to build an expert system for an entirely different domain. We described the system, with its attributes and parameters, the conflicts and the current state of the implementation. The section finishes by detailing the experiences that the developer had with the platform.

In section 6.6, we compared the SINE platform to other SiFA systems, I3D+ in particular. We investigated general aspects, development and implementational issues.

Performance issues were discussed in 6.8, where we evaluated development, runtime and maintenance issues.

The understandability of the system was evaluated in section 6.9, by analyzing the working and the output of the platform. We also presented the I3D+ simulation to an I3D expert for evaluation.

The chapter concluded with a comparison of SiFA systems based on SINE to expert systems with larger agent size (see section 6.7).

CHAPTER 7

Conclusions

7.1 Results of the Research into SiFA Negotiation

From the analysis of conflict occurrences, conflict types, the CR strategies and the agent character traits (see section 3.6, “Negotiation Strategies”), SiFAs show many more conflict possibilities and solution methods than have been discovered in previous research. This should provide many more opportunities for future research into conflict resolution using history and learning.

The analysis of functional and knowledge requirements (sections 3.7 and 3.8) can be used to guide the development of SiFA implementations, or to expand the capabilities of the SINE platform.

The list of different conflict types that has been developed can be used to refine design system implementations and remove unnecessary conflicts in systems that stem from poor system design or lack of information.

7.2 Results of Design and Use of the SINE Platform

The implementation of the I3D+ simulation and the Sailboat Designer show that the platform significantly reduces the amount of work needed to build a SiFA system and to support negotiation in it.

The quality of the results achieved by the negotiating agents and the demonstrated flexibility of the agents in situations with missing information show that SINE can both optimize design system output and reduce some of the brittleness of design expert systems. It also supports the hypothesis that negotiation is a useful tool in problem solving.

The breadth of negotiation situations producible with SiFA systems and supported by SINE allows research to investigate design systems and their internal conflicts and tradeoffs much more accurately, by breaking down the design task into very small subproblems which will be handled by distinct agents.

7.3 Future Work

Although a working platform is available and a considerable amount of research has been done in the area of SiFAs, both in this thesis and in [Victor 1993], there is still a large number of topics that can be further investigated and functionality that waits to be incorporated or improved:

- *Multiple CR Inheritance*

  Inheritance of multiple sets of rules and CR with fallback from specific to more abstract rules has not yet been implemented.

- *Design Parameter Structure*

  The structure of design parameter objects in SINE (e.g., Material, Sail-Type, etc.) is fairly fixed, and design of less pre-structured objects should be made possible.

- *Agent Grouping/Hierarchy*

  For systems with a larger number of parameters, some grouping of agents into sub-systems that are responsible for certain design areas might be useful. Meta-agents could help administrate the activation of the groups and the negotiation between groups. The structure could be similar to the hierarchy used in DSPL, but further investigation is needed to evaluate how that would change the problem solving power of SINE.

- *Negotiation*

  More negotiation should be implemented, i.e., between more agents and with more depth and variation in the existing negotiation support. The effect of different strategies and character traits of agents has not been investigated yet. Most of the negotiations involve only a small number of possible outcomes. Future work should investigate how more contextual information can be taken into account to adapt the conflict resolution to the situational needs.

- *Constraint Relaxation*

  Although SINE provides facilities for constraint relaxation, the current implementations do not employ them. A different design problem should be automated with SINE to use and analyze this feature.

- *Truth Maintenance*

  The current version of SINE informs all the concerned agents of new values of parameters. Although this is used as a basic form of truth maintenance, by ensuring that agents are aware of parameter changes, its effect is very limited. For example, the agents can

make assertions to one another and later revise their beliefs without notifying their partners. Some truth maintenance system within the agents should be available to prevent false information from remaining in the system.

- *Distributed Implementation*

  An implementation of SINE to work on several computers simultaneously, with each agent running as an individual process could be developed to investigate parallel SiFA systems. Such a system could introduce vast increases in design processing speed. Also it would lead to interesting research into the scaleability of SiFA systems.

- *Derived Parameters*

  SINE is best suited for design problems in which the choices of alternatives are predefined. Although a sample implementation has been built in which a selector uses the information from an evaluator, which in turn produced its evaluation from values derived by an estimator, the platform has only limited support for this kind of agent interaction. In the 6.4, “Design System Example with derived Attributes”, the lack is most obvious in that the agents have to use domain dependent knowledge and functionality to keep track of changes in the alternatives’ values. An enhanced version of SINE could have explicit support for parameter dependencies and agent-agent interactions. This would also simplify the blame assignment.

- *History Keeping and Design Rationale Uses*

  Even though SINE agents create history objects, that are kept in their memory and could be parsed to retrieve information, the domain independent parts of the platform do not take advantage of this information. Design rationale can be used for design revision and learning. Design history as well as rationale capture and use are both currently an active international research area. Investigation of both of these fields with respect to SiFAs could probably fill more pages than this entire thesis.

7.4 Summary

The evaluation shows that the SINE system has achieved most of its goals. It also shows that many more possibilities for research are available and that more work is needed to achieve the full potential of the SiFA paradigm. The research into negotiation has opened up a number of interesting issues, which also require more investigation, particularly the applications of learning.

APPENDIX A

Users' Guide

A.1 Introduction

In this guide we describe how a user can start and use a design expert system that has been developed with the SINE platform. We will first explain the basic components of the system, requirement specification, and modifications of data files.

Then we describe how to start the design system and how to use the user interface.

A.2 Components of a SINE-based Design Expert System

An expert system that was developed with the SINE platform consists of several parts:

- a modified version of the CLIPS expert system shell,
- a set of CLIPS definition files that form the basis of the SINE platform,
- a set of four CLIPS files for each agent in the system,
- requirements in a CLIPS file and
- data files (e.g., description of materials) for the expert system.

If set up correctly, the system will automatically take care of loading all the necessary files.

A.3 Requirement Specification

Since the requirement specification is dependent on the particular expert system that is used, we will use the example of the I3D+ simulation for all further discussion.

The requirements for the I3D+ simulation are a partial description of the material. The user has no influence on *how* these values are *interpreted*, as that knowledge is encoded in the agents. The values themselves, however, can be changed.

In the I3D+ simulation, for example, the requirements are specified as follows:

```
(req_material of material_type
            (object_type requirement)
            (full_name “Requirement on Material”)
            (bend_strength 800)
            (wear_performance 4)
            (oxid_performance 3)
            (cost 17)
            )
```

The parameters that can be defined by the user as requirements are:

- Bend Strength, i.e., the value that is entered for this attribute is used by the system to find a matching material that supports the required strength.
- Wear Performance, which is used by the Material-Wear-Performance-Evaluator to compute an factor that describes how well the selected material matches the requirement from the point of view of wear. The values in the requirement have the following meaning: very poor = 1, poor = 2, average = 3, good = 4, very good = 5. A lower index in the requirement will give a higher index for the chosen material. E.g., if the requirement has the value 2 and the material provides the performance 4, the evaluator will assert an accomplishment of the wear_performance requirement of 200% in the design database.

- Oxidation Performance, which is used by the Material-Oxidation-Performance-Evaluator to compute an index that describes how well the selected material matches the requirement from the point of view of oxidation. The meaning of the values is the same as for Wear Performance.
- Cost, which defines a maximum for the total production cost in Dollars. This value is used by the Material-Cost-Critic to prevent the selection of a material that is too expensive.

In order to change the requirements, simply edit the values by loading the `requirements.ins` file into your favorite editor (e.g., Emacs) and save the file back to disk.

A.4 Changing the Data Files

Although the data files are set up by the developers of the SINE-based design expert system, the users might need to update them for maintenance reasons. The data are stored in a CLIPS file, similar to the requirements. In order to change the data or to add new data, load the appropriate file into the editor.

For example, the file `materials.ins` contains the descriptions of all the materials that the selector agents can choose in the I3D+ simulation. A material description has the following parameters:

- Bend Strength (see section A.3);
- Wear Performance (see section A.3);
- Oxidation Performance (see section A.3);
- Thermal Conductivity (in Watts per Meter Kelvin), where a high value is preferred by the Material-Thermal-Conductivity-Selector;

- Unit Cost (in $/kg), which is used by the Material Cost Estimator to determine the cost of a single component and
- Density (in kg/ccm), which is used by the Material Cost Estimator to determine the mass of the component and to derive the cost based on the unit cost and the mass.

In the file, the format is in LISP syntax, for example, Silicon Nitride is described as follows:

```
(SN of material_type
    (object_type alternative)
    (full_name “Silicon Nitride”)
    (thermal_cond 48)
    (wear_performance 4)
    (oxid_performance 4)
    (bend_strength 701 900)
    (unit_cost 2.25)
    (density 0.001)
    )
```

The first name in the description is an internal name for the system. The name that is displayed to the user is stored in the `full_name` slot. In order to change some of the specifications of the material, just modify the values and save the file to disk again. If you want to add a material, use one of the descriptions as a template, by copying it and modifying the values. Make sure you change the internal name of the material to be distinct from all the other materials, otherwise the second of the two descriptions with the same name overrides the first one.

A.5 Starting the Design Expert System

In order to start the SINE system and the application that you want to use, you must change to the appropriate directory first. For the I3D+ simulation example, use the following command:

```
cd ~airg/SINE/src <CR>
```

Now, if the developer has installed the system correctly, you can simply start the SINE platform by typing:

```
start_sine <CR>
```

It will also load the user interface. (Note that in order to run the SINE system with the windowing user interface, you need to have the MOTIF libraries available on the computer where the software is executed. If this is not the case, you can run the system in text-mode, by calling `start_local_sine` instead.) The SINE system will automatically look for the file `system.bat` from which it reads the information about what agents and data files to load.

Now, the system is ready to start processing.

A.6 Activating the Design Process

In the window in which you started the SINE platform, you will find the following prompt:

```
CLIPS>
```

Type the following command to start the design process:

```
(Design) <CR>
```

This command calls the agenda system, which will automatically schedule the agents and execute the design process. After every execution cycle, the agenda will prompt you for input:

```
Press <return> to continue, or ‘q’ to quit.
```

If you press <CR> the program will continue with the next cycle. If you press the ‘q’ key, the agenda will terminate the current design cycle:

```
agenda: Main Agenda finished
agenda: **** Finished ****
agenda: Running time was 129 milliseconds.
0
CLIPS>
```

At this point, you can investigate values in the system by using the CLIPS commands (please refer to the CLIPS Manuals for further information [Giarratano & Riley 1993] [Giarratano & Riley 1994]). You can also continue the design process, by restarting the agenda:

```
(Design) <CR>
```

You can exit the SINE system altogether:

```
(Shutdown) <CR>
```

This will bring you back to the UNIX command line. Notice that the I3D+ simulation does not save any data or decisions. Now, use the ‘Exit’ option in the ‘File’ menu of the windowing interface to quit the user interface.

If you want to modify the requirements and/or the data, proceed as described in sections A.3, “Requirement Specification”, and A.4, “Changing the Data Files”, and then start the SINE system again.

A.7 Using the SINE Multiwindow Interface

The window interface is run as a separate process from the SINE platform. As it receives all the data from the platform, it is called a *client*, and SINE is its *server*. When the interface client starts, several windows appear. The number is dependent on the application that you are using, but for the I3D+ simulation it is eight. There will be a window for every agent type in the system, as well as a ‘Main’ window and an ‘Agenda’ window.

The agent windows display all the information that the agents produce, the agenda window prints out the messages from agenda scheduler, and the main window prints out general messages, such as changes in parameter values etc.

You can clear the window contents by clicking on the ‘Clear’ button in the Interface Main Menu Window (see Figure A-1). In the same window, you can also use the ‘Font’ and

FIGURE A-1: **SINE Interface Main Menu Window**

‘Style’ options for changing the type of font that is used in the output.

If, for some reason, the client fails to connect to the server, use the ‘Connect’ option in the ‘Communications’ menu, to reattempt a connection to the server. If this fails too, exit the client through the ‘Exit’ option in the ‘File’ menu. Exit the SINE server too, as described in the previous section. Then, wait for 1 to 2 minutes, and restart the SINE system. SINE waits for the interface to automatically start and connect, before it continues loading.

APPENDIX B

Developers' Guide

B.1 Introduction

This guide is intended to help the developer of a design expert system using SINE through the process of designing the application. It is not a description of the system architecture or the object classes and functions available in SINE. Please refer to chapter 4, "SINE: A Platform for Negotiating SiFAs", and chapter 5, "Implementation of SINE", for more information about how the platform works. Also, read through the `readme.txt` file which is located in the SINE home directory (`~airg/SINE`).

B.2 Problem Definition

Probably the best way to define a design expert system with SINE is to think of it in a three step approach. First, analyze the design problem. Make sure, you know what knowledge is needed to solve the problem. Then, structure it by analyzing the design functions, targets and points of view in the problem and the expert knowledge. The last step is to derive what the parameters are, i.e., refine the targets into structured design parameters and specify the attributes that the parameters should have.

B.2.1 Problem Analysis

First, you should analyze what the structure of the design object is and what functions have to be performed to produce the needed information. You can use the agent type definition in 3.2, “Agent Types”, for reference. Try grouping the different tasks that you find by the kind of function, as well as by what values they determine and how the values interact.

Then, derive the points of view from the expertise that you gathered during knowledge acquisition. Often, one expert uses multiple points of view during his/her work. If points of view have to be traded against one another, keep them separate and plan to have the negotiation taking care of the conflict.

B.2.2 Specification of Functions, Targets and Points of View

Now, you should be able to list all the functions that you will need, all the targets that will have to be acted on, and all the points of view, from which actions will be taken in the system. Depending on the size of the problem, a large number from around two to three times the number of parameters, up to the maximum of the product of the number of parameters, targets and points-of-view is possible. Any number of points of view above 2 can be expected. Less than two points of view will probably not lead to much negotiation.

If two points of view are closely related and non-conflicting, then consider unifying them. That will make further development easier.

You will have to define a class for each target that will be used by the agents, for example:

```
(defclass material_target
  “Superclass of all agents with material as target”
  (is-a target)
  (slot target_full_name
            (source composite)
            (default “Material”))
```

```
  (slot target_name
            (source composite)
            (default MAIN::material))
  )
```

The points of view can only be defined after the design parameters are defined.

B.2.3 Design Parameter Definition

Analyze all targets, to find those that are directly dependent on others. If a target is always directly derived from another one, consider making the second target an attribute of the first one. Using this technique, you should be able to define structures for parameters, e.g., in the I3D domain, the mass and the cost are derived from the material type. Therefore they are stored as attributes of the material. If the material is changed in the design process, the assertion of the new material will automatically remove all the old information about mass and cost from the database.

Now design CLIPS object structures that can hold all the information that will need to be stored in each parameter. For example, attribute values from the definitions of the alternatives, as well as derived information that the agents will produce will need to be stored in the material. For the I3D+ simulation, we used the following structure:

```
(defclass material_type
  “Description for material types, derived from I3D+”
  (is-a target_type)
  (role concrete)
  (pattern-match reactive)
  (slot strength
            (access read-write)
            (create-accessor read-write)
            (type NUMBER)
            (visibility public)
            (default 0)
            )
  (slot thermal_cond
            (access read-write)
            (create-accessor read-write)
```

```
            (type NUMBER)
            (visibility public)
            (default 0)
            )
  (slot wear_performance
            (access read-write)
            (create-accessor read-write)
            (type INTEGER)
            (visibility public)
            (default 0)
            )
  (slot oxid_performance
            (access read-write)
            (create-accessor read-write)
            (type INTEGER)
            (visibility public)
            (default 0)
            )
  ;; Bend Strength as a range (min .. max)
  (multislot bend_strength
            (access read-write)
            (create-accessor read-write)
            (type INTEGER)
            (visibility public)
            (default 0 0)
            )
  ;; Unit cost in $/kg
  (slot unit_cost
            (access read-write)
            (create-accessor read-write)
            (type NUMBER)
            (visibility public)
            (default 0)
            )
  ;; Density in kg/ccm
  (slot density
            (access read-write)
            (create-accessor read-write)
            (type NUMBER)
            (visibility public)
            (default 0)
            )
  ;; Mass in kg
  (slot mass
            (access read-write)
            (create-accessor read-write)
            (type NUMBER)
```

```
            (visibility public)
            (default 0)
            )
  ;; Total material cost as derived from unit cost and density
  (slot part_cost
            (access read-write)
            (create-accessor read-write)
            (type NUMBER)
            (visibility public)
            (default 0)
            )
  ;; Total material cost per batch
  (slot cost
            (access read-write)
            (create-accessor read-write)
            (type NUMBER)
            (visibility public)
            (default 0)
            )
  )
```

Please refer to the CLIPS Manuals for further information about class declarations [Giarratano & Riley 1993] [Giarratano & Riley 1994].

Now the points of view can be defined. Each point of view requires one class, for example:

```
(defclass bend_strength_pov
  “Superclass of all agents with bend-strength as point-of-view”
  (is-a pov)
  (slot pov_full_name
            (source composite)
            (default “Bend-Strength”))
  (slot attribute  ; the attribute that the agent watches
            (source composite)
            (default bend_strength))
  )
```

The attribute specifies what slot of the design parameter structure the agents will be retrieving and/or storing values in.

B.3 Implementation

B.3.1 Building Agents

In order to define the types of agents in the system, a class has to be generated for each agent. This can be done through multiple inheritance, which allows information from a set of parent classes to be reused by the agent class. For example, the Material Bend-Strength Selector was defined as follows:

```
(defclass MaterialBendStrengthSelector
  “Selector for Beam Material by strength”
  (is-a selector material_target bend_strength_pov)
  (role concrete)
  (pattern-match reactive)
  (slot design_rules
            (source composite)
            (default selector))
  (slot state
            (source composite)
            (default ready))
  (slot router
            (source composite)
            (default mbs))
  )
```

Also, we have to generate an object instance for each agent, for example:

```
(mbs of MaterialBendStrengthSelector)
```

If we need constraints and/or preferences in the agent, we specify them before the agent object, e.g.:

```
(high_bendstrength_pref of larger_comp_pref
  (full_name “Preference for high bend-strength”)
  (attribute bend_strength))

(bendstrength_con of alt_range_constraint
  (full_name “Constraint on the range of the bend-strength”)
  (objects [material] [req_material])
  (attributes bend_strength bend_strength))
```

Notice that no predicate has to be defined for the constraint, as it uses the default predicate for the alt_range_constraint class (see 5.3, “Classes and Inheritance”). The references to the preference and constraint objects can be stored directly in the agent’s instance description:

```
(mbs of MaterialBendStrengthSelector
  (preferences [high_bendstrength_pref])
  (constraints [bendstrength_con])
  )
```

B.3.2 Design Knowledge

We have already discussed how constraints and preferences can be stored in the knowledge base of an agent. If more flexible knowledge representation is needed, rules can be used. For example, the Material Cost Estimator uses the following rule to derive the object mass:

```
(defrule mce::compute_mass
  (current_instance mce ?self)
  (not (failed_constraint ?self density_con))
  (part_volume ?vol)
  =>
  (bind ?material (get ?self target_obj))
  (bind ?mass (* (get ?material density) ?vol))
  (put ?material mass ?mass)
  (printout (get ?self router)
               “The mass of the part made from “
               (get ?material full_name) “ is “
               ?mass “ kg.”  crlf)
  )
```

The `(current_instance mce ?self)` fact is asserted automatically by the estimator, so that rules have a reference to the object that invoked them. The `(part_volume ?vol)` facts is asserted in the `mce.fac` fact file. In the rule, the mass is computed by multiplying the density of the material with the declared part volume. The result is stored in the material parameter.

All the design knowledge is stored in a file with the name of the module, e.g., the design knowledge for the Material Cost Estimator is stored in `mce.clp`.

B.3.3 Negotiation Knowledge

You can use the domain independent negotiation knowledge as a basis for developing your own CR knowledge. The `cr_rules` slot in the agent class references the rule module that contains all the negotiation rules for the agent. In the I3D+ simulation no agent has its own CR knowledge, but all agents use the CR modules provided for their agent function type. For the two selectors, for example, the `selector` CR module is used. The Material Cost Estimator uses the `estimator` rule module, etc. The rules are stored in a file with the name of the module, e.g., the estimator CR knowledge is stored in `estimator.clp`.

B.3.4 Building the Data Base

Since we already have defined the class description of each parameter type (see section B.2.3), we can simply store a list with the descriptions of the alternatives in a file. Each alternative is described by partially filling the slots of the class. For the I3D+ simulation a material description has the following form:

```
(SN of material_type
    (object_type alternative)
    (full_name “Silicon Nitride”)
    (thermal_cond 48)
    (wear_performance 4)
    (oxid_performance 4)
    (bend_strength 701 900)
    (unit_cost 2.25)
    (density 0.001)
    )
```

All the material descriptions are stored in the `materials.ins` file.

B.3.5 Defining Routers

In order to print out the messages from the agents to individual windows, SINE implements an extension of the CLIPS I/O Routers. An agent sends its output to the ‘main’ router by default. This can be changed by the storing a different router name in the `router` slot of the agent’s class or instance-declaration.

Whenever a new router name is introduced, the router has to be declared to CLIPS. The `InitRouter` function in the file `sifa.c` performs this. Just add the new name to the existing list, by adding a line of the following form into the function:

```
if (strcmp (logicalName, "<router-name>") == 0) return CLIPS_TRUE;
```

B.3.6 Configuring the Interface

In order to implement an additional window in the client, follow the instructions in `client_setup.txt` in the `client` subdirectory.

B.4 Setup File

SINE automatically loads up a CLIPS batch file called `system.bat`. This file instructs SINE what other files to load and what agents to generate. It should contain the instructions described in the following.

- A declaration of all the object names of the agents in the application:

```
(defglobal MAIN ?*agents* = (create$ mbs mts mce mcc mwv mov))
```

- Instructions for loading all the domain independent files:

```
(load "standard.clp")
(load "sifa.cla")
(load "sifa.clp")
(load "agent.cla")
(load "agenda.cla")
```

```
(load “targets.cla”)
(load “povs.cla”)
(focus MAIN)
(load “selector.cla”)
(load “selector.clp”)
(focus MAIN)
(load “critic.cla”)
(load “critic.clp”)
(focus MAIN)
(load “estimator.cla”)
(load “estimator.clp”)
(focus MAIN)
(load “evaluator.cla”)
(load “evaluator.clp”)
(focus MAIN)
```

- Commands for initializing the agents’ knowledge bases (by loading the `*.clp` and `*.cla` files) and routers:

```
(init-agent-routers ?*agents*)
(load-agent-kbs ?*agents*)
(reset)
```

- An instruction for loading the definition of the agenda entry types and their ranks:

```
(load-instances “agenda.ins”)
```

- Commands for loading the description of the applications design parameters and alternatives:

```
(load-instances “parameters.ins”)
(load-instances “materials.ins”)
```

- An instruction to load the file with the requirement specification:

```
(load-instances “requirements.ins”)
```

- And finally, the load command to load the agents knowledge bases (`*.fac` and `*.ins` files)

```
(load-agent-wms ?*agents*)
```

B.5 Testing

After the developer has produced all the files that are needed for the application, the platform can be loaded. Please refer to section A.5, “Starting the Design Expert System”, for information about how to start the system. After the initializtion, the design process can be initiated through the `(Design)` command, or individual agents can be started by using the following instruction:

`(send [<agent-name>] run)`

For example to run the Material Thermal-Conductivity Selector only, you can type:

```
(send [mts] run)
```

Please refer to the `README` file in the SINE directory for further information and updates.

APPENDIX C

Negotiation Output Traces

C.1 Introduction

The following sections present two kinds of traces. The first are selected annotated outputs from both the simulation of I3D+ (from section 6.3) and the example with derived attributes (from section 6.4).

The second section contains a full run performed with the I3D+ simulation with the original material types. The third section contains the entire output of a run with the simulation, using material choices that lack certain information attributes.

C.2 Annotated Conflicts

C.2.1 Selector-Selector Conflict

This trace offers a view into a typical conflict between two selectors. The current situation is that the BendStrength Selector had previously selected a material. That selection had already been revised once when the Thermal-Conductivity Selector ran for the first time. In the meantime, the other selector had to revise its choice, due to constraints by other agents. Now, MTS notices that the value has changed from its previous selection, so it scheduled an agenda entry for ‘conflict’. It will now re-negotiate the new selection.

agenda: **** Schedule for Cycle 5 ****
agenda: Material Thermal-Conductivity Selector because of conflict, ranked at 100
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 5 ****
agenda: Running Material Thermal-Conductivity Selector because of conflict
mts Starting cycle 5
mts: Additionals:()
mts: Forbiddens:()
mts: Alts+Adds-Forb= ([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Looking for best choice in:([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Selecting Silicon Carbide No-Cost [SC0] for Material Parameter
main: Material Thermal-Conductivity Selector cannot store Silicon Carbide No-Cost in Material Parameter, because Material Bend-Strength Selector [MAIN::mts] had already stored Silicon Carbide No-Wear.
mts: selection_selection conflict detected with Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mbs: Got asked by Material Thermal-Conductivity Selector for number of proposals, replying with:3
mbs: Sending message to Material Thermal-Conductivity Selector
mts: Soliciting another proposal from Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mbs: Got asked by Material Thermal-Conductivity Selector for a different proposal, replying with:[SC2]
mbs: Sending message to Material Thermal-Conductivity Selector
mts: Agreeing with alternate proposal [SC2] from agent Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mbs: Modifying design value to accept proposed change from agent Material Thermal-Conductivity Selector
main: Material Bend-Strength Selector overrides old value Silicon Carbide No-Wear in Material Parameter with Silicon Carbide No-Oxid
mts: Finished cycle 5

C.2.2 Estimator-Selector Conflict

This is a typical case of an evaluator whose precondition constraints are not satisfied. Thus it complains to the selector about the constraint failure and passes it the constraint for consideration in future selections.

agenda: **** Schedule for Cycle 3 ****
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 3 ****
agenda: Running Material Cost Estimator because of usage
mce: Starting cycle 3
mce: Constraint:Constraint for valid density satisfied.
mce: Constraint:Constraint for valid unit cost failed.
mce: selection_estimation conflict detected with Material Bend-Strength Selector
mce: Current material 'Silicon Carbide No-Cost' violated the Constraint for valid unit cost
mce: Requesting Material Bend-Strength Selector to insert the [unit_cost_con] to his list of constraints
mce: Sending message to Material Bend-Strength Selector
mbs: Got told by Material Cost Estimator to insert constraint [unit_cost_con], into his list.
mbs: Adding Constraint for valid unit cost to Material Bend-Strength Selector
mbs: Changing state from waiting to ready
mce: The mass of the part made from Silicon Carbide No-Cost is 0.005 kg.
mce: The cost for the object made from Silicon Carbide No-Cost is 0.0 $ per part.
mce: Finished cycle 3
mce:
mce: Changing state from ready to waiting

C.2.3 Critic-Selector Conflict

In this trace the critic notices that the material cost is not low enough. After questioning the MaterialCostEstimator to make sure that it uses the 'detailed' estimation process, it proceeds to request that the selector not to use that particular choice anymore.

agenda: **** Schedule for Cycle 6 ****
agenda: Material Cost Critic because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 6 ****
agenda: Running Material Cost Critic because of usage
mcc: Starting cycle 6
mcc: Constraint:Constraint on the Material Cost failed.
mcc: Requesting estimate process info from Material Cost Estimator
mcc: Sending message to Material Cost Estimator

mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘detailed’
mce: Sending message to Material Cost Critic
mcc: Estimate quality of Material Cost Estimator is acceptable.
mcc: selection_criticism conflict detected with Material Bend-Strength Selector
mcc: Requesting Material Bend-Strength Selector not to use the [SN] alternative
mcc: Sending message to Material Bend-Strength Selector
mbs: Got told by Material Cost Critic to delete alternative [SN], from his list.
mbs: Removing alternative Silicon Nitride from Material Bend-Strength Selector
mbs: Changing state from waiting to ready
mcc: Finished cycle 6
mcc:
mcc: Changing state from ready to waiting

C.2.4 Critic-Estimator Conflict

This trace shows how the critic notices that the estimation process is not good enough. It asks the estimator to use the more detailed process.

agenda: **** Schedule for Cycle 7 ****
agenda: Material Cost Critic because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 7 ****
agenda: Running Material Cost Critic because of usage
mcc: Starting cycle 7
mcc: Constraint:Constraint on the Material Cost satisfied.
mcc: Requesting estimate process info from Material Cost Estimator
mcc: Sending message to Material Cost Estimator
mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘rough’
mce: Sending message to Material Cost Critic
mcc: estimate_criticism conflict detected with Material Cost Estimator
mcc: Requesting Material Cost Estimator to use the detailed process.
mcc: Sending message to Material Cost Estimator
mce: Got asked by Material Cost Critic to change its current process to detailed
mce: Changing process for Material Cost Estimator to detailed
mce: Sending message to Material Cost Critic
mcc: Received acceptance from Material Cost Estimator to use the detailed process
mcc: Finished cycle 7
mcc:
mcc: Changing state from ready to waiting

C.2.5 Evaluator-Estimator Conflict

In this trace, an evaluator notices that one of the materials it is evaluating does not have the needed information about reliabilty associated with it. It complains to the Material Thermal-Conductivity Estimator and requests that it add a reliability attribute to the materials.

```
agenda:  **** Schedule for Cycle 3 ****
agenda:  Material Thermal-Conductivity-Evaluation Evaluator because of usage, ranked
         at  60
agenda:  **** Executing Cycle 3 ****
agenda:  Running Material Thermal-Conductivity-Evaluation Evaluator because of usage
mev:     Starting cycle 3
mev:     Updating evaluations
mev:     Preference:Quantified preference for high thermal conductivity evaluates to 0.0
mev:     Material 'Zirconia no-density' has no reliability value.
mev:     estimation_evaluation conflict detected with Material Thermal-Conductivity Esti-
         mator
mev:     Requesting Material Thermal-Conductivity Estimator to add the reliability value
mev:     Sending message to Material Thermal-Conductivity Estimator
mte:     Got asked by Material Thermal-Conductivity-Evaluation Evaluator to acquire
         data about reliability
mte:     Sending message to Material Thermal-Conductivity-Evaluation Evaluator
mte:     Changing state from waiting to ready
mev:     Received acceptance from Material Thermal-Conductivity Estimator to achieve
         reliability -1
mev:     Finished cycle 3
mev:
mev:     Changing state from ready to waiting
```

In the next trace, from the same run, the evaluator requests the estimator to estimate with an error value less than 10%. This prompts the estimator to change to a detailed process.

```
agenda:   **** Schedule for Cycle 5 ****
agenda:  Material Thermal-Conductivity-Evaluation Selector because of selection, ranked
         at  50
agenda:  Material Thermal-Conductivity-Evaluation Evaluator because of usage, ranked
         at  60
agenda:   **** Executing Cycle 5 ****
agenda:  Running Material Thermal-Conductivity-Evaluation Evaluator because of usage
mev:     Starting cycle 5
```

mev: Updating evaluations
mev: Preference: Quantified preference for high thermal conductivity evaluates to 0.0
mev: Material ‘Zirconia no-density’ has poor error value.
mev: estimation_evaluation conflict detected with Material Thermal-Conductivity Estimator
mev: Requesting Material Thermal-Conductivity Estimator lower the error value
mev: Sending message to Material Thermal-Conductivity Estimator
mte: Got asked by Material Thermal-Conductivity-Evaluation Evaluator to achieve 10% error
mte: Removed old memo object
mte: Sending message to Material Thermal-Conductivity-Evaluation Evaluator
mte: Changing process for Material Thermal-Conductivity Estimator to detailed
mte: Got asked by Material Thermal-Conductivity-Evaluation Evaluator to acquire data about error
mte: Sending message to Material Thermal-Conductivity-Evaluation Evaluator
mev: Received acceptance from Material Thermal-Conductivity Estimator to achieve error 10
mev: Finished cycle 5
mev:
mev: Changing state from ready to waiting

C.3 I3D+ Conflict Simulation

The following trace is a simulation of a typical I3D+ task. The materials are identical in all aspects to the materials used in I3D. The requirements are set up so that the two selectors go into conflict and resolve their dispute. The cost maximum cannot be satisfied, which leads the critic to request that the selector propose another material, which it cannot, since the material requirements are too constrained.

```
(req_material of material_type
        (object_type requirement)
        (full_name “Requirement on Material”)
        (bend_strength 800)
        (wear_performance 1)
        (oxid_performance 1)
        (cost 17)
        )
```

```
;; Material Objects
;; derived from I3D+
;; using very poor = 1, poor = 2, average = 3, good = 4, very good = 5

(SC of material_type
   (object_type alternative)
   (full_name “Silicon Carbide”)
   (thermal_cond 50)
   (bend_strength 401 700)
   (wear_performance 1)
   (oxid_performance 2)
   (unit_cost 2.35)
   (density 0.001)
   )

(SN of material_type
   (object_type alternative)
   (full_name “Silicon Nitride”)
   (thermal_cond 48)
   (wear_performance 4)
   (oxid_performance 4)
   (bend_strength 701 900)
   (unit_cost 2.25)
   (density 0.001)
   )

(AL of material_type
   (object_type alternative)
   (full_name “Alumina”)
   (thermal_cond 49)
   (wear_performance 5)
   (bend_strength 301 400)
   (unit_cost 2.55)
   (density 0.001)
   )

(ZR of material_type
   (object_type alternative)
   (full_name “Zirconia”)
   (thermal_cond 49)
   (wear_performance 4)
   (oxid_performance 4)
   (bend_strength 901 1200)
```

```
   (unit_cost 2.45)
   (density 0.001)
   )

agenda:  **** Schedule for Cycle 1 ****
agenda:  Material Bend-Strength Selector because of selection, ranked at  70
agenda:  Material Thermal-Conductivity Selector because of selection, ranked at  70
agenda:  **** Executing Cycle 1 ****
agenda:  Running Material Bend-Strength Selector because of selection
mbs:     Starting cycle 1
mbs:     Changing state from ready to waiting
mbs:     Additionals:()
mbs:     Forbiddens:()
mbs:     Alts+Adds-Forb= ([SC] [SN] [AL] [ZR])
mbs:     Remaining:([SC] [SN] [AL] [ZR])
mbs:     Checking constraint:[bendstrength_con]
mbs:     Looking for best choice in:([SN])
mbs:     Selecting Silicon Nitride [SN] for Material Parameter
main:    Initializing Material Parameter with Silicon Nitride from Material Bend-Strength
         Selector
mov:     Changing state from waiting to ready
mwv:     Changing state from waiting to ready
mce:     Changing state from waiting to ready
mbs:     Material Bend-Strength Selector was able to select his preferred value for Mate-
         rial Parameter
mbs:     Finished cycle 1
mbs:
Press <return> to continue, or ‘q’ <return> to quit.

agenda:  **** Schedule for Cycle 2 ****
agenda:  Material Thermal-Conductivity Selector because of selection, ranked at  70
agenda:  Material Cost Estimator because of usage, ranked at  60
agenda:  Material Wear-Performance Evaluator because of usage, ranked at  60
agenda:  Material Oxidation-Performance Evaluator because of usage, ranked at  60
agenda:  **** Executing Cycle 2 ****
agenda:  Running Material Thermal-Conductivity Selector because of selection
mts:     Starting cycle 2
mts:     Changing state from ready to waiting
mts:     Additionals:()
mts:     Forbiddens:()
mts:     Alts+Adds-Forb= ([SC] [SN] [AL] [ZR])
mts:     Looking for best choice in:([SC] [AL] [ZR] [SN])
mts:     Selecting Silicon Carbide [SC] for Material Parameter
```

main: Material Thermal-Conductivity Selector cannot store Silicon Carbide in Material Parameter, because Material Bend-Strength Selector [MAIN::mts] had already stored Silicon Nitride.
mts: selection_selection conflict detected with Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mbs: Got asked by Material Thermal-Conductivity Selector for number of proposals, replying with:1
mbs: Sending message to Material Thermal-Conductivity Selector
mts: Adapting to the only proposal from Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mts: Finished cycle 2
mts:
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 3 ****
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 3 ****
agenda: Running Material Cost Estimator because of usage
mce: Starting cycle 3
mce: Constraint:Constraint for valid density satisfied.
mce: Constraint:Constraint for valid unit cost satisfied.
mce: The mass of the part made from Silicon Nitride is 0.005 kg.
mce: The cost for the object made from Silicon Nitride is 0.01125 $ per part.
mce: The total material cost for Silicon Nitride is roughly $ 22.5 for a batch size of 2000.
mce: Finished cycle 3
mce:
mce: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 4 ****
agenda: Material Cost Critic because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 4 ****
agenda: Running Material Cost Critic because of usage
mcc: Starting cycle 4
mcc: Constraint:Constraint on the Material Cost failed.
mcc: Requesting estimate process info from Material Cost Estimator
mcc: Sending message to Material Cost Estimator

mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘rough’
mce: Sending message to Material Cost Critic
mcc: estimate_criticism conflict detected with Material Cost Estimator
mcc: Requesting Material Cost Estimator to use the detailed process.
mcc: Sending message to Material Cost Estimator
mce: Got asked by Material Cost Critic to change its current process to detailed
mce: Changing process for Material Cost Estimator to detailed
mce: Sending message to Material Cost Critic
mcc: Received acceptance from Material Cost Estimator to use the detailed process
mcc: Finished cycle 4
mcc:
mcc: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 5 ****
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 5 ****
agenda: Running Material Cost Estimator because of usage
mce: Starting cycle 5
mce: Constraint:Constraint for valid density satisfied.
mce: Constraint:Constraint for valid unit cost satisfied.
mce: The mass of the part made from Silicon Nitride is 0.005 kg.
mce: The cost for the object made from Silicon Nitride is 0.01125 $ per part.
mce: The total material cost for Silicon Nitride is exactly $18.0 for a batch size of 2000.
mce: Finished cycle 5
mce:
mce: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 6 ****
agenda: Material Cost Critic because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 6 ****
agenda: Running Material Cost Critic because of usage
mcc: Starting cycle 6
mcc: Constraint:Constraint on the Material Cost failed.
mcc: Requesting estimate process info from Material Cost Estimator
mcc: Sending message to Material Cost Estimator

mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘detailed’
mce: Sending message to Material Cost Critic
mcc: Estimate quality of Material Cost Estimator is acceptable.
mcc: selection_criticism conflict detected with Material Bend-Strength Selector
mcc: Requesting Material Bend-Strength Selector not to use the [SN] alternative
mcc: Sending message to Material Bend-Strength Selector
mbs: Got told by Material Cost Critic to delete alternative [SN], from his list.
mbs: Removing alternative Silicon Nitride from Material Bend-Strength Selector
mbs: Changing state from waiting to ready
mcc: Finished cycle 6
mcc:
mcc: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 7 ****
agenda: Material Bend-Strength Selector because of selection, ranked at 70
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 7 ****
agenda: Running Material Bend-Strength Selector because of selection
mbs: Starting cycle 7
mbs: Changing state from ready to waiting
mbs: Additionals:()
mbs: Forbiddens:([SN])
mbs: Alts+Adds-Forb= ([SC] [AL] [ZR])
mbs: Remaining:([SC] [AL] [ZR])
mbs: Checking constraint:[bendstrength_con]
mbs: Selector Material Bend-Strength Selector out of choices.
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 8 ****
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 8 ****
agenda: Running Material Wear-Performance Evaluator because of usage
mwv: Starting cycle 8
mwv: Constraint:Constraint for valid wear_performance satisfied.
mwv: Preference:Preference for high wear-performance evaluates to 4.0
mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘detailed’
mce: Sending message to Material Cost Critic

```
mce:    Got asked by Material Cost Critic for the name of its current process. Replying
        with 'detailed'
mce:    Sending message to Material Cost Critic
mwv:    Finished cycle 8
mwv:
mwv:    Changing state from ready to waiting
mbs:    Finished cycle 8
mbs:
Press <return> to continue, or 'q' <return> to quit.

agenda:  **** Schedule for Cycle 9 ****
agenda:  Material Oxidation-Performance Evaluator because of usage, ranked at  60
agenda:  **** Executing Cycle 9 ****
agenda:  Running Material Oxidation-Performance Evaluator because of usage
mov:    Starting cycle 9
mov:    Constraint:Constraint for valid oxid_performance satisfied.
mov:    Preference:Preference for high oxid-performance evaluates to 4.0
mov:    Finished cycle 9
mov:
mov:    Changing state from ready to waiting
Press <return> to continue, or 'q' <return> to quit.

agenda:  **** Schedule for Cycle 10 ****
agenda:  Main Agenda empty
agenda:  **** AGENDA EMPTY ****
agenda:  Main Agenda finished
agenda:  **** Finished ****
agenda:  Running time was 0 seconds.
0
CLIPS>
```

C.4 I3D+ Conflict Simulation with Information Gaps

For the next trace, more material types have been added to the database. These materials are clones of I3D materials, but they are lacking some information, e.g., the wear_performance, oxid_performance, unit_cost, or density attribute is not filled in. This will lead to several conflicts between the selectors and estimators/evaluators, which will eventually get solved.

```
(req_material of material_type
     (object_type requirement)
     (full_name “Requirement on Material”)
     (bend_strength 500)
     (wear_performance 2)
     (oxid_performance 4)
     (cost 50)
     )

(SC0 of material_type
   (object_type alternative)
   (full_name “Silicon Carbide No-Cost”)
   (thermal_cond 51)
   (bend_strength 401 700)
   (density 0.001)
   )

(SC1 of material_type
   (object_type alternative)
   (full_name “Silicon Carbide No-Wear”)
   (thermal_cond 52)
   (bend_strength 401 700)
   (unit_cost 2.35)
   (density 0.001)
   )

(SC2 of material_type
   (object_type alternative)
   (full_name “Silicon Carbide No-Oxid”)
   (thermal_cond 52)
   (bend_strength 401 700)
   (wear_performance 1)
   (unit_cost 2.35)
   (density 0.001)
   )

(SC of material_type
   (object_type alternative)
   (full_name “Silicon Carbide”)
   (thermal_cond 50)
   (bend_strength 401 700)
   (wear_performance 1)
   (oxid_performance 2)
```

```
   (unit_cost 2.35)
   (density 0.001)
   )

(SN0 of material_type
   (object_type alternative)
   (full_name “Silicon Nitride - No Wear Info”)
   (thermal_cond 49)
   (oxid_performance 4)
   (bend_strength 701 900)
   (unit_cost 2.25)
   (density 0.001)
   )

(SN1 of material_type
   (object_type alternative)
   (full_name “Silicon Nitride - No density info”)
   (thermal_cond 50)
   (wear_performance 1)
   (oxid_performance 4)
   (bend_strength 701 900)
   (unit_cost 0.001)
   )

(SN2 of material_type
   (object_type alternative)
   (full_name “Silicon Nitride - No unit cost”)
   (thermal_cond 50)
   (wear_performance 0)
   (oxid_performance 4)
   (bend_strength 701 900)
   (density 0.001)
   )

(SN of material_type
   (object_type alternative)
   (full_name “Silicon Nitride”)
   (thermal_cond 48)
   (wear_performance 4)
   (oxid_performance 4)
   (bend_strength 701 900)
   (unit_cost 2.25)
   (density 0.001)
```

```
   )

(AL of material_type
   (object_type alternative)
   (full_name “Alumina”)
   (thermal_cond 49)
   (wear_performance 5)
   (bend_strength 301 400)
   (unit_cost 2.55)
   (density 0.003) ;; fake
   )

(ZR of material_type
   (object_type alternative)
   (full_name “Zirconia”)
   (thermal_cond 49)
   (wear_performance 4)
   (oxid_performance 4)
   (bend_strength 901 1200)
   (unit_cost 2.45)
   (density 0.002) ;; fake
   )

(ZR0 of material_type
   (object_type alternative)
   (full_name “Zirconia no-density”)
   (thermal_cond 49)
   (wear_performance 4)
   (oxid_performance 4)
   (bend_strength 901 1200)
   (unit_cost 2.45)
   )

agenda:  **** Schedule for Cycle 1 ****
agenda:  Material Bend-Strength Selector because of selection, ranked at  70
agenda:  Material Thermal-Conductivity Selector because of selection, ranked at  70
agenda:  **** Executing Cycle 1 ****
agenda:  Running Material Bend-Strength Selector because of selection
mbs:     Starting cycle 1
mbs:     Changing state from ready to waiting
mbs:     Additionals:()
mbs:     Forbiddens:()
```

mbs: Alts+Adds-Forb= ([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mbs: Remaining:([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mbs: Checking constraint:[bendstrength_con]
mbs: Looking for best choice in:([SC0] [SC1] [SC2] [SC])
mbs: Selecting Silicon Carbide No-Cost [SC0] for Material Parameter
main: Initializing Material Parameter with Silicon Carbide No-Cost from Material Bend-Strength Selector
mov: Changing state from waiting to ready
mwv: Changing state from waiting to ready
mce: Changing state from waiting to ready
mbs: Material Bend-Strength Selector was able to select his preferred value for Material Parameter
mbs: Finished cycle 1
mbs:
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 2 ****
agenda: Material Thermal-Conductivity Selector because of selection, ranked at 70
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 2 ****
agenda: Running Material Thermal-Conductivity Selector because of selection
mts: Starting cycle 2
mts: Changing state from ready to waiting
mts: Additionals:()
mts: Forbiddens:()
mts: Alts+Adds-Forb= ([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Looking for best choice in:([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Selecting Silicon Carbide No-Cost [SC0] for Material Parameter
main: Material Thermal-Conductivity Selector agrees with Material Bend-Strength Selector about value in object Material Parameter
mts: Material Thermal-Conductivity Selector was able to select his preferred value for Material Parameter
mts: Finished cycle 2
mts:
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 3 ****
agenda: Material Cost Estimator because of usage, ranked at 60

agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 3 ****
agenda: Running Material Cost Estimator because of usage
mce: Starting cycle 3
mce: Constraint:Constraint for valid density satisfied.
mce: Constraint:Constraint for valid unit cost failed.
mce: selection_estimation conflict detected with Material Bend-Strength Selector
mce: Current material 'Silicon Carbide No-Cost' violated the Constraint for valid unit cost
mce: Requesting Material Bend-Strength Selector to insert the [unit_cost_con] to his list of constraints
mce: Sending message to Material Bend-Strength Selector
mbs: Got told by Material Cost Estimator to insert constraint [unit_cost_con], into his list.
mbs: Adding Constraint for valid unit cost to Material Bend-Strength Selector
mbs: Changing state from waiting to ready
mce: The mass of the part made from Silicon Carbide No-Cost is 0.005 kg.
mce: The cost for the object made from Silicon Carbide No-Cost is 0.0 $ per part.
mce: Finished cycle 3
mce:
mce: Changing state from ready to waiting
Press <return> to continue, or 'q' <return> to quit.

agenda: **** Schedule for Cycle 4 ****
agenda: Material Bend-Strength Selector because of selection, ranked at 70
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 4 ****
agenda: Running Material Bend-Strength Selector because of selection
mbs: Starting cycle 4
mbs: Changing state from ready to waiting
mbs: Additionals:()
mbs: Forbiddens:()
mbs: Alts+Adds-Forb= ([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mbs: Remaining:([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mbs: Checking constraint:[unit_cost_con]
mbs: Remaining:([SC1] [SC2] [SC] [SN0] [SN1] [SN] [AL] [ZR] [ZR0])
mbs: Checking constraint:[bendstrength_con]
mbs: Looking for best choice in:([SC1] [SC2] [SC])
mbs: Selecting Silicon Carbide No-Wear [SC1] for Material Parameter

main: Material Bend-Strength Selector overrides old value Silicon Carbide No-Cost in Material Parameter with Silicon Carbide No-Wear
mce: Changing state from waiting to ready
mts: Current value Silicon Carbide No-Wear for target is different from preferred choice Silicon Carbide No-Cost, scheduling mts for conflict resolution.
mbs: Material Bend-Strength Selector was able to select his preferred value for Material Parameter
mbs: Finished cycle 4
mbs:
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 5 ****
agenda: Material Thermal-Conductivity Selector because of conflict, ranked at 100
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 5 ****
agenda: Running Material Thermal-Conductivity Selector because of conflict
mts: Starting cycle 5
mts: Additionals:()
mts: Forbiddens:()
mts: Alts+Adds-Forb= ([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Looking for best choice in:([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Selecting Silicon Carbide No-Cost [SC0] for Material Parameter
main: Material Thermal-Conductivity Selector cannot store Silicon Carbide No-Cost in Material Parameter, because Material Bend-Strength Selector [MAIN::mts] had already stored Silicon Carbide No-Wear.
mts: selection_selection conflict detected with Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mbs: Got asked by Material Thermal-Conductivity Selector for number of proposals, replying with:3
mbs: Sending message to Material Thermal-Conductivity Selector
mts: Soliciting another proposal from Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mbs: Got asked by Material Thermal-Conductivity Selector for a different proposal, replying with:[SC2]
mbs: Sending message to Material Thermal-Conductivity Selector
mts: Agreeing with alternate proposal [SC2] from agent Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector

mbs: Modifying design value to accept proposed change from agent Material Thermal-Conductivity Selector
main: Material Bend-Strength Selector overrides old value Silicon Carbide No-Wear in Material Parameter with Silicon Carbide No-Oxid
mts: Finished cycle 5
mts:
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 6 ****
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 6 ****
agenda: Running Material Cost Estimator because of usage
mce: Starting cycle 6
mce: Constraint:Constraint for valid density satisfied.
mce: Constraint:Constraint for valid unit cost satisfied.
mce: The mass of the part made from Silicon Carbide No-Oxid is 0.005 kg.
mce: The cost for the object made from Silicon Carbide No-Oxid is 0.01175 $ per part.
mce: The total material cost for Silicon Carbide No-Oxid is roughly $ 23.5 for a batch size of 2000.
mce: Finished cycle 6
mce:
mce: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 7 ****
agenda: Material Cost Critic because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 7 ****
agenda: Running Material Cost Critic because of usage
mcc: Starting cycle 7
mcc: Constraint:Constraint on the Material Cost satisfied.
mcc: Requesting estimate process info from Material Cost Estimator
mcc: Sending message to Material Cost Estimator
mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘rough’
mce: Sending message to Material Cost Critic
mcc: estimate_criticism conflict detected with Material Cost Estimator
mcc: Requesting Material Cost Estimator to use the detailed process.
mcc: Sending message to Material Cost Estimator
mce: Got asked by Material Cost Critic to change its current process to detailed

mce: Changing process for Material Cost Estimator to detailed
mce: Sending message to Material Cost Critic
mcc: Received acceptance from Material Cost Estimator to use the detailed process
mcc: Finished cycle 7
mcc:
mcc: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 8 ****
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 8 ****
agenda: Running Material Cost Estimator because of usage
mce: Starting cycle 8
mce: Constraint:Constraint for valid density satisfied.
mce: Constraint:Constraint for valid unit cost satisfied.
mce: The mass of the part made from Silicon Carbide No-Oxid is 0.005 kg.
mce: The cost for the object made from Silicon Carbide No-Oxid is 0.01175 $ per part.
mce: The total material cost for Silicon Carbide No-Oxid is exactly $18.8 for a batch size of 2000.
mce: Finished cycle 8
mce:
mce: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 9 ****
agenda: Material Cost Critic because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 9 ****
agenda: Running Material Cost Critic because of usage
mcc: Starting cycle 9
mcc: Constraint:Constraint on the Material Cost satisfied.
mcc: Requesting estimate process info from Material Cost Estimator
mcc: Sending message to Material Cost Estimator
mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘detailed’
mce: Sending message to Material Cost Critic
mcc: Estimate quality of Material Cost Estimator is acceptable.
mcc: Finished cycle 9
mcc:
mcc: Changing state from ready to waiting

Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 10 ****
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 10 ****
agenda: Running Material Wear-Performance Evaluator because of usage
mwv: Starting cycle 10
mwv: Constraint:Constraint for valid wear_performance satisfied.
mwv: Preference:Preference for high wear-performance evaluates to 0.5
mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘detailed’
mce: Sending message to Material Cost Critic
mce: Got asked by Material Cost Critic for the name of its current process. Replying with ‘detailed’
mce: Sending message to Material Cost Critic
mwv: Finished cycle 10
mwv:
mwv: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 11 ****
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 11 ****
agenda: Running Material Oxidation-Performance Evaluator because of usage
mov: Starting cycle 11
mov: Constraint:Constraint for valid oxid_performance failed.
mov: selection_evaluation conflict detected with Material Bend-Strength Selector
mov: Current material ‘Silicon Carbide No-Oxid’ violated the Constraint for valid oxid_performance
mov: Requesting Material Bend-Strength Selector to insert the [oxid_performance_con] to his list of constraints
mov: Sending message to Material Bend-Strength Selector
mbs: Got told by Material Oxidation-Performance Evaluator to insert constraint [oxid_performance_con], into his list.
mbs: Adding Constraint for valid oxid_performance to Material Bend-Strength Selector
mbs: Changing state from waiting to ready
mov: Preference:Preference for high oxid-performance evaluates to 0.0
mov: Finished cycle 11
mov:
mov: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 12 ****
agenda: Material Bend-Strength Selector because of selection, ranked at 70
agenda: **** Executing Cycle 12 ****
agenda: Running Material Bend-Strength Selector because of selection
mbs: Starting cycle 12
mbs: Changing state from ready to waiting
mbs: Additionals:()
mbs: Forbiddens:()
mbs: Alts+Adds-Forb= ([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mbs: Remaining:([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mbs: Checking constraint:[oxid_performance_con]
mbs: Remaining:([SC] [SN0] [SN1] [SN2] [SN] [ZR] [ZR0])
mbs: Checking constraint:[unit_cost_con]
mbs: Remaining:([SC] [SN0] [SN1] [SN] [ZR] [ZR0])
mbs: Checking constraint:[bendstrength_con]
mbs: Looking for best choice in:([SC])
mbs: Selecting Silicon Carbide [SC] for Material Parameter
main: Material Bend-Strength Selector overrides old value Silicon Carbide No-Oxid in Material Parameter with Silicon Carbide
mov: Changing state from waiting to ready
mwv: Changing state from waiting to ready
mce: Changing state from waiting to ready
mts: Current value Silicon Carbide for target is different from preferred choice Silicon Carbide No-Oxid, scheduling mts for conflict resolution.
mbs: Material Bend-Strength Selector was able to select his preferred value for Material Parameter
mbs: Finished cycle 12
mbs:
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 13 ****
agenda: Material Thermal-Conductivity Selector because of conflict, ranked at 100
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 13 ****
agenda: Running Material Thermal-Conductivity Selector because of conflict
mts: Starting cycle 13
mts: Additionals:()
mts: Forbiddens:()

mts: Alts+Adds-Forb= ([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Looking for best choice in:([SC0] [SC1] [SC2] [SC] [SN0] [SN1] [SN2] [SN] [AL] [ZR] [ZR0])
mts: Selecting Silicon Carbide No-Cost [SC0] for Material Parameter
main: Material Thermal-Conductivity Selector cannot store Silicon Carbide No-Cost in Material Parameter, because Material Bend-Strength Selector [MAIN::mts] had already stored Silicon Carbide.
mts: selection_selection conflict detected with Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mbs: Got asked by Material Thermal-Conductivity Selector for number of proposals, replying with:1
mbs: Sending message to Material Thermal-Conductivity Selector
mts: Adapting to the only proposal from Material Bend-Strength Selector
mts: Sending message to Material Bend-Strength Selector
mts: Finished cycle 13
mts:
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 14 ****
agenda: Material Cost Estimator because of usage, ranked at 60
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 14 ****
agenda: Running Material Cost Estimator because of usage
mce: Starting cycle 14
mce: Constraint:Constraint for valid density satisfied.
mce: Constraint:Constraint for valid unit cost satisfied.
mce: The mass of the part made from Silicon Carbide is 0.005 kg.
mce: The cost for the object made from Silicon Carbide is 0.01175 $ per part.
mce: The total material cost for Silicon Carbide is exactly $18.8 for a batch size of 2000.
mce: Finished cycle 14
mce:
mce: Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 15 ****
agenda: Material Wear-Performance Evaluator because of usage, ranked at 60
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at 60
agenda: **** Executing Cycle 15 ****
agenda: Running Material Wear-Performance Evaluator because of usage
mwv: Starting cycle 15

```
mwv:    Constraint:Constraint for valid wear_performance satisfied.
mwv:    Preference:Preference for high wear-performance evaluates to 0.5
mwv:    Finished cycle 15
mwv:
mwv:    Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 16 ****
agenda: Material Oxidation-Performance Evaluator because of usage, ranked at  60
agenda: **** Executing Cycle 16 ****
agenda: Running Material Oxidation-Performance Evaluator because of usage
mov:    Starting cycle 16
mov:    Constraint:Constraint for valid oxid_performance satisfied.
mov:    Preference:Preference for high oxid-performance evaluates to 0.5
mov:    Finished cycle 16
mov:
mov:    Changing state from ready to waiting
Press <return> to continue, or ‘q’ <return> to quit.

agenda: **** Schedule for Cycle 17 ****
agenda: Main Agenda empty
agenda: **** AGENDA EMPTY ****
agenda: Main Agenda finished
agenda: **** Finished ****
agenda: Running time was 0 seconds.
0
CLIPS>
```

Bibliography

Austin 1962 — J. L. Austin. *How to do Things with Words.* Harvard University Press and Cambridge, MA, and Clarendon Press, London, 1962. Reprinted in *Readings in the Philosophy of Language.* J. F. Rosenberg & C. Travis eds., Prentice-Hall Inc., New Jersey 1971

Agha & Hewitt 1985 — G. Agha & C. Hewitt. "Concurrent Programming using Actors, Exploiting Large Scale Parallelism" *Proceedings of the 5th Conference on Foundations of Software Technology and Theoretical Computer Science,* Springer Verlag, 1985. Reprinted in *Readings in Distributed Artificial Intelligence,* A. H. Bond and L. Gasser (eds.), Morgan Kaufmann Publishers, Inc. San Mateo, California, 1988

Bond & Gasser 1988 — A. H. Bond and L. Gasser (eds.) *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers, Inc. San Mateo, California, 1988

Brown 1992 a — D. C. Brown. “Design”. *Encyclopedia of AI*, 2nd edition, S. Shapiro, ed., J. Wiley, New York, NY, 1992

Brown 1992 b — D. C. Brown. *“The Reusability of DSPL Systems”. Preprints ot the Workshop on Reusable Design Systems: Understanding the Role of Knowledge.* Second International Conference on Artificial Intelligence in Design, Carnegie-Mellon University, Pittsburgh, USA

Brown 1994 — D. C. Brown. “Types of Theories and their Roles” in *AID-94 Workshop on Nature & Role of Theory in Design, 1994*

Brown & Chandrasekaran 1989 — D. C. Brown, B. Chandrasekaran. “Design Problem Solving: Knowledge Structures and Control Strategies”. *Research Notes in Artificial Intelligence Series*, Pitman Publishing, Ltd., London, England, May 1989

Bussmann & Mueller 1993 — S. Bussmann & J. Mueller. *Bargaining Agents.* Submitted to IJCAI, 1993

Cohen & Perrault 1979 — P. R. Cohen & C. R. Perrault. “Elements of a Plan-Based Theory of Speech Acts”, in *Cognitive Science 3*: pp. 177 - 212, 1979. Ablex Publishing Corp., 1985. Reprinted in *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser (eds.), Morgan Kaufmann Publishers, Inc. San Mateo, California, 1988

Connah et al. — Connah, Shiels & Wavish. *A Testbed for Research on Cooperating Agents*

Cromarty 1987 — A. S. Cromarty. “Control of Processes by Communication over Ports as a Paradigm for Distributed Knowledge-Based System Design” in *Expert Database Systems.* L. Kerschberg, ed. The Benjamin Cummings Publishing Co. Inc. 1987. Reprinted in *Readings in Distributed Artificial Intelligence,* A. H. Bond and L. Gasser (eds.), Morgan Kaufmann Publishers, Inc. San Mateo, California, 1988

Davis & Smith 1981 — R. Davis & R. G. Smith. “Negotiation as a Metaphor for Distributed Problem Solving” in *Artificial Intelligence 20*, pp. 63 - 100, 1983. North Holland. Reprinted in *Readings in Distributed Artificial Intelligence,* A. H. Bond and L. Gasser (eds.), Morgan Kaufmann Publishers, Inc. San Mateo, California, 1988

Douglas 1992 — R. E. Douglas Jr.. *SNEAKERS: A Concurrent Engineering Demonstration System.* Masters Thesis, Worcester Polytechnic Institute, 1992

Douglas et al. 1993 — R. E. Douglas, D. C. Brown, D. C. Zenger. “A Concurrent Engineering Demonstration & Training System for Engineers and Managers” in *Revue Internationale de CFAO et d’Infographie”* (International Journal of CAD/CAM and Computer Graphics) special issue on “AI and

Computer Graphics", (Ed.) I. Costea, Hermes, Vol.8, No.3, pp. 263-301.

Garvey et al. 1994 — A. Garvey, K. Decker & V. Lesser. *A Negotiation based Interface between a Real-time Scheduler and a Decision-Maker.* Technical Report, CS Dept. UMass Amherst. 1994

Giarratano & Riley 1993 — J. C. Giarratano & G. Riley. *CLIPS Reference Manual. Volumes 1 & 2.* NASA Lyndon B. Johnson Space Center Information Systems Directorate, Software Technology Branch, Version 6.0, June 2nd 1993

Giarratano & Riley 1994 — J. C. Giarratano & G. Riley. *Expert Systems: Principles and Programming,* 2nd ed.. PWS Publishing CO, Boston, MA.1994

Grosz & Sidner 1990 — B. J. Grosz & C. L. Sidner. "Plans for Discourse" in *Intensions & Communication.* Cohen, Morgan & Pollack (eds.) MIT Press, Cambridge, MA. 1990

Gruber 1993 — T. H. Gruber. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing.* Rev. August 23, 1993. Substantial revision of a paper presented at the International Workshop on Formal Ontology, March, 1993, Padova, Italy. To appear in a collection edited by Nicola Guarino. Available as Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University.

Finin et al. 1993 — T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzon, D. McKay, J. McGuire, R. Pelavin, S. Shapiro & C. Beck. *DRAFT Specification of the KQML Agent-Communication Language.* The DARPA Knowledge Sharing Initiative External Interfaces Working Group, 1993. More information can be received through ftp from ksl.stanford.edu

Genesereth & Fikes 1992 — M. R. Genesereth & R. E. Fikes. *Knowledge Interchange Format Version 3.0 Reference Manual.* Computer Science Dept., Stanford University, CA, 1992. More information can be received through ftp from ksl.stanford.edu

Huhns 1987 — M. N. Huhns (ed.). *Distributed Artificial Intelligence Vol. 1*, Pitman Publishing, London, and Morgan Kaufmann Publishers, San Mateo, CA, 1987

Huhns & Gasser 1989 — M. N. Huhns, L. Gasser (eds.). *Distributed Artificial Intelligence Vol. 2*, Pitman Publishing, London, and Morgan Kaufmann Publishers, San Mateo, CA, 1989

Khedro & Genesereth 1994 — T. Khedro & M. Genesereth. “Solution Consistency and Convergence in Cooperative Distributed Problem Solving”. *Proceedings of the AAAI ‘94 Conference*, 1994

Klein 1991 — M. Klein. “Supporting Conflict Resolution in Cooperative Design Systems” in *IEEE Transactions on Systems,*

Man, and Cybernetics, Vol. 21, No. 6, November/ December 1991

Klein & Lu 1990 — M. Klein & C.-Y. Lu. “Conflict Resolution in Cooperative Design” in *The International Journal for AI in Engineering*. No. 4, pgs. 168 - 180. 1990

Klein & Lu 1991 — M. Klein & C.-Y. Lu. “Insights into Cooperative Group Design: Experience with the LAN Designer System” in *Proceedings of the AIENG Conference.* 1991

Laasri et al. 1992 — B. Laasri, H. Laasri, S. Lander & V. Lesser. “A Generic Model for Intelligent Negotiating Agents” in *International Journal on Intelligent Cooperative Information Systems*, pgs. 291 - 317. World Scientific Publishing Company, 1992

Lander & Lesser 1991 — S. E. Lander & V. R. Lesser. “Customizing Distributed Search Among Agents with Heterogeneous Knowledge” in *Proceedings of the 5th International Symposium on AI Applications in Manufacturing & Robotics*, Cancun, Mexico. Dec 1991

Lochbaum et al. 1990 — K. E. Lochbaum, B. J. Grosz & C. L. Sidner. “Models of Plans to Support Communication” in *Proceedings of the AAAE ‘90 Conference*, Boston, MA, July 1990

Liu & Sycara 1993 — J. Liu & K. P. Sycara. “Emergent Constraint Satisfaction through Multi-Agent Coordinated Interaction”. *Proceedings of MAAMAW.* 1993

Kuokka et al. 1993 — D. R. Kuokka, J. G. McGuire, R. N. Pelavin, H. C. Weber. H. M Tenenbaum, T. Gruber and G. Olsen. “SHADE: Technology for Knowledge-Based Collaborative Engineering” in *AAAI Wkshp. on Collaborative Design 1993*, as well as *Concurrent Engineering: Research and Applications*, Vol 1, pp. 137 - 146, 1993

Searle 1965 — J. Searle. “What is a Speech Act” in *Philosophy in America.* Max Black, ed. George Allen & Unwin Ltd., London, 1965. Reprinted in *Readings in the Philosophy of Language.* J. F. Rosenberg & C. Travis (eds.), Prentice-Hall Inc., New Jersey 1971

Sidner 1992 — C. L. Sidner. *Using Discourse to Negotiate in Collaborative Activity: An Artificial Language*. Technical Report, DEC Cambridge Research Lab, Cambridge, MA, 1992

Smith 1980 — R. G. Smith. “The Contract Net Protocol: High-Level Communication and Control in Distributed Problem Solver” in Proceedings AAAI-86, pp. 51 - 57, 1986. Reprinted in *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser (eds.), Morgan Kaufmann Publishers, Inc. San Mateo, California, 1988

Sycara 1987 — K. P. Sycara. “Finding Creative Solutions in Adversarial Impasses” in *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA. July 1987

Sycara 1990 — K. P. Sycara. “Cooperative Negotiation in Concurrent Engineering Design” in *Cooperative Engineering Design*, Springer Verlag Publications, 1990

Taleb-Bendiab & Oh 1993 — A. Taleb-Bendiab & V. Oh. “Speech-Act based Communication Protocol to support Multi-Agent Cooperative Design Systems”. *Proceedings of the 1993 AI in Engineering Conference*, p. 107, Computational Mechanics Inc., 1993

Victor 1993 — S. K. Victor. *Negotiation Between Distributed Agents in a Concurrent Engineering System.* Masters Thesis, Worcester Polytechnic Institute, 1993

Victor et al. 1993 — S. K. Victor, D. C. Brown, J. J. Bausch, D. C. Zenger, R. Ludwig, R. D. Sisson. “Using Multiple Expert Systems with Distinct Roles in a Concurrent Engineering System for Powder Ceramic Components” in *International Conference on Artificial Engineering*, Tolouse, France, July 1993

Victor & Brown 1994 — S. K. Victor, D. C. Brown. “Designing with Negotiation using Single Function Agents”. *Applications of Artificial Intelligence in Engineering IX*, G. Rzevski, R. A.

Adey, D. W. Russell (eds.), Proc. AIENG'94, 9th Int. AI in Engineering Conf., Pennsylvania, USA. Computational Mechanics Publications, pp. 173-179.

Werkman & Barone 1991 — K. J. Werkman & M. Barone. "Evaluating Alternative Connection Designs Through Multiagent Neogiation" *Computer Aided Cooperative Product Development*, D. Sriram, R. Logcher, S. Fukuda (eds.), Lecture Notes Series, No. 492, pp. 298 - 333, Springer Verlag, 1992

Werner 1989 — E. Werner. "Cooperating Agents: A unified Theory of Communication and Social Structure" in *Distributed Artificial Intelligence Vol. 2*, M. N. Huhns, L. Gasser (eds.) Pitman Publishing, London, and Morgan Kaufmann Publishers, San Mateo, CA, 1989

Wong 1992 — S. T. C. Wong. *Cooperative Decision Making based on Preferences.* Technical Report, Institute for New Generation Computer Technology, Tokyo, Japan 1993, also accepted for publication in ACM Transactions on Information Systems, 1994

Wong 1993 — S. T. C. Wong. *Coping with Conflict in Cooperative Information Systems* Technical Report, Institute for New Generation Computer Technology, Tokyo, Japan 1993