Conflicts and Negotiations in Single Function Agent Based Design Systems

by
Ilan Berker

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

May 1995

APPROVED:

David C. Brown, Major Advisor

Robert E. Kinicki, Head of Department

Abstract

Design is a very complicated and ill-defined problem solving activity. Routine parametric design is a more restricted and well-defined version of design problems. Even this restricted version requires many different kinds of expert knowledge and the ability to perform a variety of tasks. One approach to solving this restricted version is to use Single Function Agents (SiFAs), each of which can perform a very specialized task, from a single point of view. The ability to represent expertise with different points of view is very important in design. These different points of view usually cause conflicts among agents, and these conflicts need to be resolved in order for the design process to be successful. Therefore, agents need to be capable of detecting and resolving these conflicts. This thesis presents a model of conflicts and negotiations in the SiFA framework. Some extensions to the present state of the SiFA paradigm are introduced. A hierarchy of possible conflicts is proposed and the steps of the negotiation process are discussed. The ability of agents to negotiation in order to resolve conflicts makes SiFA-based design systems more versatile, less brittle, and easier to construct and maintain. Also, the extended SiFA paradigm, where agents have negotiation capabilities leads to many interesting directions for further research.

Acknowledgements

I would like to thank my advisor Prof. David C. Brown for his valuable ideas and suggestions that helped shape this work, my reader Prof. Lee A. Becker for correcting many errors present in the earlier drafts of the thesis, and all the members of the Artificial Intelligence in Design Group for their comments and contributions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Design is a very important activity and requires formal study. The importance of design activity has been recognized since approximately 2000 B.C., when Hammurabi, the king of Babylon enacted a law about the dangers of designing houses. The law said:

> If a designer-builder has designed-built a house for a man and his work is not good, and if the house he has designed-built falls in and kills the householder, that designer-builder shall be slain.

The original of Hammurabi's Code is on a stela in Cuneiform in the Louvre, Paris.

Design in general is a very complicated task and it is very hard to model all design activity in a single computational framework. Therefore this work focuses on a restricted set of design problems.

This chapter explains the set of problems that this work deals with, introduces multi-agent systems and single function agents, and then describes the main focus of our work, which is conflicts and negotiation.

1.1 Design Problems

The space of all design problems can be mapped on a plane where one axis extends from routine to non-routine problems and the other from parametric to conceptual as in figure 1.1.

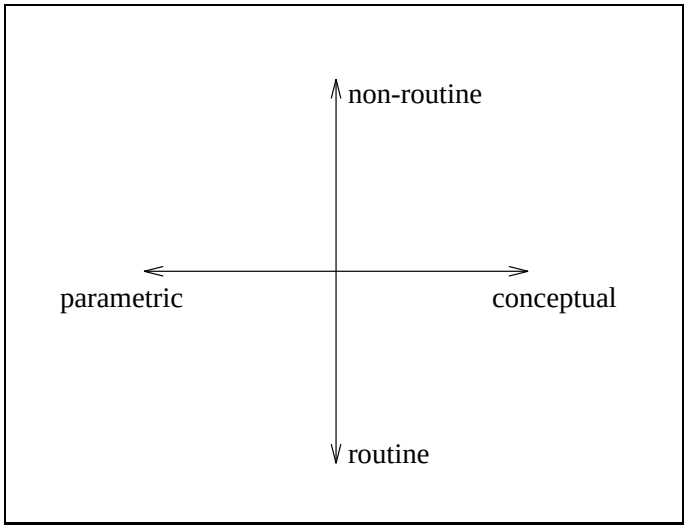

Figure 1.1: The space of all design problems

These axis are continuous. There are no clear cut lines that differentiate problems into routine or non-routine, and parametric or conceptual. Still, the problems this work deals with can be loosely characterized as the ones in the lower left quadrant of this plane, routine parametric design problems.

Routineness of a design problem suggests that the knowledge and processes required to solve the problem are known and well understood [Brown & Chandrasekaran 92]. Being parametric means that the structure of the design and all parameters of it are known in advance, so that the design process has to deal with assigning a set of values to these parameters. The values should satisfy a set of constraints on the parameters. These constraints may arise from physical laws, some preferences, and user requirements.

1.2 Multi-agent Design Systems

As design is a very complicated task, it needs to be decomposed into sub-tasks each of which requires different expertise. This specialized expertise is best represented as an agent. This way of breaking up an expert system, functionally decomposes the complicated task that the system performs into less complicated, autonomous pieces we call agents.

The multi-agent paradigm is very modular and therefore supports ease of construction and maintainability.

Another advantage of having multiple agents is the support for concurrent engineering. Each agent can have a different point of view and can therefore contribute to the design process by taking into account downstream effects of design decisions.

Some attributes of agents in general are *veracity*, the assumption that an agent will not knowingly communicate false information, *benevolence*, the assumption that agents do not have conflicting goals and that every agent will always try to do what is asked of it, and *rationality*, the assumption that an agent will act in order to achieve its goals [Goodwin 93].

The agents in a cooperative design system may be characterized by veracity and rationality but benevolence does not apply. It is very important that agents be allowed to have conflicting goals and this is exactly what makes design a complicated problem.

1.3 Single Function Agents

Taking the multi-agent paradigm to an extreme results in Single Function Agents (SiFAs) where each agent performs a single function in the design process and therefore contains knowledge that is as specialized as possible.

Such a view of agents enables the study of the building blocks of what makes up a design system in terms of the knowledge used and the functionality that is required.

The single function paradigm is explained in more detail in chapter 3.

1.4 Conflicts and Negotiation

The use of single function agents, or any other form of agents in a system brings with it the possibility of conflicts between agents. Although it may be possible to build a system where all conflicts are resolved during development time, in most design problems this is either impossible or extremely difficult.

In most cases, development time conflict resolution is not even desirable since there are many advantages of run time conflict resolution. The run time interaction between the agents supports concurrent engineering. Negotiation also causes the behavior of the system to emerge from the necessities of the particular problem at hand. So the system is more flexible and does not suffer from the brittleness problem of traditional expert systems. Run time conflict resolution requires the conflict resolution knowledge to be treated as first class knowledge in the agents just like domain knowledge and control knowledge [Klein 91]. So the separation and explicit representation of different knowledge types is enforced.

One way of resolving run time conflicts is for the agents to negotiate. This is the most general way of resolving conflicts. The negotiation process is very flexible and encompasses all other conflict resolution methods. A method where conflicts are resolved by some table lookup mechanism as in the I3D+ system [Victor & Brown 94], where the outcome of every conflict is stored may be seen as a restricted case of negotiation.

Therefore studying negotiation promises an overall understanding of conflict resolution.

1.5 COSINE: A Wine Glass Designer

One goal of this work was to build a single function agent based design system to demonstrate the the capabilities and restrictions of negotiation in the SiFA paradigm. So a wine glass designer called COSINE was implemented.

This work is a continuation of previous work on SiFAs that developed a SiFA platform called SINE (SiFA Negotiations), and therefore was named COSINE.

The wine glass domain is simple enough to enable building a prototype in a short amount of time. Since wine glass design does not involve complicated technical domain knowledge, expert knowledge is readily available. The simplicity of the domain also makes it attractive for demonstration purposes.

This domain is also complicated enough to demonstrate all the power of a SiFA system with negotiation capabilities and the richness of the possible conflict situations.

The parameters of the design, as shown in figure 1.2, are cup radius, cup thickness, stem length, stem radius, base radius, and base thickness.

There are many constraints among these parameters. For example, as the ratio between the cup radius and the base radius has to be within certain limits so that the cup is stable. Also, the cup radius has to be within certain limits so that it holds a reasonable amount of liquid. Similar constraint involving other parameters exits.

The rest of this thesis is organized as follows. Chapter 2 is about previous work in this field; chapter 3 gives details of the SiFA paradigm; chapter 4 explains the extensions to the earlier SiFA paradigm; chapter 5 and chapter 6 discuss conflicts and negotiations; chapter 7 contains examples of sample runs from COSINE to demonstrate the ideas presented in this work; chapter 8 gives details of the implementation of COSINE; chapter 9 is an evaluation of this work; chapter 10 presents further research issues; and the document ends with a summary and conclusions in chapter 11. The appendices contain sample rules from COSINE and a sample run.

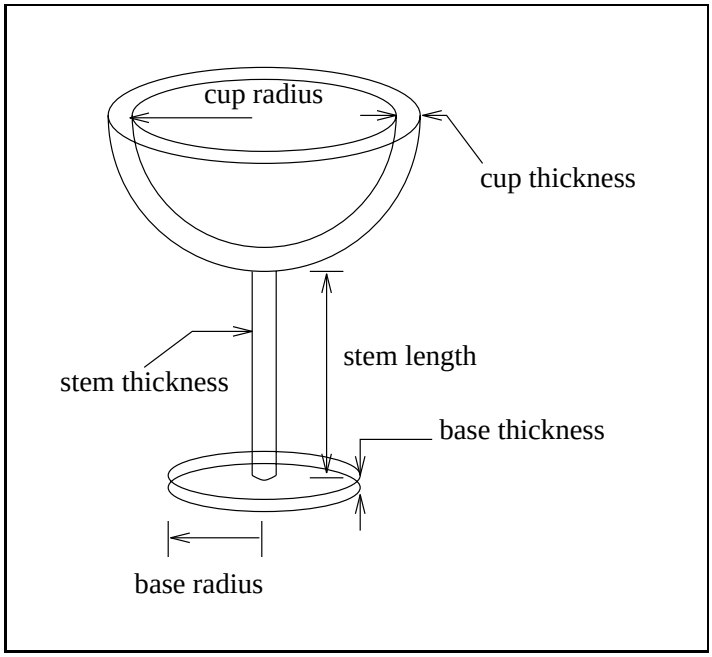

Figure 1.2: Wine glass parameters

Chapter 2

Previous Work

This chapter contains a short review of previous work related to this thesis including work on agents, multi-agent systems, conflict resolution, negotiation, finally other SiFA systems built at WPI. A full description of the SiFA paradigm which is essential in order to understand the work presented in this thesis, is presented in chapter 3.

2.1 Agents and Multi-Agent Systems

In recent years there has been an increased interest in agent based systems. The concept of an agent has become important both in Artificial Intelligence (AI) and in other fields of computer science.

Wooldridge & Jennings [1994] define agents to be software-based computer systems that show the following properties:

- *autonomy:* agents have some control over their actions and internal states,
- *social ability:* agents interact with other agents using an agent communication language,
- *reactivity:* agents can perceive their environment and can respond to changes,
- *pro-activeness:* agents don't act only in reaction to their environments but can

also show goal directed behavior by taking initiative.

In AI terms, an agent has more human like properties such as knowledge, beliefs, intentions, and obligations, in addition to the above mentioned properties [Shoham 90]. Discussion of other attributes of agents and attempts to define an agent can be found in [Goodwin 93].

Communication in agent theory is usually based on speech act theory which was originated by Austin [1962], and further developed by Searle [1969].

Agents also have a direct influence on Distributed Artificial Intelligence (DAI) research because of their concurrent nature [Bond & Gasser 88] [Huhns 87].

Many difficult tasks, including design, have parallel decompositions that result in easier subtasks than serial decompositions. Parallel decompositions allow opportunistic collaboration among the subtasks [Talukdar & Cardozo 88].

2.2 Conflict Resolution

Klein [1991] suggests a model of conflict resolution having a hierarchy of conflicts with the most abstract conflicts at the top and most concrete conflicts at the leaves. There is a corresponding hierarchy of resolution strategies. This structure is shown in figure 2.1. Klein's hierarchy has domain dependent and domain independent conflicts and their associated resolution strategies. The nodes higher in the hierarchy represent the domain independent conflicts while the lower nodes become more domain dependent. The conflict hierarchy first differentiates all conflicts into two broad categories. The first is two agents giving incompatible specifications for a design component. The second category is one agent criticizing specifications asserted by another agent.

Klein's model of agents, as shown in figure 2.2 all have their own design knowledge and conflict resolution knowledge. The agents have differing design expertise,

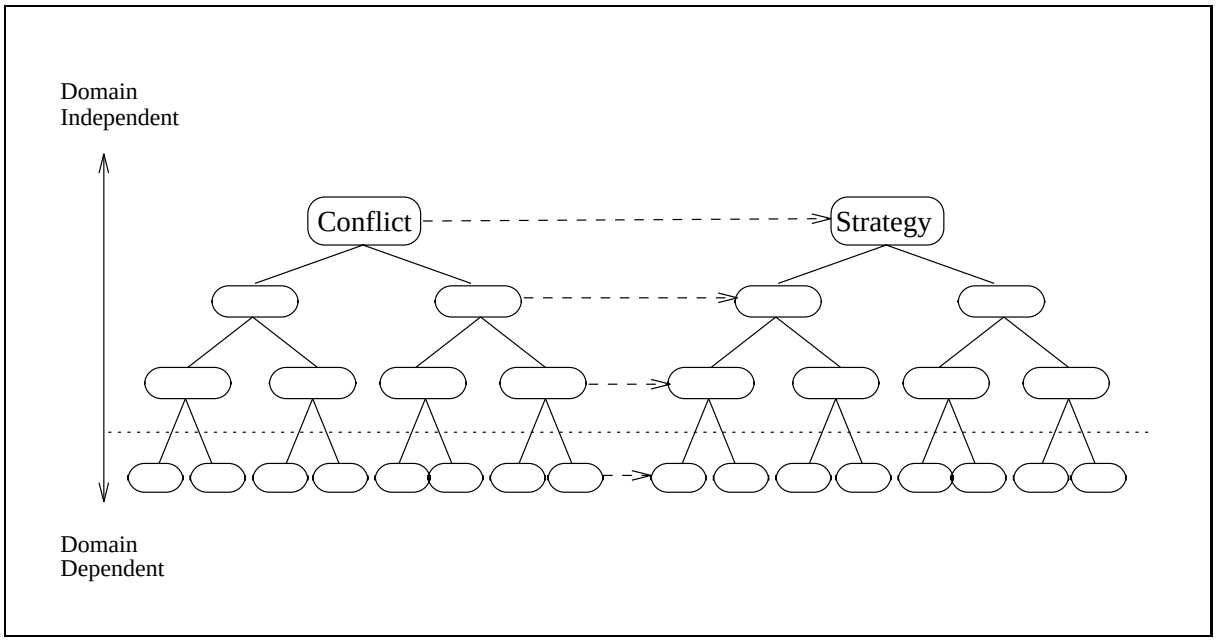

Figure 2.1: Hierarchy of conflicts and strategies

but their conflict resolution knowledge is identical, consisting of the two hierarchies mentioned above.

The operation of the conflict resolution components of the agents in the Klein model, shown in figure 2.3, are as follows. Conflicts that occur are matched to the most specific conflict type on the hierarchy. To able able to do this matching, agents need to gather information about the current conflict situation. Agents get this information by sending queries to each other using a query language. Then the corresponding strategy on the hierarchy of resolution strategies is used to solve the conflict, probably after some specialization which again involves queries among agents. If this strategy fails, the strategy represented by the parent node of the current strategy is used. Since this strategy is more general, it is applicable to a wider class of conflicts. The process ends either when the conflict is resolved or when the strategy represented by the root node is reached and there is not an agreement between the conflicting agents, in which case there is a failure. An example of a general strategy is:

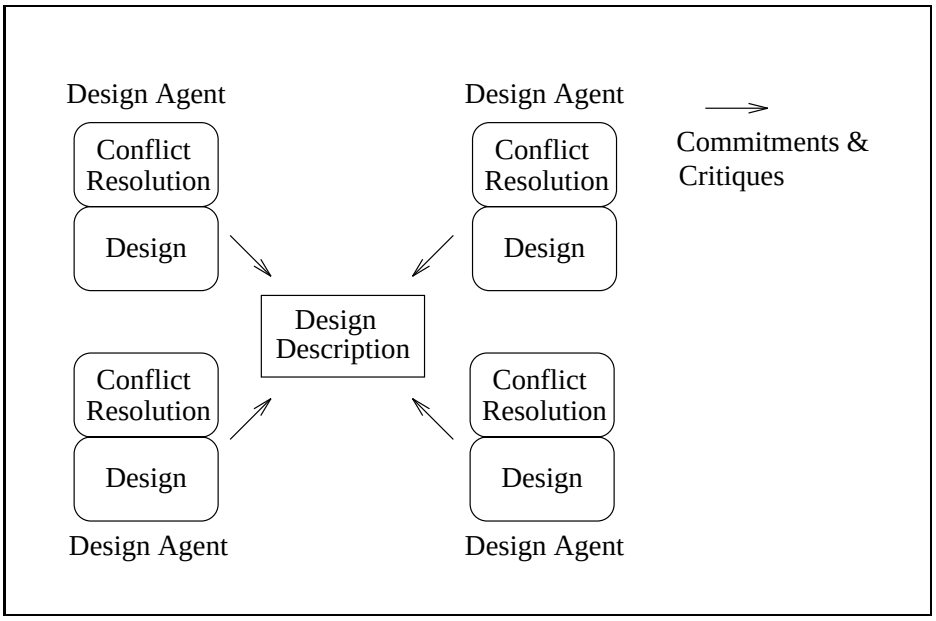

Figure 2.2: Components of agents

If two plans for achieving two different agents' goals conflict then find an alternate way of achieving one goal that does not conflict with the other agent's plan for achieving its goal [Klein & Lu 90] [Klein & Lu 91].

The SINE work has taken a small step in the direction of this model. Work presented in this thesis has many more similarities to Klein's work as it defines a similar conflict hierarchy. The main differences are that the conflict hierarchy presented in chapter 5 does not have any domain dependent nodes, and the agents, as explained in section 4.3 all have their own unique conflict resolution knowledge.

Another interesting approach to conflict resolution is Sycara's Situation Assessment Packages (SAPs) [Sycara 87]. SAPs are information structures that contain, among other things, a description of a problem solving situation, expectations, reasons for expectation violations, and warnings for failures.

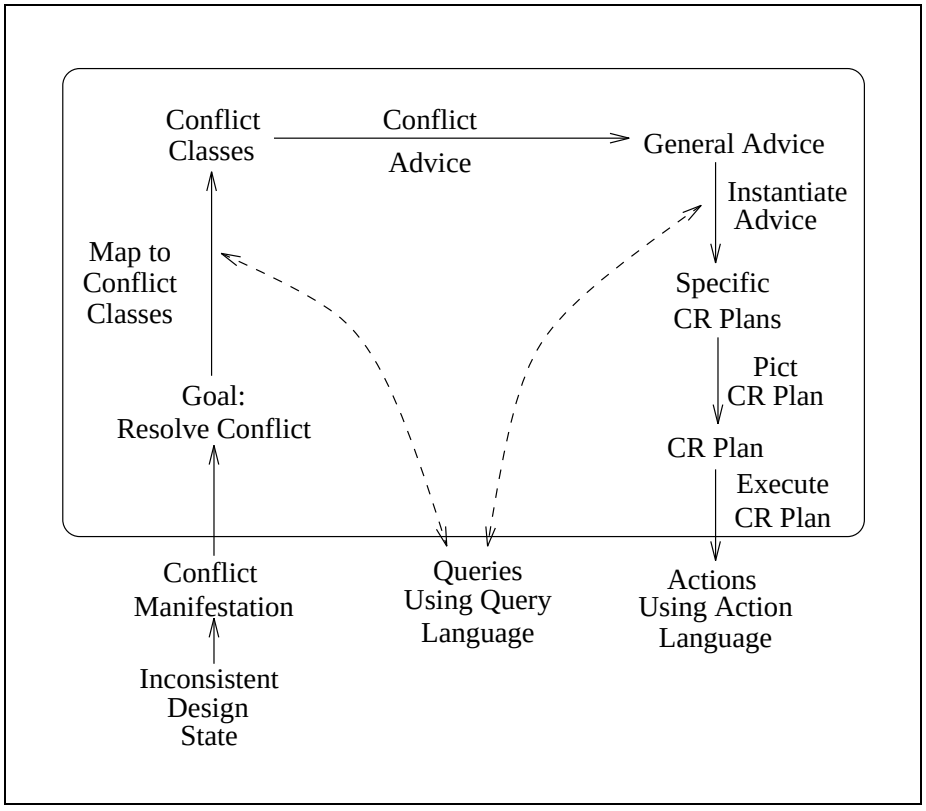

Figure 2.3: Operation of conflict resolution component

2.3 Negotiation

Negotiation is a prevalent approach to conflict resolution in the design domain.

Sycara defines negotiation to be the process by which resolution of inconsistencies is achieved in order to arrive at a coherent set of design decisions [Sycara 90]. She also argues that negotiation has to be iterative rather than a one shot process, and this complicated process is not amenable to traditional AI techniques [Sycara 88].

In Sycara's model there are four conflict situations where negotiation is used in design. These conflicts are:

- Different agents make conflicting recommendations for a parameter value;
- A value proposed by one agent makes it impossible for another agent to offer consistent values for other attributes;
- A decision of one agent adversely affects optimality of other agents;
- Alternate approaches achieve similar functional results.

The negotiation process according to Sycara [1990] proceeds as follows:

1. Generation of proposal
2. Generation of counter proposal based on feedback from dissenting agents
3. Communication of justifications and supporting evidence

Sycara's agents have beliefs, intentions, and goals, and they use Case Based Reasoning (CBR) extensively during negotiation for tasks such as plan generation, plan evaluation, plan modification, and argumentation generation.

Lander and Lesser [1991] use the negotiated search paradigm for conflict resolution among heterogeneous and reusable expert agents in their TEAM framework. This

paradigm allows agents to be both logically and implementationally heterogeneous. They also examine a limited but very efficient negotiated search strategy called linear compromise. Although SiFAs are different from the agents in Lander's model, some of the negotiation ideas apply.

Werkman's Designer Fabricator Interpreter (DFI) is a system where agents with different points of view cooperatively evaluate different suggestions for a design parameter [Werkman & Barone 91]. Unlike SiFA systems, DFI uses an arbitrator as a means of central control through which agents communicate. There is no direct interaction between agents.

Polat et al. [1993] describe a problem-solving environment that supports multi-agent conflict detection and resolution. They include a flowchart that describes the general conflict resolution process between agents. This has many similarities with our approach, discussed in section 6.6.

2.4 Single Function Agents

Although there has been a lot of work done on multi-agent systems, Single Function Agents (SiFAs) are a relatively new way of building multi-agent systems.

Three systems have been developed at WPI using SiFAs. The first one of these, I3D [Victor et al. 93], was a system that integrates part design and manufacturing plan production for Powder Processing Applications. It used cooperative expert agents that assisted a human designer in the design of simple powder ceramic components by offering cost estimation, material selection, process simulation, and inspection planning. The agents used in I3D were:

Advisor: Provides information about what parameter's value to decide next;

Critic: Produces criticisms of design decisions;

Planner: Provides a choice of ordered actions;

Selector: Picks a value from a list;

Estimator: Produces estimates of values.

The agents were carefully sequenced, and all possible conflicts that might occur were anticipated in advance and removed during development of the system.

The second system built was I3D+ [Victor & Brown 94]. It had agenda-based scheduling of the agents and also allowed conflicts about the values of parameters to occur among agents. The conflicts were classified into six types depending on the relation between agents' local goals and the global goal as shown in table 2.1. The agents used simple negotiation schemes depending on the goal specified in the requirements.

Table 2.1: Conflicts in I3D+

Type	Agent1 Goal	Agent2 Goal	Global Goal	Result
1	X	Y	Z	Either one wins
2	X	Y	X	Agent1 wins
3	X	Y	Y	Agent2 wins
4	X	Y	X,Y	Either one wins
5	X	X	Z	Either one wins
6	X	X	X	Either one wins

SNEAKERS [Douglas et al. 93] was built to train users in Concurrent Engineering. The user interacted with agents that had different functions and points of view. SNEAKERS did not have selectors, estimators, or planners, but in addition to advisors and critics, it had the following types of agents:

Analyst: Performs numerical analysis to derive attributes;

Evaluator: Evaluates the whole design from a certain perspective;

Suggestor: Offers suggestions for satisfying criticisms.

The points of view that SNEAKERS had were design, manufacturing, assembly, cost, packaging, marketing, safety, and disposal,

After these systems, SINE was developed as a platform to build multi-agent design systems using SiFAs [Brown et al. 94]. It was possible to simulate the negotiation behavior of I3D+ using SINE. The work on SINE formalizes the SiFA model by classifying SiFAs as points on a three dimensional matrix with axes as labeled function, point of view, and target. Agents of a design system are classified according to their functions as estimator, evaluator, selector, advisor, critic, praiser, and suggestor. SINE proposes a communication language between agents, classifies some of the possible conflicts and describes simple communication patterns for negotiation. The SiFA paradigm and the SINE work is discussed in detail in chapter 3.

2.5 Summary

This chapter described some of the earlier work done in the areas of agents, conflicts, conflict resolution, negotiation, including the work done at WPI with SiFAs.

The next chapter will discuss the SiFA paradigm in detail. An understanding of the SiFA paradigm is essential to understanding the work presented in this thesis.

Chapter 3

The Single Function Agent Paradigm

This chapter explains, in detail, the Single Function Agent (SiFA) paradigm as it was defined in [Dunskus 94]. Information about the implementation of a platform called SINE that supports SiFAs can also be found in [Dunskus 94]. Extensions to the SiFA paradigm that are part of the work done for this thesis are explained in chapter 4.

3.1 What is a SiFA?

A SiFA is an agent that performs a single *function* on a single *target* from a single *point of view*. These terms are explained in the following paragraphs.

The function performed by a SiFA determines its type. There are only a limited number of functions needed for design problems. We conjecture that a set of agents with these functions is sufficient for most design problem solving activities. These agents types are:

Selector: Selects a value for a parameter by picking a value from a list of possible values. These are usually suitable for discrete valued parameters.

Advisor: Produces a value for a parameter by some means other than picking a value

from a list. Advisors are more suitable for real valued parameters. Advisors and selectors are the agents where most of the design knowledge is stored.

Estimator: Produces estimates of values for a parameter. Unlike selectors, estimators can work with insufficient information, so the values they produce are just estimates of what the final value should be.

Evaluator: Evaluate the value of a parameter, producing a value of goodness for that value, usually represented as a percentage.

Critic: Criticizes values of parameters by pointing out constraints or quality requirements that are not met by the current value.

Praiser: Praises values of parameters by pointing out why the value is desirable.

Suggestor: Suggests what to do to remove an existing conflict or to avoid a conflict altogether.

The target of a SiFA is a single parameter of the design. In the case of wine glass design, the targets may be cup radius, stem length, and the other parameters. It is very important that each agent has a single parameter. A selector can only select values for a single parameter, a critic can criticize only one parameter's value, and similarly for other types of SiFAs.

The point of view of an agent is some aspect of the design that the agent considers while doing its work. Usually, the point of view of the agent is a goal that the agent is trying to optimize. Examples of points of view for agents in wine glass design are cost (as in all design, the cost should be minimized), style (the glass should look nice), stability (the cup should not fall over because of a very large cup and a small base), volume (the cup should hold a reasonable amount of liquid).

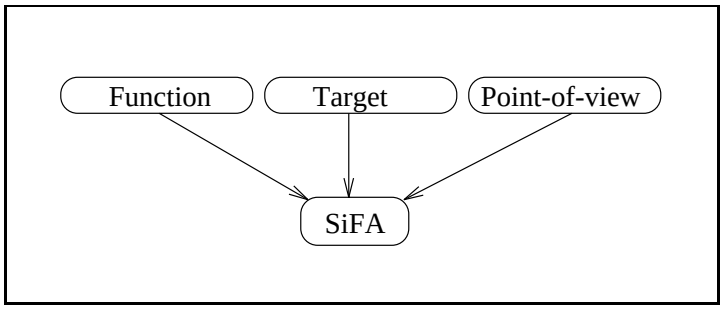

Figure 3.1: The components making up a SiFA

In the SiFA paradigm, an agent is characterized by a single function, target, and point of view as in figure 3.1. A naming convention for individual SiFAs is to specify the target first, then the point of view and lastly, the function. So a selector whose target is the cup radius and is trying to maximize stability of the cup is a *cup radius stability selector*.

There are many things that are common to agents with the same function, target, or point of view. Defining an agent involves inheriting all of this common information and then adding the knowledge that is specific to this particular agent. So a cup radius selector would inherit from a generic selector class, cup radius class and stability class in an object oriented manner. There can be class hierarchy with a class for each function, target and point of view. The class for the functions includes the basic functionality that all agents of that type have to have. The class for a target includes all operations that are meaningful on that target. The class for a point of view may include the basic knowledge required to optimize the design from that point of view.

As each agent has these three aspects, agents can be viewed as points on a three dimensional space where the axes are labeled with these three aspects. Figure 3.2 shows the space of agents for the wine glass design. There is no ordering between the values on the axes.

In any particular design problem, there can be at most one agent for any point in

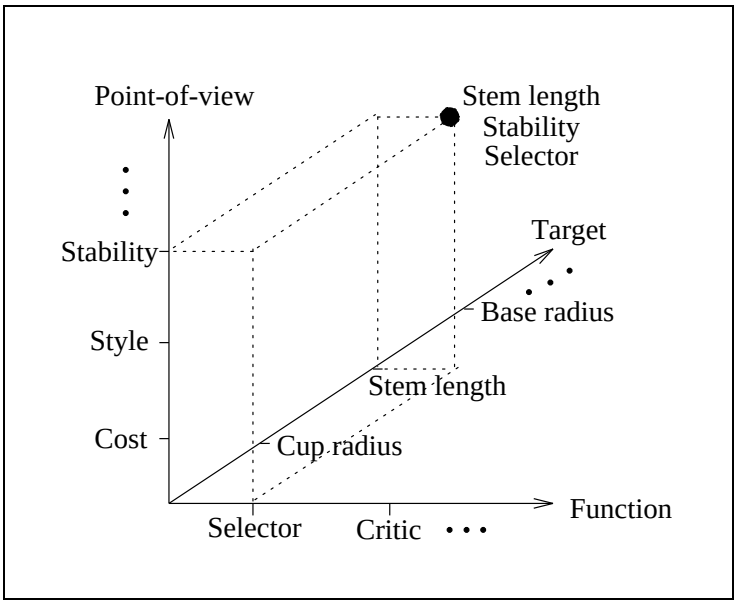

Figure 3.2: The space of agents

the space. More than one agent would mean that there are two identical agents in the system. Agents corresponding to some possible combinations of function, target, and point of view will not exist either because they are not meaningful, or because the expertise to build them does not exist.

3.2 Communication Between SiFAs

SiFAs communicate by sending each other messages. Each agent can communicate directly with any other agent without the need for an arbitrator. The communication language used is based on KQML (Knowledge Query and Manipulation Language) [Finin et al.].

The current state of the design resides on a blackboard structure which is accessible to all agents. Even thought agents exchange messages directly, design data flows thorough the blackboard. This is shown in figure 3.3.

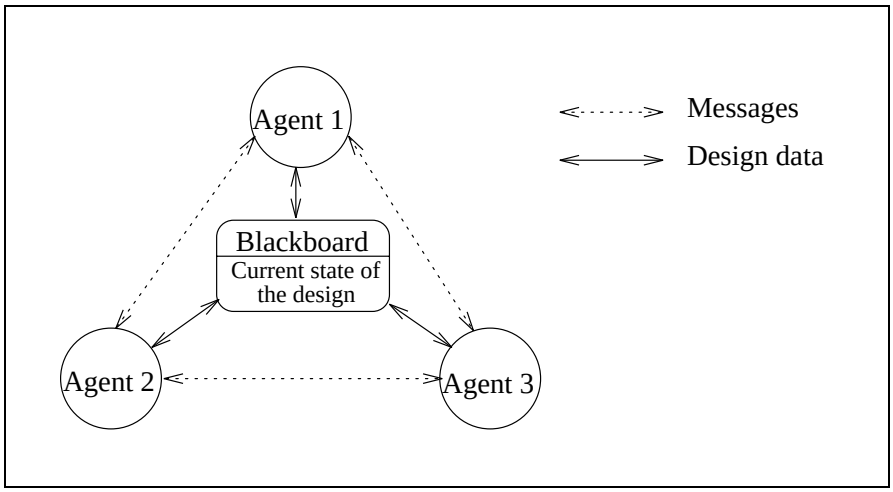

Figure 3.3: Communication and data flow

3.3 The Control Mechanism

An agenda mechanism is used to schedule agents. Agents are polled by the agenda and each agent replies with what it is that the agent wants to do. Based on the replies, the agenda orders the agents according to the importance of the tasks they will carry out, i.e., a conflict detection is more important than doing design. Then the agenda gives control to that agent. When the agent's work is finished, the cycle is repeated.

3.4 Classification of Conflicts

As in most multi-agent systems, conflicts are possible between SiFAs. A conflict is any situation when two agents disagree. The disagreement can be about the value of a design parameter or any other piece of information generated during the design, such as a criticism or an evaluation. The agent that detects a conflict is called the *initiator* of the conflict.

The conflicts between SiFAs are classified based on the pair of agent types (func-

tions) that are involved in the conflict.

Each possible conflict may be represented in a matrix shown in table 3.1. The agents that label the rows are the initiators of the conflict. The entries that are marked are the currently known meaningful conflicts. Some conflicts, such as a conflict between an estimator and an evaluator, initiated by the evaluator, are not meaningful. Some of the entries however are not marked because they have not been investigated yet.

Table 3.1: Matrix of conflict types

	Selector	Estimator	Evaluator	Critic	Praiser	Suggestor
Selector	√√	√	√			√
Estimator	√	√				
Evaluator	√√	√√	√			
Critic	√	√	√	√	√	
Praiser	√			√	√	√
Suggestor	√	√	√	√	√	√

Certain entries may have more than one kind of conflict associated with them, i.e., there are two different possible conflicts that an evaluator can initiate with a selector. The number of possible kinds of known conflicts in any entry are designated by the number of √'s.

3.5 Summary

The SiFA paradigm, as it had been before this thesis, was described in this chapter. SiFA's were defined and some aspects such as control and communication were discussed.

From this point on, the material presented describes the work done as part of this thesis. The next chapter explains the extension made to the SiFA paradigm.

Chapter 4

Extensions to the SiFA Paradigm

This chapter explains the extensions made to the SiFA paradigm. These extensions were necessary to be able to isolate different kinds of knowledge encompassed by a SiFA system, study all possible kinds of conflicts between agents, and study the resulting interactions and negotiations between SiFAs.

The following sections describe each of the required extensions to the paradigm.

4.1 The Parameter Block

In previous work with SiFAs, there was just one value associated with each parameter and each agent had a parameter as its target. Selectors or advisors provided the values for the parameters. Critics, praisers and evaluators worked with these values but the critiques, praises, or evaluations they produced were not stored separately. This is not very desirable if we wish to separate out all information.

The first extension to the SiFA model is to represent all value, estimate, criticism, praise, and evaluation *entities* separately, as *first class objects*. Being a first class object means that critiques, praises, and evaluations have the same status as the value of a parameter. They are all directly accessible and they can all be the target of an agent. This is done by having value, estimate, criticism, praise, and evaluation entities organized in a structure called the parameter block which is shown in figure 4.1.

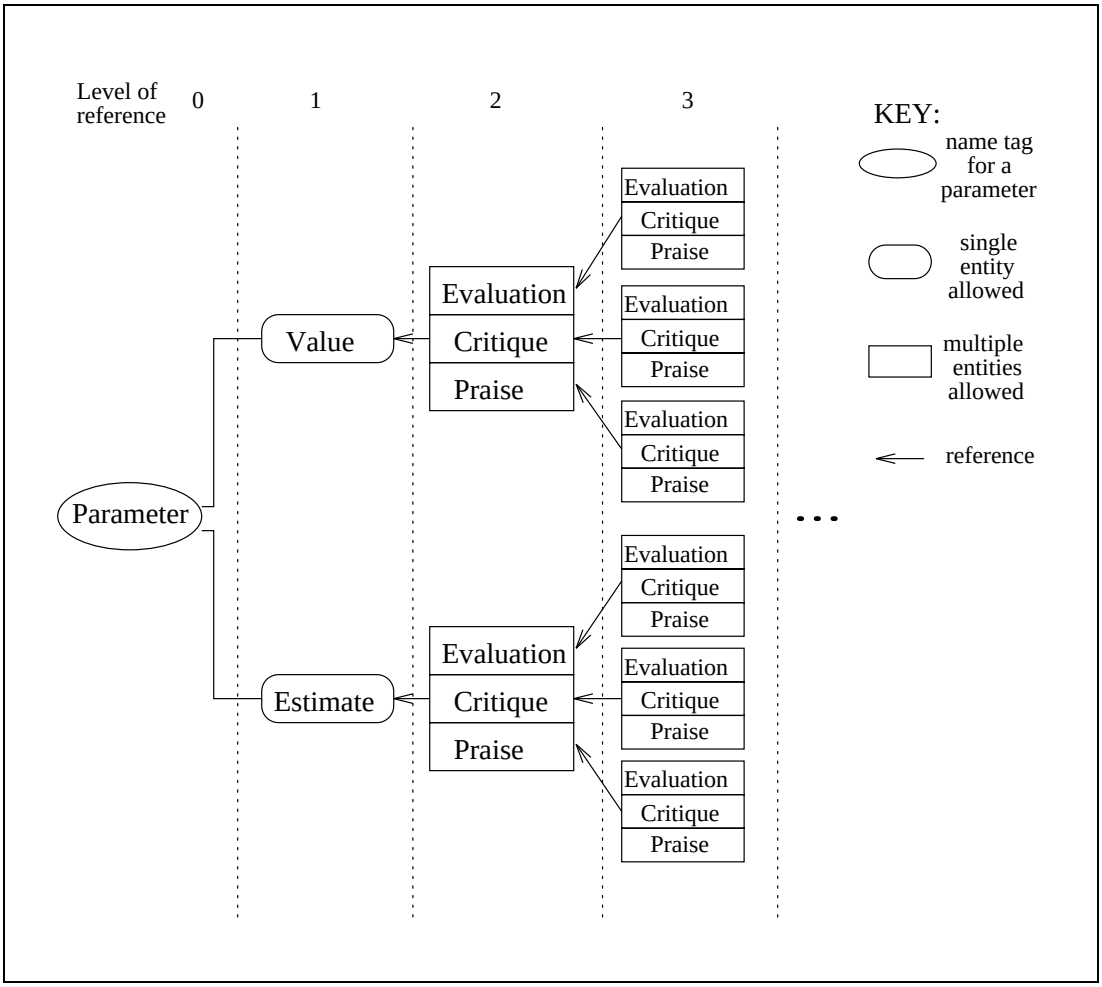

Figure 4.1: The parameter block

The root of the parameter block is just the name of a parameter. The first *level of reference* has two entities. These are the value and the estimate of the parameter. There can be only one value and one estimate at any time.

The second level of reference has evaluations, criticisms, and praises of the value and the estimate. These entities are said to reference the value or the estimate of the value. There can be multiple criticisms, praises, and evaluations of the same value or estimate.

The third level of reference has evaluations, criticisms, and praises of second level evaluations, criticisms, and praises. These entities refer to the first level entities which refer to the first level entities. These *chains of reference* uniquely determine what each entity refers to. So at the third level, there exist entities such as the evaluation of the criticism of the value of the parameter. The third level entities typically contain meta-level information about the design, i.e., the evaluation of a criticism is not directly about the object being designed, but has an extra level of reference. Once again, there can be multiple evaluations, criticisms, and praises of the same second level entity.

It is also possible to imagine this structure growing into fourth, even fifth level references. Although what would be the contents of such entities might not be immediately obvious, the model allows such information to be represented if it is meaningful in any design problem and the experts of the domain wish to describe it.

These entities are stored on the central blackboard in the same way as the values of the design parameters that make up the current state of the design. They are accessible to all agents if they wish to use them.

4.2 Targets of Agents

When all entities are represented as first class objects, they can all be the target of an agent. This means that the target of a critic is no longer just the name of a parameter, but more specifically, it is a criticism entity, such as the criticism of the evaluation of the estimate. Each agent can acts on, or modify, the contents of its target. So a critic can store a criticism in the critique entity which is its target, an evaluator can store an evaluation in the evaluation entity which is its target. Similarly, the target of an estimator is an estimate entity, the target of a praiser is a praise entity and the target of a selector is a value entity. This idea is presented in figure 4.2.

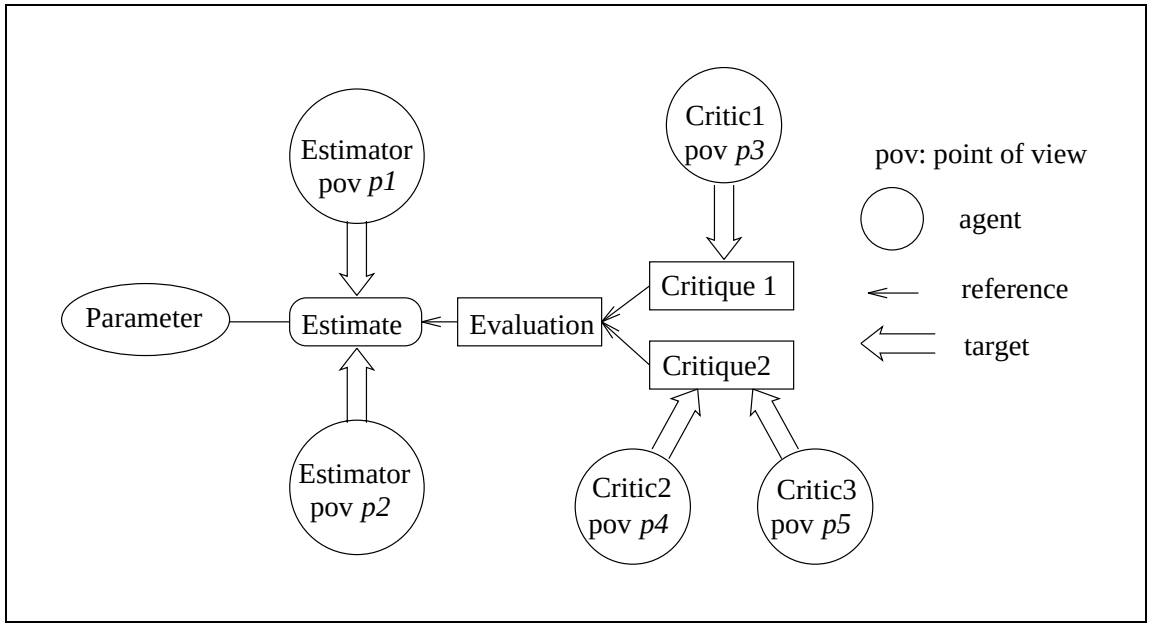

Figure 4.2: Two estimate evaluation critics

There can be more than one criticism of the evaluation entity, from different points of view, in which case there would be one agent for each of these criticisms, shown as Critic1 and Critic2 in figure 4.2. These critic agents would have different points of view and different targets, as they are acting on two different entities. It is also

possible for two critics to have different points of view but the same target as is the case with Critic2 and Critic3, if it is not desirable to store both criticisms produced by the two critics as separate entities.

Only one value and one estimate are possible at any time in a parameter block, unlike evaluations, criticisms, and praises, so there may be more than one selector or estimator agent, with different points of view, that have the same estimate or value as their target. This is also shown in figure 4.2.

4.3 Knowledge in a SiFA

The knowledge that has to be present in a SiFA is determined by the tasks the agent has to carry out (shown in figure 4.3). Each agent has to have knowledge about how to perform these tasks.

Figure 4.3: The knowledge contained in a SiFA

The types of knowledge can be roughly divided into three categories. The first one contains design and redesign knowledge. The design knowledge allows the agent

to carry out its main function which may be one of selection, estimation, evaluation, criticism or praise. The redesign knowledge is used when the agent has to produce a value, estimate, evaluation, criticism or praise after it has already produced it once. This is necessary when the first entity produced causes a conflict in the system and the agent is asked for another one during negotiation. This situation is explained in chapter 6.

The second category contains conflict indication, detection, and classification knowledge that is used to carry out the respective tasks of conflict handling. These typically involve checking if some constraints which are internal to the agent, referred to as *internal constraints* are violated or not. These tasks are explained in detail in chapter 6. The knowledge in this category also includes the knowledge of the types of conflicts in which the particular agent can be involved.

The third category consists of negotiation strategy selection, refinement, and execution knowledge. This category of knowledge enables the agent to negotiate with other agents in conflict situations. Negotiation strategies and the associated tasks are also explained in chapter 6.

4.4 Selectors/Advisors

In the SINE work, there were two different types of agents capable of producing values. These were selectors and advisors. While selectors chose a value from a list of possible alternatives, advisors applied some arbitrary function to its inputs to produce a value.

The distinction between selectors and advisors was not based on *what* they do but *how* they do it. Since the main criteria in dividing agents into types is their functionality in the system, we decided to merge selectors and advisors into a single agent type and call it a *selector*. The new selectors are capable of choosing values

from a list of alternatives, can use arbitrary functions to produce values, or utilize a combination of the two processes.

The combination is useful if the selector first chooses a value from a list and then applies a function to it, or if it applies a function to inputs but uses values from a list to instantiate some variables in the function. An example for the latter case is when a cup radius selector chooses a coefficient from a list to multiply with the value of the base radius in order to produce the value of the cup radius in the wine glass design domain.

4.5 User Requirements

In the SINE platform, the user could specify the values of some parameters or some attributes of a parameter of the design. For example, the user would specify the value of the stem radius in the wine glass design.

In the new model of SiFAs, the user requirements can be any arbitrary set of constraints involving the design parameters. Then, a critic or a group of critics need to check for those constraints and detect a conflict when one of the constraints is not satisfied. See section 6.2 for a detailed explanation of conflict detection.

This approach has the disadvantage that one particular SiFA-based design system cannot be used to create multiple designs each of which satisfy a different set of user requirements. In order to have additional user requirements, the critics in the system need to be modified. A solution to this problem is automatic generation of critics. Since all critics checking for different user requirements are identical except for the constraint they check, it is possible to create them automatically. This is discussed further in section 10.5 as a future research direction.

4.6 Summary

This chapter explained the extensions that were made to the SiFA paradigm including the notion of a parameter block, the specific targets of agents, the knowledge contained in an agent, how the selectors and advisors are handled as one agent, and finally how user requirements are handled in the new extended model.

The next chapter deals with the possible conflicts that can occur in a SiFA-based system and presents a hierarchy of such conflicts.

Chapter 5

Hierarchy of Conflicts

This chapter presents a hierarchical classification of SiFA conflicts. Such a classification is comparable to the one described in [Klein & Lu 90], and shown in figure 2.1, in terms of its organization.

The conflict hierarchy for SiFAs is shown in figure 5.1. The nodes represent conflict types. This is an is-a hierarchy, where each node is a specialization of its parent node and therefore inherits all of its properties. The root node, labeled *conflict*, encompasses all conflict types. Klein's conflict hierarchy has this property as well. The main distinction between the two are that the SiFA hierarchy does not have any domain dependent nodes although its leaf nodes are very well defined conflicts.

It is possible to change the order of the criteria used at each level to specialize the conflicts. For example the criteria used to specialize them in the second and third levels of the right hand side of the hierarchy may be switched. This would create a different hierarchy than the one described here. The version presented here, however, does the best job of grouping common aspects of conflicts into a single more general node. That is, if we used another hierarchy, there would be some nodes that represent the same group of conflicts in more than one place in the hierarchy.

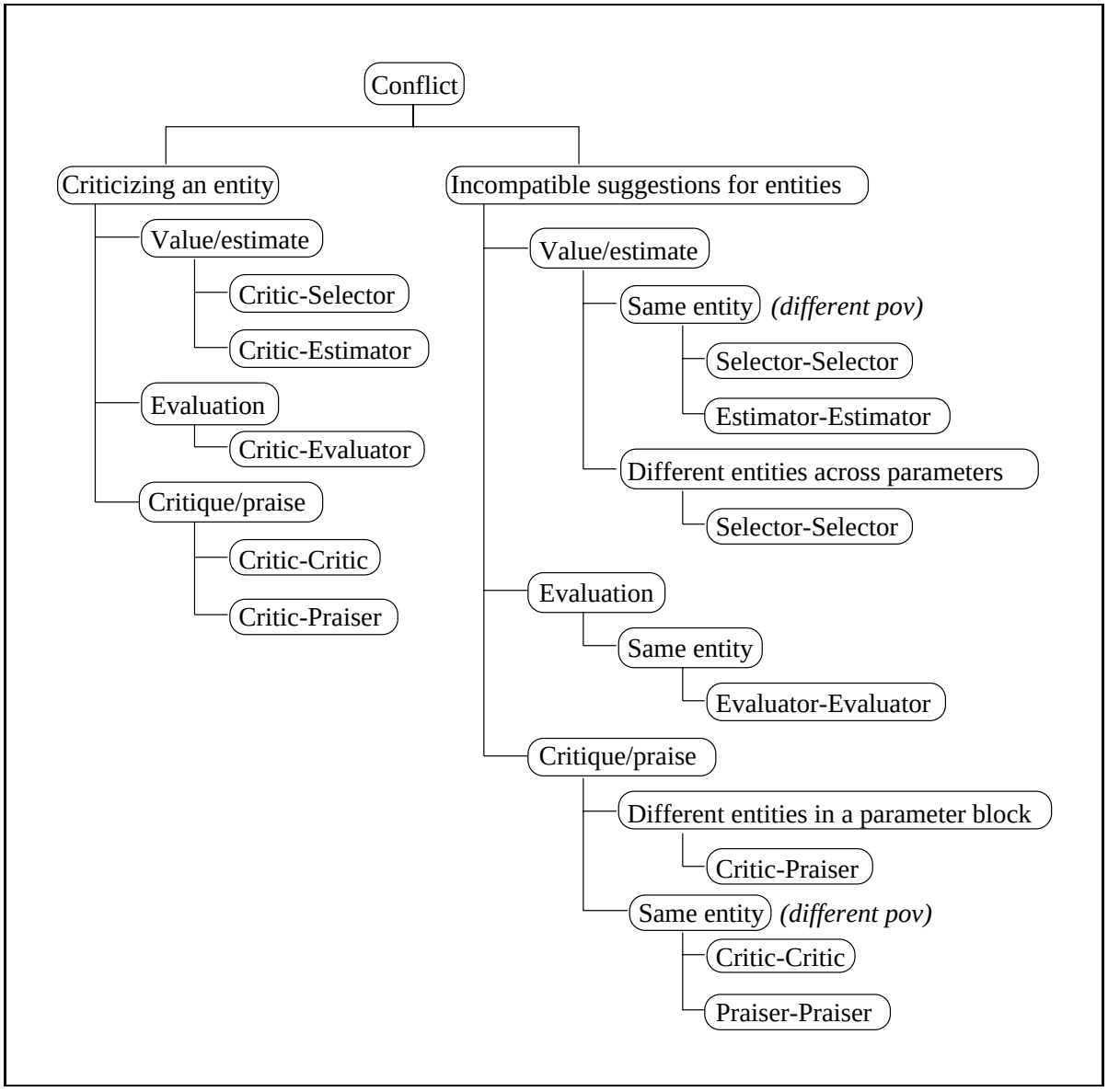

Figure 5.1: The SiFA conflict hierarchy

The following sections describe the hierarchy of conflicts in detail.

5.1 Criticizing an Entity

Criticizing an entity is the the type of conflict where a critic produces a criticism of some entity and initiates a conflict. This occurs when the critic is not satisfied with the entity it is watching over. The entity that is being watched over is the entity being criticized which is the one that the critic's target refers to. This situation is shown in figure 5.2. These conflicts are referred to as *criticism conflicts*.

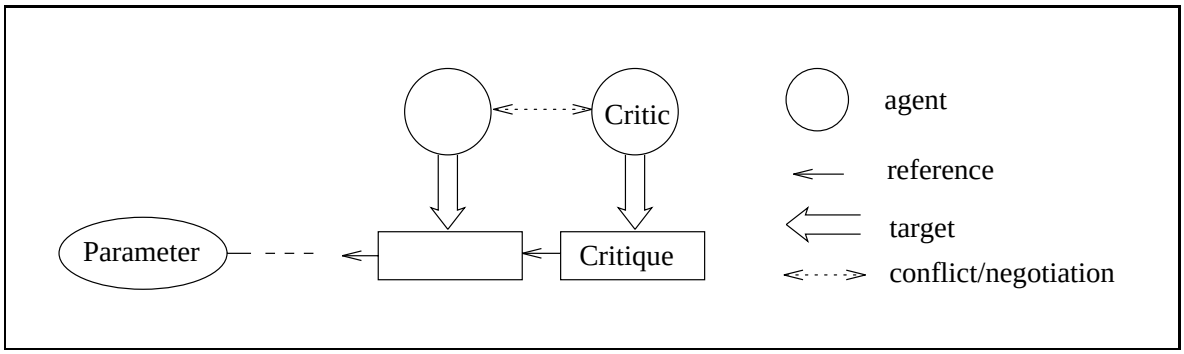

Figure 5.2: Criticizing an entity

The entity being criticized may be any one of the value, estimate, evaluation, criticism, or praise entities. The next level of specialization of the hierarchy differentiates conflicts into three based on the type of entity being criticized.

5.1.1 Value/Estimate Criticism

In this group of conflicts, the entity being criticized is a value or an estimate. There are similarities between these two entities because there can be at most one value or estimate of a parameter at any time. This conflict situation is shown in figure 5.3. The criticism of a value or estimate may be one of *too high*, *too low*, *wrong units* or *too imprecise*.

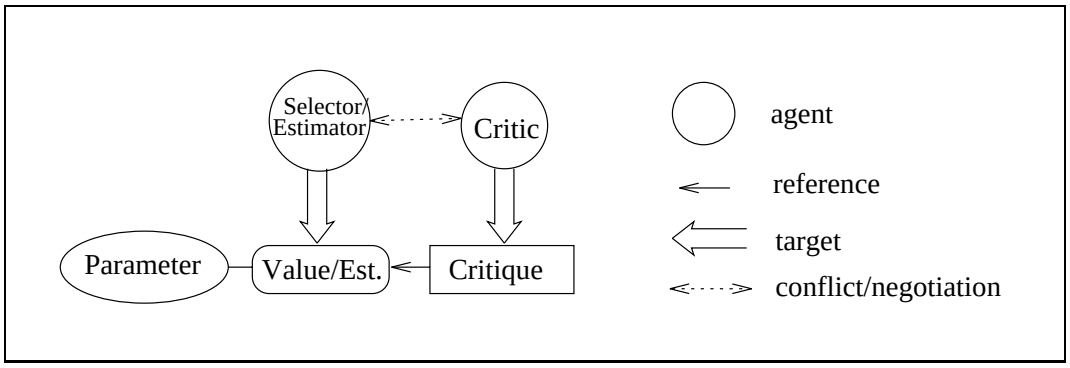

Figure 5.3: Criticizing a value/estimate

This group can further be broken down into two subgroups where the criticized entity is a value, and where the entity is an estimate, resulting in conflicts between a critic and a selector, and between a critic and an estimator respectively. Both types of conflict are initiated by the critic.

5.1.2 Evaluation Criticism

The situation where the criticized entity is an evaluation is shown in figure 5.4. An evaluation may be criticized because it is *too imprecise*, *too high*, or *too low*.

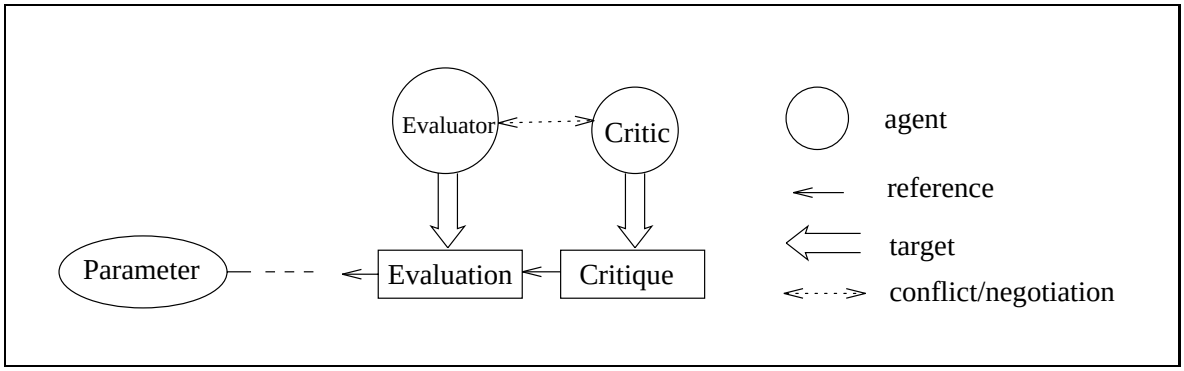

Figure 5.4: Criticizing an evaluation

All conflicts of this type are between an evaluator and a critic. The conflict is initiated by the critic.

5.1.3 Critique/Praise Criticism

Just as for any other entity, it is possible to have criticisms of either criticisms or praises that will lead to conflicts. Such a criticism may be one of *too negative*, *too positive*, *too imprecise*, or *not substantiated*. This is shown in figure 5.5.

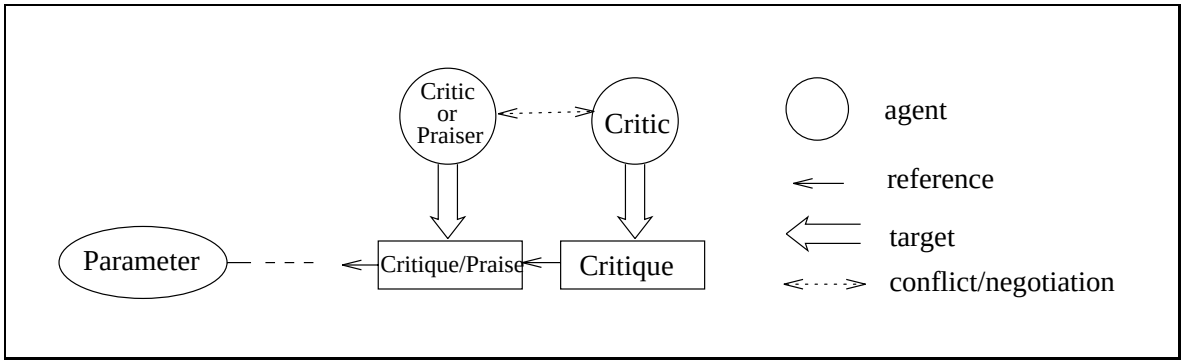

Figure 5.5: Criticizing a critique/praise

This conflict type may be divided into two groups, criticizing a critique, and criticizing a praise, resulting in conflicts between two critics, and between a critic and a praiser respectively.

5.2 Incompatible Suggestions

If there are two or more agents with the same target, or if there is a relationship between the targets of two agents, then there is a possibility for conflict among these agents. If the results of these agents are not compatible in some way, there is conflict. These conflicts are also referred to as *incompatibility conflicts*.

Similar to the left hand side of the hierarchy, the next level of specialization is based on the entity that is the subject of the conflict. Here, the subject of the conflict is not what the criticism refers to, but it is the target of the agents suggesting incompatible values, estimates, evaluations, etc.

Analogous to the right side of the hierarchy, the conflicts are divided into three groups.

5.2.1 Value/Estimate Incompatibility

Incompatible suggestions for values and estimates can occur in two ways.

The first is when there are two agents with the same target and the suggested values or estimates are incompatible. This will cause a conflict because only one value or estimate may be stored in the target entity. This situation is shown in figure 5.6. Both agents have to have the same type, selector or estimator. So the conflict is between two selectors or two evaluators. The points of view of the conflicting agents have to be different else the agents would be identical.

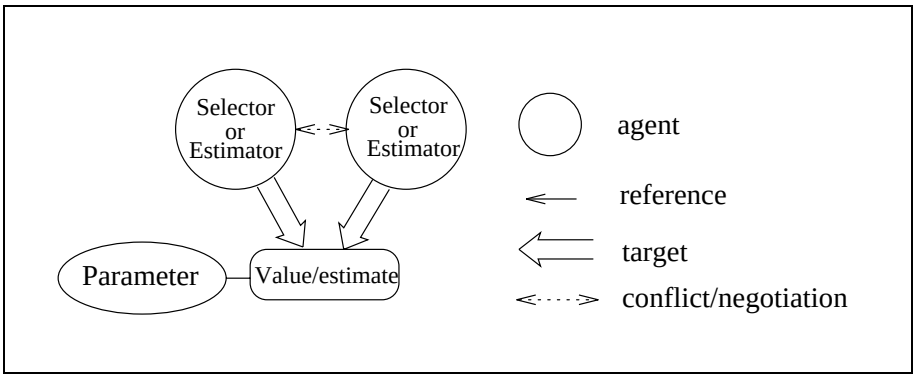

Figure 5.6: Incompatible suggestions for the same value/estimate entity

The second kind of conflict can occur between two entities across parameters. This happens when the value of one parameter is not compatible with the value of the other parameter as shown in figure 5.7. The two value entities have to be in different parameter blocks since each parameter can have only one value entity. This type of conflict is possible only between two selectors. As a simple example, consider the case in the wine glass design where the a base radius selector generates a value for the base radius and then checks the internal constraint that the base radius has

to be greater than the stem radius. If this constraint is violated, then the base radius selector will be in conflict with the stem radius selector.

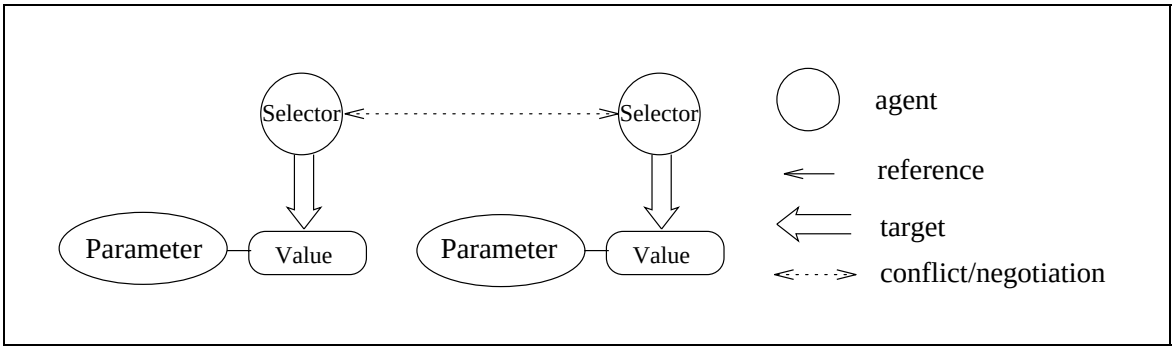

Figure 5.7: Incompatible suggestions for different value entities

5.2.2 Evaluation Incompatibility

Incompatible suggestions for an evaluation entity occurs when two evaluators with different points of view have the same evaluation entity as their target as shown in figure 5.8. This kind of conflict will occur only between two evaluators. It is not meaningful to have a conflict because of incompatibility between evaluations from two different parameter blocks. The evaluation of an entity in one parameter block is not related in any way to an evaluation in another parameter block.

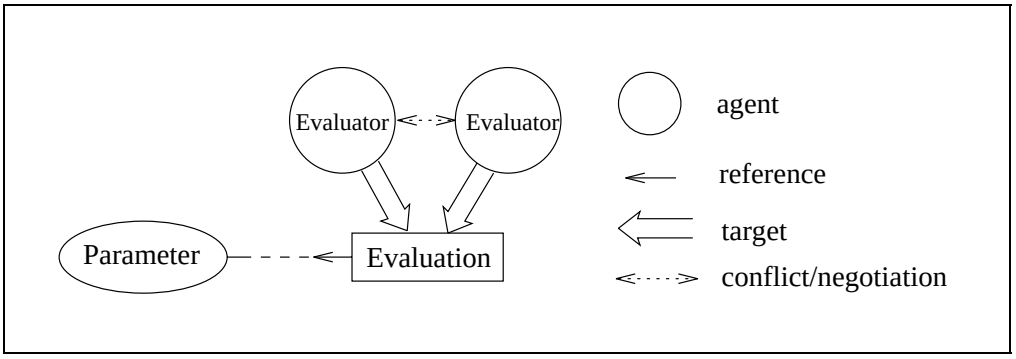

Figure 5.8: Incompatible suggestions for an evaluation

5.2.3 Critique/Praise Incompatibility

Two kinds of conflicts are possible in this group.

One kind is when the targets of two critics or two praisers with different points of view have the same target. This situation, shown in figure 5.9, is very similar to incompatible evaluations. Both agents have to be the same type, so the conflict occurs between two critics or two praisers.

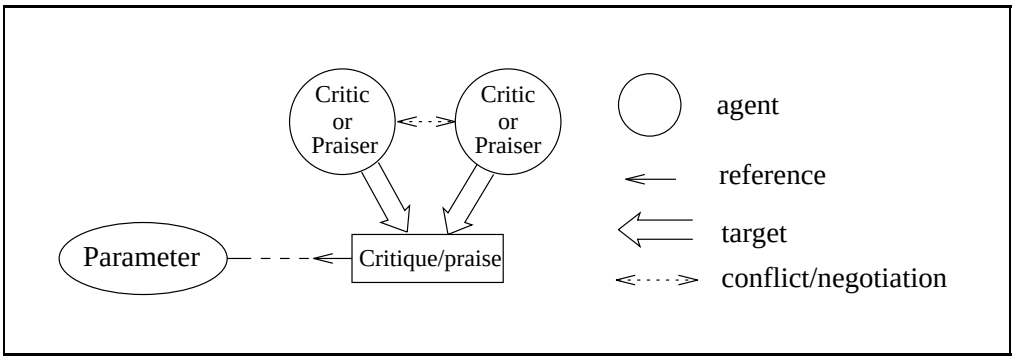

Figure 5.9: Incompatible suggestions for the same critique/praise entity

The other kind happens when there is an incompatibility between a criticism and a praise of the same entity. In this situation, shown in figure 5.10, the targets of the agents in conflict are different but the targets refer to the same entity. In other words, the criticism and the praise of an entity are not compatible. This conflict occurs between a critic and a praiser and can be initiated by either of them. The agents in conflict have to have targets in the same parameter block and also at the same level of reference, since the incompatible critic and praise refer to the same entity. This conflict is not meaningful across parameters.

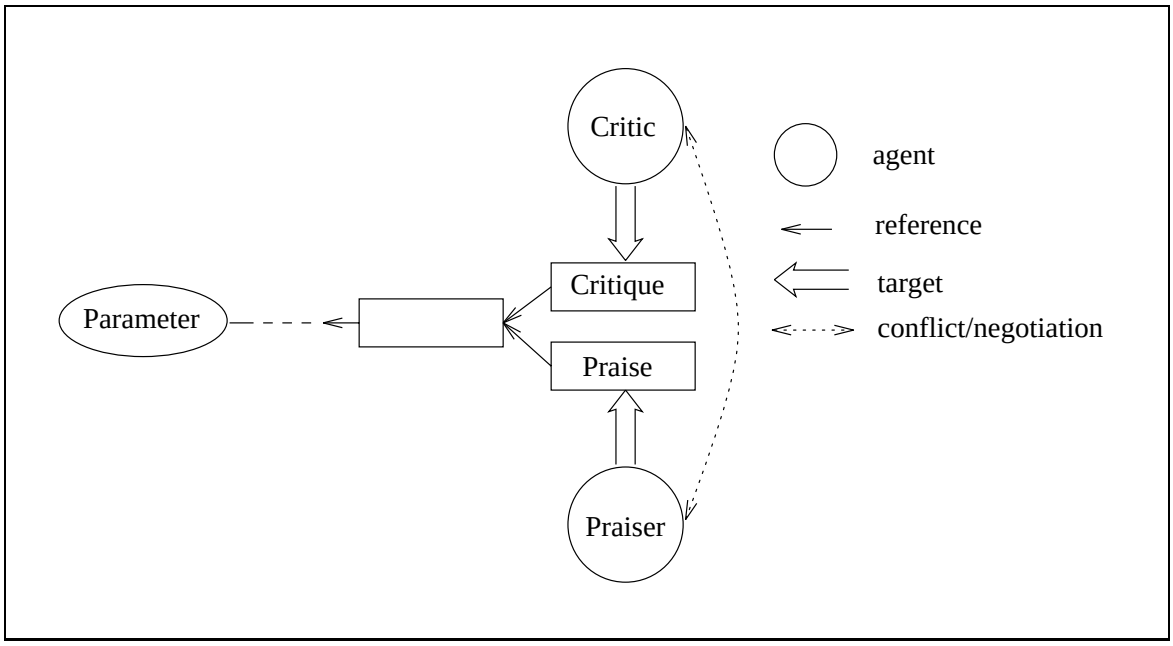

Figure 5.10: Incompatible suggestions for the different critique/praise entities

5.3 Incomplete Knowledge

This hierarchy does not contain any conflicts where one agent complains about not having enough knowledge to be able to do its job.

Such situations are possible between estimator-selector pairs, and evaluator-selector pairs. Suppose a selector chooses a material. Later an estimator, that is supposed to produce an estimate for another parameter, needs to use the value of the material parameter, but does not have knowledge about the particular material chosen.

Alternatively, an evaluator that is supposed to evaluate the particular material does not know about that material and therefore cannot evaluate it.

These situations are neither criticism conflicts, nor incompatibility conflicts. When an estimator or an evaluator does not know about the value produced by an estimator, learning should take place. The selector should be notified of the situation by a complaint message from the agent with the incomplete knowledge. Then, the selector

should send information about its choice to the complaining agent. This is only one of the many possibilities for learning in the SiFA paradigm. Learning in multi-agent systems is a future research direction discussed in section 10.4.

5.4 Domain Independence

Although this hierarchy is very fine grained, it is highly domain independent. There isn't anything that is particular to any domain such as mechanical or electrical design in the hierarchy. It applies equally to all parametric and routine design problems which can be solved by SiFAs.

The hierarchy could be expanded for another level from the leaf nodes by specializing based on the points of view of agents participating in the conflict. If some points of view can be grouped together into classes that are present in all domains, such as points of view about manufacturability, durability, and cost, then the additional level of the hierarchy would again be domain independent. Any other specialization, for example, one that involves targets or parameters, such as stem length in the wine glass design, would certainly be highly domain dependent. This point will be discussed as a future research issue in chapter 10.

5.5 Summary

The hierarchy of conflicts in a SiFA based system was presented in this chapter. Each type of conflict in the hierarchy was discussed separately, followed by a discussion of how incomplete knowledge should be handled and how the proposed hierarchy is domain independent.

The next chapter explains all aspects of the negotiation process in the SiFA framework.

Chapter 6

SiFA Negotiations

This chapter describes the negotiation process by which conflicts between SiFAs can be resolved.

Before the actual negotiation can start, the system has to go through a few preliminary steps. These are indication of a possible conflict, detection of the conflict, selection of a negotiation strategy to use, and refinement of that strategy. Only then can the actual negotiation take place. The actual negotiation is simply the execution of the selected strategy. All of these steps put together is called the *negotiation process*. The result of the process is either a solution to the conflict or the signaling of a failure.

Some of the steps in the negotiation process are very trivial in the SiFA paradigm since the conflicts are very well defined and the number of different kinds of conflicts an agent can be in is very limited. It is important that the steps be simple as SiFAs have limited computational power.

All of the negotiation process steps are explained in the following sections.

6.1 Conflict Indication

Agents need to be able to notice situations where there is the possibility of a conflict. This act of realizing a potential conflict is called *conflict indication*.

Conflict indication is a very simple task for the agents in the SiFA paradigm.

Each time an agent acts on its target by modifying it, the state of the overall system changes, therefore there is potential for a conflict. The way a possible conflict is indicated depends on to which one of the two general conflict categories the conflict belongs.

6.1.1 Indication of a Criticism Conflict

Conflicts which are caused by the criticism of an entity, the ones on the left hand side of the conflict hierarchy, are noticed by the critic producing the criticism. Each time a critic produces a criticism, there is a conflict indication.

The production of a criticism may also cause an incompatibility conflict if the target entity holding the criticism is already storing a critique. This is discussed in the next subsection.

6.1.2 Indication of an Incompatibility Conflict

Whenever an agent acts on its target by producing a value, estimate, evaluation, praise, or criticism, it checks to see if there is the possibility of a conflict.

For an agent to indicate a same entity conflict, all of the three conditions given below have to be satisfied:

- The target entity already contains a value, estimate, evaluation, critique, or praise,
- The owner of the entity (the agent who put in the old contents) is not itself (the agent acting on the target now), and
- The old value, estimate, evaluation, critique or praise is not identical to the new one the agent is attempting to put into the entity.

If all these conditions are true, the agent notices the possibility of a conflict and indicates a conflict. The indication of a conflict does not mean that there is necessarily going to be a conflict. Once there is indication of a conflict, the conflict detection task will determine if there is actually a conflict or not.

For an agent to indicate a conflict involving two entities within a parameter block or across parameter blocks, the agent has to know which entities are related to its target, and check the contents of those entities. The agent cannot simply look for identity between its target and the related entities. It should use a simple metric of incompatibility between the contents of the entities in order to decide if there is the possibility of a conflict and whether the more costly conflict detection task should be carried out.

6.2 Conflict Detection

After an agent indicates a conflict, it is that same agent's responsibility to figure out whether there is actually a conflict or not. Suppose there is a 0.01% difference in the values proposed for the value entity of a parameter by two different selectors. Unless that value is extremely sensitive, this small difference does not constitute a conflict. So the indication of a conflict does not necessarily mean that there is actually a conflict.

Conflict detection is a knowledge based task and it is not always trivial. How detection works in the SiFA paradigm depends on whether the conflict is a criticism conflict or an incompatibility conflict.

6.2.1 Detection of a Criticism Conflict

As soon as a critic produces a criticism and indicates that there is a potential conflict, it has to decide if the criticism actually signals a conflict or not.

This is usually an easy task for the agent since it knows the reason why it produced that particular criticism. If the criticism was due to the violation of a hard constraint that tests some physical properties or user requirements, then there is probably a conflict. On the other hand, if the violated constraint that produced the criticism was just a preference of that agent, then there is no conflict and the design can go on. The criticism stays on the blackboard and the agents whose targets were criticized may take it into consideration if they wish.

6.2.2 Detection of an Incompatibility Conflict

Detecting an incompatibility conflict requires more knowledge and is usually a more complicated task than detecting criticism conflicts.

There are three possible cases, conflicts about a single entity where the targets of two agents are the same entity (figure 6.1a), conflicts involving different entities within a parameter block (figure 6.1b), and conflicts between entities across parameter blocks (figure 6.1c).

Same Entity Conflicts

Once the agent notices that what is already stored in its target entity is not identical to what it wants to put in that entity, and indicates a conflict, the agent has to decide if what it wants to store is compatible with what is already there. If they are compatible then there is no conflict although they may not be identical. It is not sufficient to base conflict detection simply on checking if values, estimates, evaluations, criticisms, or praises are exactly identical.

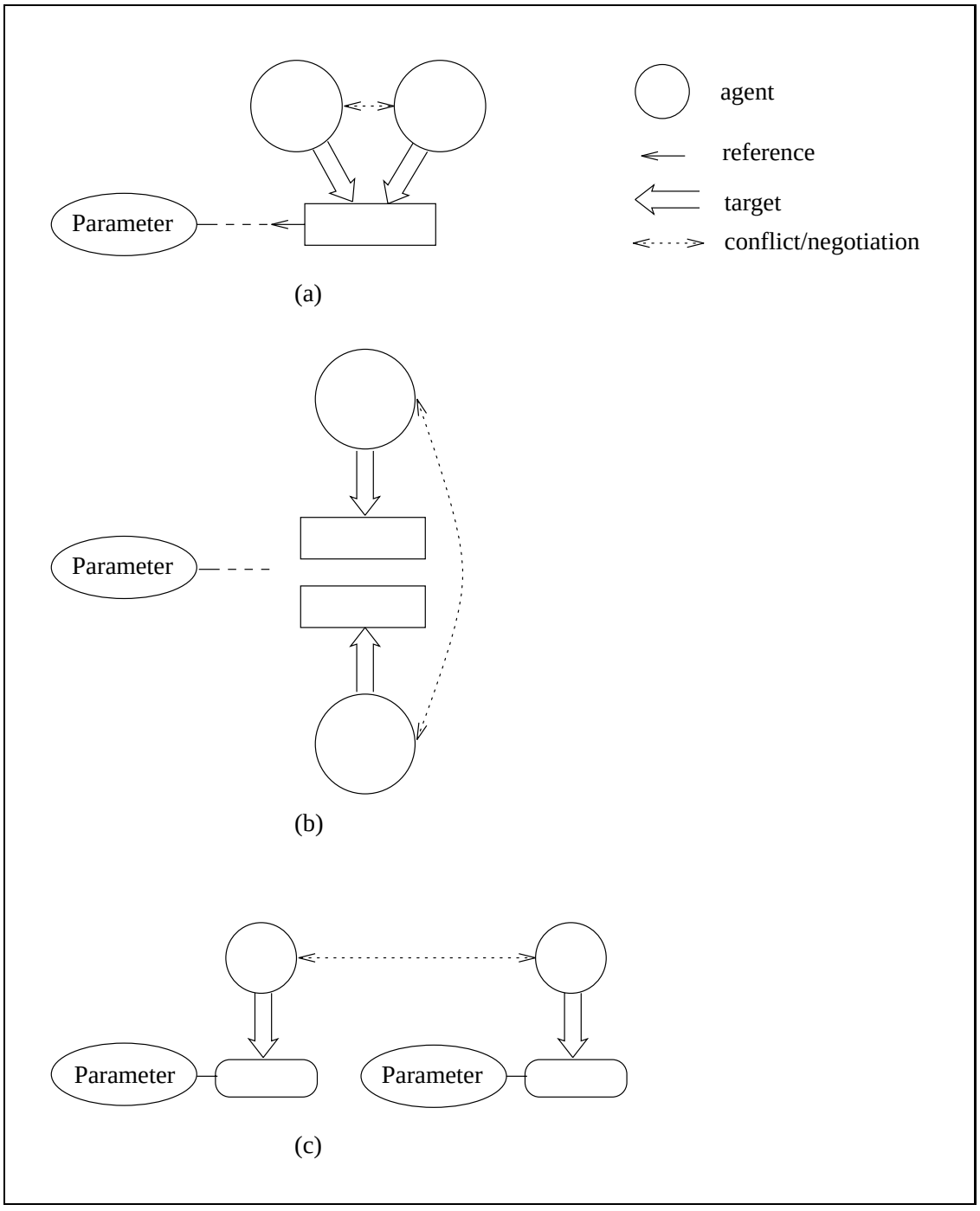

Figure 6.1: Types of incompatibility conflicts

This situation is best exemplified in the case of values. Suppose a design parameter can take on continuous numeric values, and its current value has been set to 5.00 by selector A. If later selector B calculates a value for the same parameter from a different point of view and comes up with 5.01, it indicates a possible conflict, but it has to do more work in order to decide if there is indeed a conflict or not. Selector B needs to know how sensitive the value of the parameter is. This sensitivity also depends of selector B's point of view, or what selector B is trying to optimize. If allowing up to 1% changes in the value of the parameter is acceptable by selector B, then there is no conflict. Selector B can just leave the value as 5.00 and does not need to insist on 5.01. Similar situations can occur with other agent types and entities as well.

For agents to be able to detect incompatibility conflicts, they should have the ability to decide if something that is stored in their target is compatible with what they produce. Target entities may be numeric or symbolic. Value entities would be numeric for parameters like length, but symbolic for parameters like color or material. Estimates would generally be of the same type as the value of the parameter. Evaluations may be symbolic such as good, average, bad, or numeric if the quality is measured as a percentage. Critiques and praises may contain both numeric and symbolic information.

If the target entity is numeric, then the agent should have either a percentage or a fixed value by which the numeric value, estimate or evaluation it produces may be relaxed. For example, the agents wants the value to be 50, but can relax 5%, so anything between 45 and 55 is fine, or relaxation by only 3 units is possible so anything between 47 and 53 is acceptable.

If the target entity is symbolic, there are two possibilities. If there is an ordering to the symbols, such as bad, average, good, then the agents can have a constraint such as "anything better than average is acceptable". If there is no ordering to the symbols than the agents need to have a list of things that they find acceptable.

Different Entities Within a Parameter Block

Detecting this kind of conflict is the same as detecting conflicts involving one entity as far as the knowledge based reasoning is concerned. The difference is that the agent is not comparing something that is already in its target with what it wants to store, but it is comparing some other entity in the parameter block with what it wants to store in its target.

This means that agents need to know what entities have the potential to be incompatible with their targets. This is not a big problem since critics and praisers are the only kind of agents that can get into such conflicts because of incompatible criticisms and praises of the same entity.

Different Entities Across Parameter Blocks

This case is very similar to detection of conflict involving entities within a parameter block. The only difference is that only selectors can have such conflicts. This means that selectors need to know what values are related to the value they are producing and check for the constraints between these values.

6.3 Conflict Classification

After an agent detects a conflict, it has to figure out what kind of conflict it is. This is a very simple task in the SiFA paradigm. First of all each agent can be involved in a very limited set of conflicts. Also, the detection process already gives the agent enough information to immediately classify the conflict as one of the leaf nodes of the conflict hierarchy.

If a critic detects a conflict after producing a criticism because some hard constraint is violated, then it knows that it is involved in a criticism conflict. It also

knows if the conflict is about a value, estimate, evaluation, critic or praise because it knows what it is criticizing, the entity its target refers to. The type of the agent it is in conflict with follows immediately from the type of the entity. A value entity means that the other agent is a selector, an estimate means an estimator, a criticism means a critic, etc. Finally, it can easily figure out what particular agent it is in conflict with by inspecting the *owner* field of the entity that is the subject of the conflict. The owner field of an entity contains the name of the agent that put the current value, estimate, evaluation, critique, or praise into the entity.

If an agent detects a conflict when acting on its target because its target entity or some other entity related to the target already contains something which is not compatible, then the agent knows that it is involved in an incompatibility constraint. It can deduce the type of the entity from the knowledge of its target and the related entity if the conflict involves more than one entity. The agent with which there exists a conflict is figured out by looking at the owner field of the entity causing the incompatibility.

6.4 Negotiation Strategy Selection

A *negotiation strategy* is a body of knowledge, usually representable by a set of rules, that allow the agent to carry out a negotiation. The negotiation strategies for an agent are also called its *negotiation modes*.

After a conflict has been detected and classified, the agent that detected the conflict has to start negotiating, as the initiator, with the conflicting agent, referred to as the *partner*. In order to do that, it has to select a negotiation strategy. Then, the partner also has to select a strategy.

In the context of SiFAs, since the conflicts an agent can be involved in are very specific, the number of strategies an agent can have for any conflict are also limited.

As the number of conflicts an agent can be involved in is also small, the total number of strategies it needs is small as well. Usually a SiFA has a single strategy to deal with a certain conflict, but more than one strategy for a single conflict type is also possible.

If an agent has a single negotiation strategy for the conflict it is in, then the strategy selection task is trivial, there is a one to one match between the conflict and the strategy.

If the agent has more than one strategy for a particular conflict, then it needs a criteria to select its negotiation strategy. This criteria may be the point of view of its partner or the particular value, estimate, evaluation, criticism, or praise that is the the subject of the conflict. Suppose a selector A suggests a value of 5.00 for a parameter, then selector B detects a conflict because the value 5.00 is more than 10% away from what it considered to be the ideal value. Then it can choose its negotiation strategy based on the fact that the value is 10% away from the ideal by using a rule such as "if the difference is less than 10% use strategy 1 else use strategy 2".

The partner also needs to select a strategy. The partner can classify the conflict based on the first message it receives from the initiator. This message identifies the initiator, its target, and what the conflict is about. The partner is aware of its own target, so when it receives the first message in the negotiation, it is able to classify the conflict. Then it can select a negotiation strategy in a fashion similar to the initiator.

It is also possible for agents to switch from one negotiation mode to another during the execution of the negotiation strategy, based on the interaction with the other agent. This interaction is in the form of messages the agent receives. This is discussed later in section 6.6.

6.5 Negotiation Strategy Refinement

The negotiation strategy selected may still be a generalized strategy that has to be instantiated by giving values to some variables of the strategy.

The main reason why an agent would have generalized strategies is to save space. The agent may need a few different strategies for a type of conflict, and if these strategies are very similar except for a few minor points, then it is better to have only one generalized strategy rather than storing all these similar strategies separately. The generalized strategy can have variable parts to allow for the differences in the group of similar strategies it represents. The generalized strategy can then be instantiated using some criteria similar to the ones in strategy selection.

Another reason why a generalized strategy may be useful is if the agent does not have enough information to select a strategy before the negotiation starts. In this case, the agent can start negotiation using the generalized strategy and then instantiate it during the negotiation as more information becomes available.

6.6 Negotiation Strategy Execution

After conflict indication, detection, strategy selection and refinement, the agents are finally ready to negotiate. The negotiation consists of the initiator's and the partner's execution of their respective strategies.

It is possible for these strategies to clash in the sense that the agents will not be able to reach an agreement. In such a case either the conflict will not be resolved and a the design will halt with a failure, or one or both agents will change their strategies during the negotiation. The change of strategy is explained in section 6.7.

6.6.1 Negotiation Messages

The negotiation strategies are usually a set of rules that specify how to initiate the negotiation and what to do in response to a received message. The messages contain KQML-based utterances. Each message has a primitive such as *tell* and *ask*, the sender, the receiver, a subject which is the entity that the message is concerned with, such as the cup radius value, and two other slots containing the body of the message. Three example messages are shown in tables 6.1, 6.2, and 6.3.

Table 6.1: An "ask alternative" message

Primitive	Ask
From	Cup radius Stability Critic
To	Cup radius Stability Selector
Subject	Cup radius Value
Slot 1	Alternative
Slot 2	

Cup radius Stability Critic asks Cup radius Stability Selector to propose an alternative value for the cup radius value

Table 6.2: A "tell no alternative" message

Primitive	Tell
From	Cup radius Stability Selector
To	Cup radius Stability Critic
Subject	Cup radius Value
Slot 1	Alternative
Slot 2	none

Cup radius Stability Selector tells Cup radius Stability Critic that there no possible alternatives for Cup radius Value.

Table 6.3: An "ask relaxation" message

Primitive	Ask
From	Cup radius Stability Selector
To	Cup radius Stability Critic
Subject	Cup radius Value
Slot 1	Relax
Slot 2	

Cup radius Stability Selector asks Cup radius Stability Critic to relax its constraint for Cup radius Value.

6.6.2 Negotiation Graphs

It is possible to represent a negotiation between two SiFAs using *negotiation graphs*. A negotiation graph shows all possible sequences of messages sent and actions performed by the agents involved in a particular kind of conflict.

The start node of the graph shows the message sent by the initiator of the conflict. A path through a negotiation graph starting at the start node and ending at a node without any outgoing arcs is a full transcript of a possible negotiation. Whenever there is more than one outgoing arc from a node, this means that the negotiation can proceed in any one of those ways. Which one of those arcs is taken depends on the state of the particular agents involved in the negotiation.

Figure 6.2 shows a negotiation graph for a conflict between a critic of a value and the selector of that same value. What this graph indicates is that the critic asks the selector for alternatives and the selector either provides more alternatives or asks the critic to relax its constraint. If the selector proposes an alternative value, the critic may still disagree, and if the critic relaxes its constraint, the relaxed constraint may still be violated. So the negotiation process is a cycle of proposals of alternatives and constraint relaxations. The negotiation either ends in success if the critic is satisfied with the value or in failure if the critic cannot relax its constraint anymore and the

selector can give no other alternative values for the parameter.

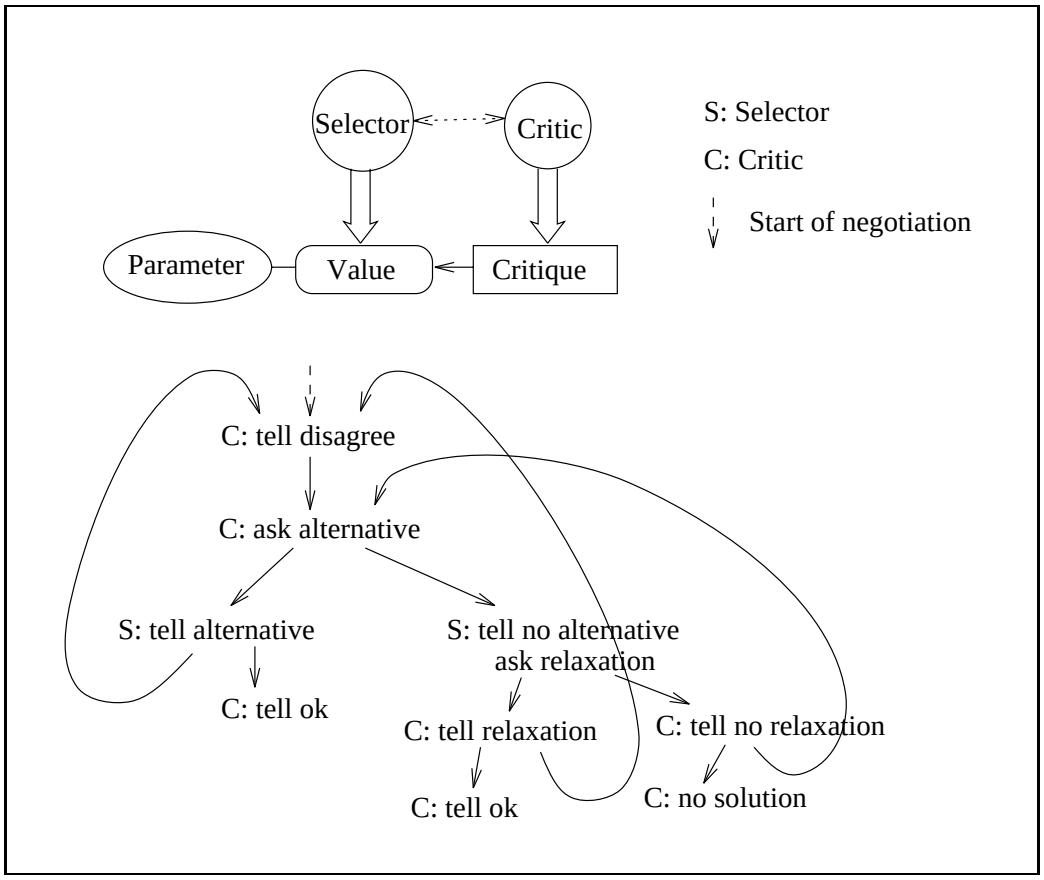

Figure 6.2: Selector-Critic negotiation graph

It should be noted that the negotiation graph does not give any information about the order of constraint relaxations by the critic and alternative proposals by the selector. This is dependent on the respective agents and the strategies they use. The selector may use one of the following strategies, where the first is one a *greedy* strategy, the second is *maximally cooperative*, and the third is a mixture of the two:

- When asked to supply an alternative value, decline and ask for a relaxation immediately,
- When asked to supply an alternative, give alternatives until there are no more

possible alternatives, only then ask for a constraint relaxation,

- Supply alternatives and ask for relaxations in an interleaved manner, possibly giving one alternative and asking for a relaxations before giving another alternative.

Similarly, the critic may use a greedy, maximally cooperative or mixture strategy as follows:

- Wait for a "no more alternatives" message from the selector before relaxing a constraint,
- Relax a constraints before asking for alternatives,
- Relax constraints and ask for alternatives in an interleaved manner, possibly doing one relaxation and asking for one alternative before doing another relaxation.

Figure 6.3 shows another negotiation graph, this time for a conflict between two selectors with the same target. As for the selector-critic conflict, the selectors can ask each other for alternative values or constraint relaxations in any order that is dictated by their respective strategies. The left and right halves of the graph are identical except that the roles of selector1 and selector2 are reversed.

Negotiation graphs are helpful in building the negotiation knowledge into a SiFA since the graphs show all messages that the agent will need to respond to and all possible answers that it can give to a specific message for each conflict type that it can be involved in. Since the conflict situations in the SiFA paradigm are very well specified, the negotiation graphs are fairly small and easy to build.

The drawback of using negotiation graphs is that they get very complicated very quickly if we wish to allow the agents to use a larger vocabulary so that they can

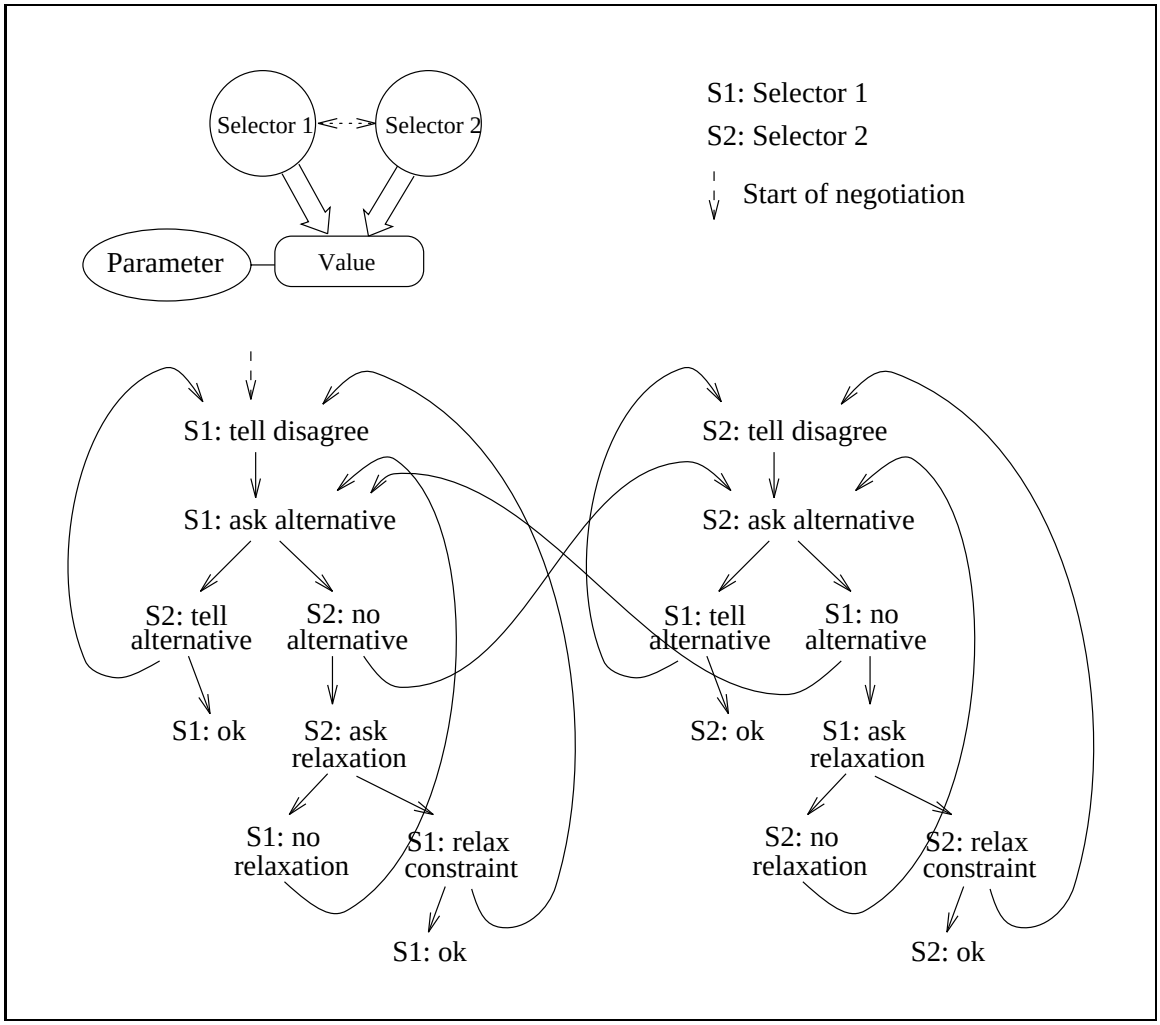

Figure 6.3: Selector-Selector negotiation graph

talk about things other than alternatives and relaxations. If the agents are to have history keeping and learning capabilities, then they do need a larger vocabulary and then it becomes infeasible to represent all possible paths of interaction between two agents. In that case, the designers of a SiFA should try to anticipate in advance what messages the agent they are building should be able to respond to, and build that capability into the agent.

6.7 Changing Negotiation Modes

Agents have the capability to change their negotiation strategy during negotiation. They can be in one negotiation mode at the beginning of the negotiation, and in another mode later in the negotiation. What prompts an agent to change its mode of negotiation is the interaction with its partner or its own internal state.

An example of changing modes triggered by the interaction is when a selector changes its negotiation strategy depending on whether it receives a message asking for an alternative value or a message asking for a constraint relaxation.

An example of changing modes prompted by an agent's internal state is when a selector changes strategy when it runs out of alternatives. Suppose a selector is asked for alternative values and it supplies them until it can no longer give possible alternatives. At that point it can go into a *don't care* mode meaning that it does not care what the value of the parameter is and the other agent can have any value it likes. This is possible if the agent's point of view is not very critical, such as style or colorfulness as opposed to strength or cost.

6.8 Emergent Behavior

The negotiation graphs do not fully determine how a negotiation will proceed since there are multiple paths through a negotiation graph. The particular path followed in a negotiation depends on the context in which the conflict occurred, the particular strategies of the involved agents, their internal knowledge, and their internal states. There is also the possibility of changing negotiation modes during negotiation.

This means that the the negotiation behavior of the agents cannot be predicted accurately in advance and the systems behavior as a whole will emerge at run-time as a function of the negotiation capabilities of the agents in the system.

Furthermore, if the agents are built without a priori knowledge of the negotiation graphs, then it is not possible to predict even the general structure of the negotiations before run-time.

Although each SiFA is extremely specialized and well defined in what can do and how it behaves in every situation, the power, flexibility, and unpredictability of a design system based on the SiFA paradigm, comes from the richness of the interaction between SiFAs. This rich interaction is the result of the negotiation process described in this chapter.

All of this potential for variability and the emergence of unexpected overall behavior of the SiFA based design system allows for the possibility of producing designs that will be judged as being creative, at least by the designers of the SiFA system.

6.9 Summary

In this chapter, a full view of the negotiation process in the SiFA framework was presented. The individual steps of conflict indication, detection, classification, negotiation strategy selection, refinement and execution were discussed in detail with

explanations of how they can be carried out.

The next chapter provides excerpts of sample runs of COSINE to demonstrate the ideas presented in this chapter.

Chapter 7

Selected Conflict Examples

This chapter contains annotated excerpts from sample runs of COSINE. The aim of the chapter is to give concrete examples of the more interesting conflicts that were introduced in the previous chapters, along with their resolution in the SiFA paradigm as implemented in the wine glass designer COSINE.

The sample runs are annotated with figures showing the conflict situation and some text explaining the behavior of the system.

A complete run of COSINE where these conflicts may be found in their full context is given in Appendix B.

7.1 Evaluator-Critic Criticism Conflict

This is a conflict between the cup radius value evaluator from the volume point of view and its critic, as shown in figure 7.1. Since the evaluation of the value is a second level entity, the criticism of that evaluation is a third level entity. This is an interesting conflict because it involves this third level entity.

Once there is a value for the cup radius, the preconditions of the evaluator are satisfied, so it provides the evaluation *good* from the volume point of view since the difference between the volume of the cup and the ideal value for volume, which is 20 cc. according to this evaluator, is less than 3 cc. This is the limit the volume can

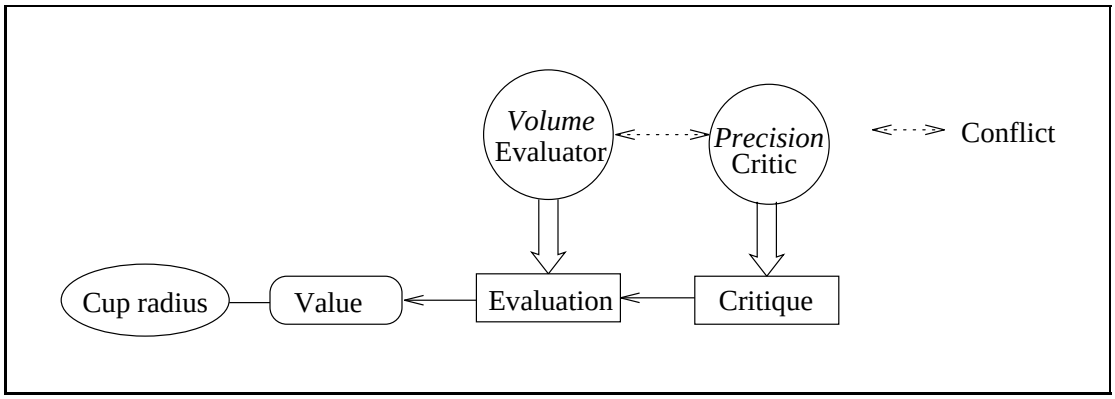

Figure 7.1: Criticism of an evaluation

deviate from the ideal and still be considered to be good. The critic of the evaluation has a constraint which says that the evaluation should be numeric, not symbolic. As this constraint is violated, the critic produces a criticism and initiates a conflict. When the evaluator is asked for a more precise evaluation, it supplies an evaluation represented as a percentage of the optimal quality that can be achieved from its point of view. This percentage represents how close to the ideal volume the current volume is and is calculated using the formula $(1 - |20 - currentvolume|/20) \times 100$ where 20 is the ideal value for volume for this particular evaluator.

Here is the excerpt:

```
Cup Radius Value Volume Evaluator: Evaluation asserted. Cup
radius value is good from a volume pov.

Cup Radius Value Evaluation Precision Critic: Conflict detected
with Cup Radius Value Volume Evaluator.

Cup Radius Value Evaluation Precision Critic: Criticism asserted.
Cup radius value evaluation is too imprecise.
```

```
Cup Radius Value Evaluation Precision Critic: Ask Cup Radius
Value Volume Evaluator to offer a more precise evaluation.

Cup Radius Value Volume Evaluator: Changing to precision
evaluation mode.

Cup Radius Value Volume Evaluator: Evaluation asserted. Cup
radius value's quality is 88 percent.

Cup Radius Value Evaluation Precision Critic: Criticism
retracted.  Cup radius value evaluation is not imprecise anymore.

Cup Radius Value Evaluation Precision Critic: Conflict with Cup
Radius Value Volume Evaluator resolved.
```

7.2 Critic-Praiser Incompatibility Conflict

This is a conflict between the critic and the praiser of the cup radius value as shown in figure 7.2. Both agents have style as the point of view of. That is, they are trying to decide if the value of the cup radius makes the cup look good or not. The conflict between these two agent involves two entities within the same parameter block, one praise and one criticism.

Once there is a value for the cup radius, the critic produces a criticism of the value saying that it makes the cup ugly. The critic does not initiate a conflict though. The criticism produced just reflects a preference and not necessarily a conflict which would require the design process to stop. Then, the praiser produces praise saying the opposite of what the critic said. As the praiser disagrees with the criticism, it

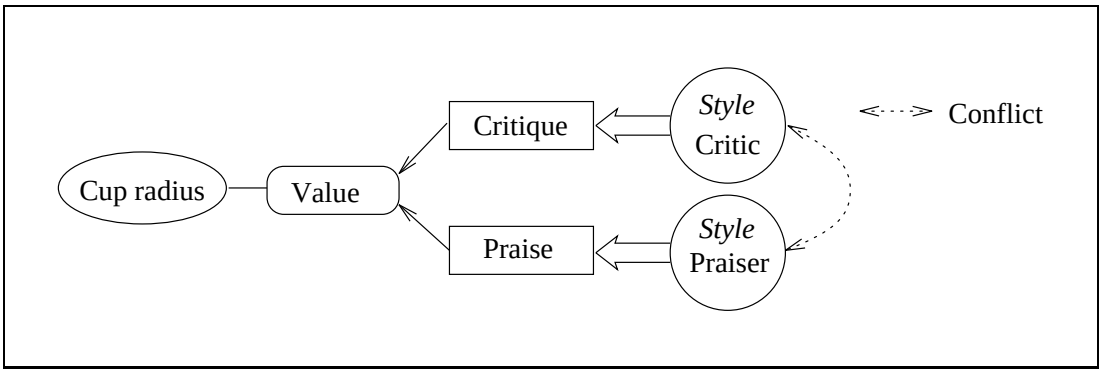

Figure 7.2: Critic-praise incompatibility

initiates a conflict with the owner of the criticism. When the praiser asks the critic to retract its criticism, the critic does so, and the conflict is resolved.

Here is the excerpt:

```
Cup Radius Value Style Critic: Criticism asserted. Cup radius
value makes cup ugly from style pov.

Cup Radius Value Style Praiser: Praise asserted. Cup radius value
makes cup beautiful from style pov.

Cup Radius Value Style Praiser: Conflict detected with Cup Radius
Value Style Critic .

Cup Radius Value Style Praiser: Ask Cup Radius Value Style Critic
to remove its criticism.

Cup Radius Value Style Critic: Criticism retracted.

Cup Radius Value Style Praiser: Conflict with Cup Radius Value
```

```
Style Critic resolved.
```

In some situations, it is possible for the praiser to initiate the conflict before producing its praise, and to assert the praise only after the conflict has been resolved. This depends on whether the praiser gives more importance to doing its job, which is to produce praise, or to detect possible conflicts. In the previous example, the praiser preferred to produce the praise first.

7.3 Estimator-Estimator Incompatibility Conflict

This is an incompatibility conflict between two cup radius estimators with the same target and different points of view as shown in figure 7.3. It is very similar to conflicts between two selectors with the same target. The idea of a conflict between estimators makes it interesting though.

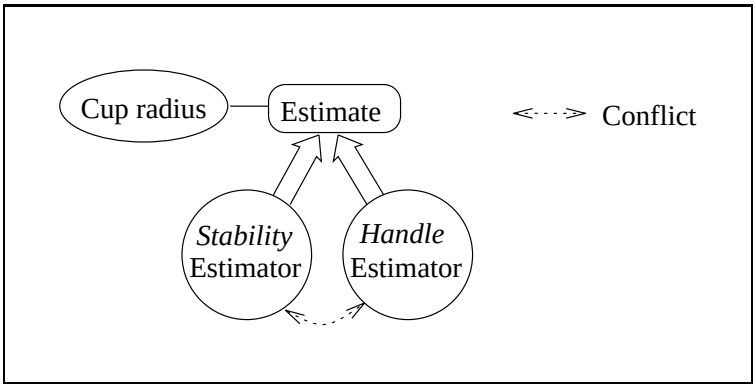

Figure 7.3: Estimate-estimate incompatibility

At some point during the design, the estimator with the stability point of view makes an estimation using the formula $cupradius \approx baseradius \times 2$. This estimate is not compatible with the estimate the estimator with the handleability point of view wishes to make using the formula $cupradius \approx stemlength/2$. Therefore a conflict is initiated by the latter. The estimator with the handleability point of view

asks the estimator with the the stability point of view to give an alternate estimate. The stability estimator produces the alternate estimation by changing the constant coefficient in its estimation formula from 2 to 1.5. The handleability estimator finds this alternate estimate to be compatible with its own estimate, hence the conflict is resolved.

Here is the excerpt:

```
Cup Radius Stability Estimator: Cup radius estimated to be
2.0 cm.

Cup Radius Handleability Estimator: Conflict detected with Cup
Radius Stability Estimator.

Cup Radius Handleability Estimator: Ask Cup Radius Stability
Estimator for an alternative estimate.

Cup Radius Stability Estimator: Using alternative estimate.

Cup Radius Stability Estimator: Cup radius re-estimated to be
1.5 cm.

Cup Radius Handleability Estimator: Conflict with Cup Radius
Stability Estimator resolved.
```

7.4 Selector-Selector Conflict Across Parameter Blocks

Only selectors are able to have conflict with agents whose targets are in other

parameter blocks. These conflicts are among the most widely studied in the conflict resolution and constraint satisfaction literature. This example shows how they can be handled in the SiFA paradigm.

Here the stem radius selector with strength point of view and the stem length selector with stability point of view are in conflict, as these two values are related in some way through a constraint in at least one of the agents‘ knowledge bases.

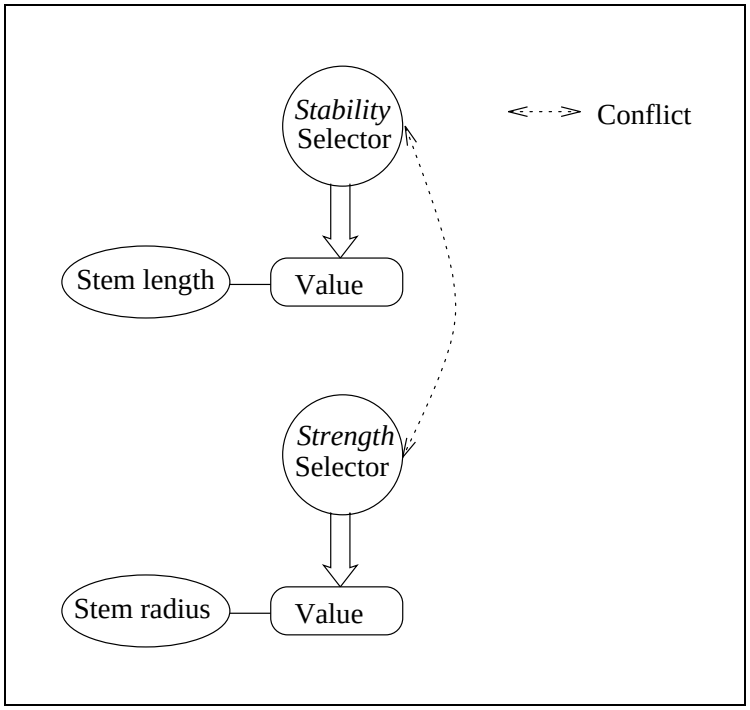

Figure 7.4: Value-value incompatibility across parameter blocks

The stem length selector sets the value of the stem length to a default value. At some point later in the design, the stem radius selector produces the value of the stem radius using the value of the stem length and the formula $stemradius = stemlength/10$. The stem radius selector realizes that the value it produced violates one of its constraints, namely, $|stemlength - stemradius| < 3cm$. Therefore it initiates a conflict. When the stem length selector produces an alternative, the stem radius

selector is able to produce a value which satisfies all constraints it knows about, and the conflict is resolved.

Here is the excerpt:

```
Stem Length Stability Selector: Set stem length to default value
of 4 cm.

Stem Radius Strength Selector: Set stem radius value to 0.4 cm.

Stem Radius Strength Selector: Conflict detected with Stem Length
Stability Selector.

Stem Radius Strength Selector: Ask Stem Length Stability Selector
for an alternative value.

Stem Length Stability Selector: Using alternative value.

Stem Length Stability Selector: Redesigned stem length value to
3 cm.

Stem Radius Strength Selector: Conflict with Stem Length
Stability Selector resolved.

Stem Radius Strength Selector: Redesigned stem radius value to
0.3 cm.
```

7.5 Selector-Critic Criticism Conflict

This conflict is initiated by the cup radius critic with the stability point of view when it criticizes the value of the cup radius. This situation is shown in figure 7.5. The owner of the value is the cup radius selector whose point of view is also stability. The purpose of this conflict is to show the use of constraint relaxations in negotiation.

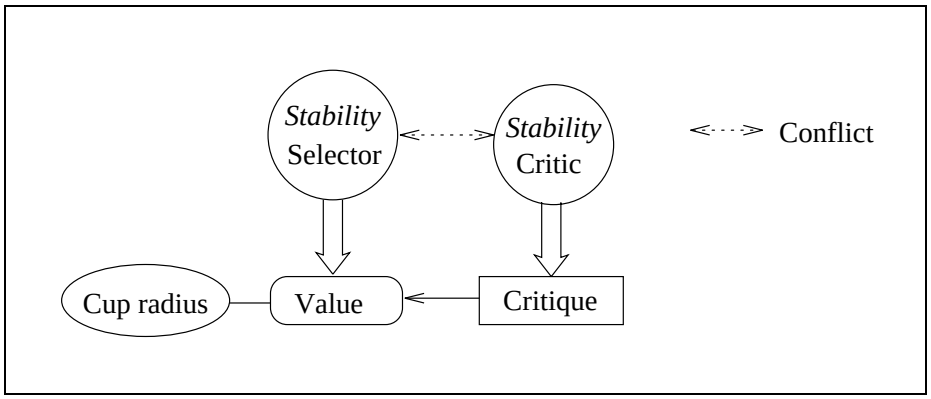

Figure 7.5: Criticism of a value

At some point in the design the cup radius selector sets the value of the cup radius. When the base radius value is also present, the critic of the cup radius value complains because it has the constraint that $|cupradius - baseradius| < 1cm$. The critic asks for an alternative but the selector does not come up with an alternative. Instead it asks for the critic to relax its constraint. The critic relaxes its constraint by modifying the constant on the right hand side of its inequality to 1.5 and the constraint is not violated anymore. The conflict is resolved.

Here is the excerpt:

```
Cup Radius Value Stability Critic: Conflict detected with Cup
Radius Stability Selector.


Cup Radius Value Stability Critic: Criticism asserted. Cup radius
```

```
value is too high from stability pov.

Cup Radius Value Stability Critic: Ask Cup Radius Stability
Selector for an alternative value.

Cup Radius Stability Selector: No alternative values.

Cup Radius Stability Selector: Ask Cup Radius Value Stability
Critic to relax constraints.

Cup Radius Value Stability Critic: Constraint relaxed.

Cup Radius Value Stability Critic: Criticism retracted. Cup
radius value is not too high from stability pov anymore.

Cup Radius Value Stability Critic: Conflict with Cup Radius
Stability Selector resolved.
```

7.6 Selector-Selector Incompatibility Conflict

In this example, two cup radius selectors with different points of view are in conflict. This situation is shown in figure 7.6. The purpose of this example is to show the possibility of a strategy change during negotiation.

The cup radius selector with the style point of view is the current owner of the cup radius value entity, that is, the current value of the cup radius was set by the style selector. When the selector with the stability point of view is able to produce a value, it realizes that the current value is not compatible with the value it has

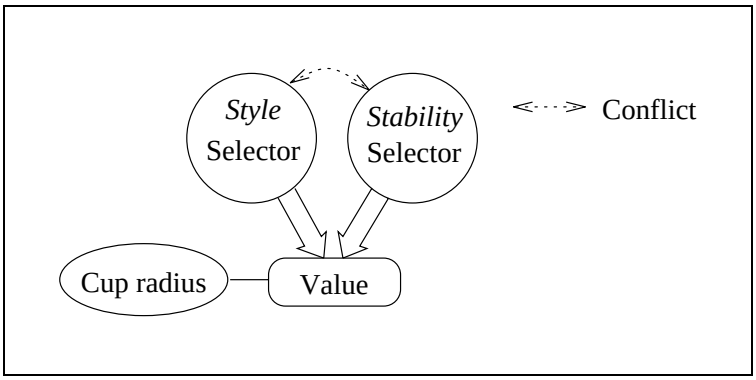

Figure 7.6: Value-value incompatibility

produced, hence a conflict is detected. When asked for an alternative, the style selector would normally provide an alternative value, but in this case, it has run out of all alternatives. It switches its negotiation mode to a *don't care* mode which means that it does not care about the value of the cup radius, so the agent that initiated the conflict can choose any value. The style selector is designed to go into the don't care mode in such situations since the style point of view is not as important as the stability of the cup.

The excerpt follows:

```
Cup Radius Stability Selector: Conflict detected with Cup Radius
Style Selector.

Cup Radius Stability Selector: Ask Cup Radius Style Selector for
an alternative value.

Cup Radius Style Selector: No alternative values.

Cup Radius Style Selector: Switching to dont care mode.
```

```
Cup Radius Style Selector: Tell Cup Radius Stability Selector
that I don't care about value of cup radius.

Cup Radius Stability Selector: Ask Cup Radius Style Selector to
retract the value of cup radius.

Cup Radius Stability Selector: Conflict with Cup Radius Style
Selector resolved.

Cup Radius Style Selector: Value of cup radius retracted.
```

7.7 Summary

The examples presented in this chapter demonstrate the capabilities of COSINE but their main purpose is to give solid examples of the different conflict situations and the negotiation behavior of SiFAs.

The next chapter contains an explanation of the implementation of COSINE. Although the implementation details are not very important to this work, they demonstrate one possible approach to implementing SiFA systems.

Chapter 8

Implementation of COSINE

COSINE is the SiFA based wine glass designer built as part of this work to demonstrate the ideas presented in this thesis. This chapter explains some of the main features of the implementation of this system. The parameters of the design are explained in section 1.5 and shown in figure 1.2.

The cup radius parameter is the focus of all the demonstrations. The entities in the cup radius block and the associated agents are shown in figure 8.1. The points of view of the agents are shown in italics. Besides those shown here, COSINE also has selectors for all other parameters to allow the design of a whole wine glass.

COSINE was developed using CLIPS (C Language Integrated Production System) version 6.02 which is an expert system building tool. Detailed information about CLIPS is available in [Giarratano & Riley 93]. The development platform for COSINE was a DEC Alpha 3000.

8.1 SINE versus COSINE

Instead of extending the SINE platform to include the new extensions to the SiFA paradigm and the negotiation capabilities, COSINE was built from scratch using a slightly different approach. The reason for this was mainly due to time limitations. Developing the wine glass designer using CLIPS directly rather than the SINE plat-

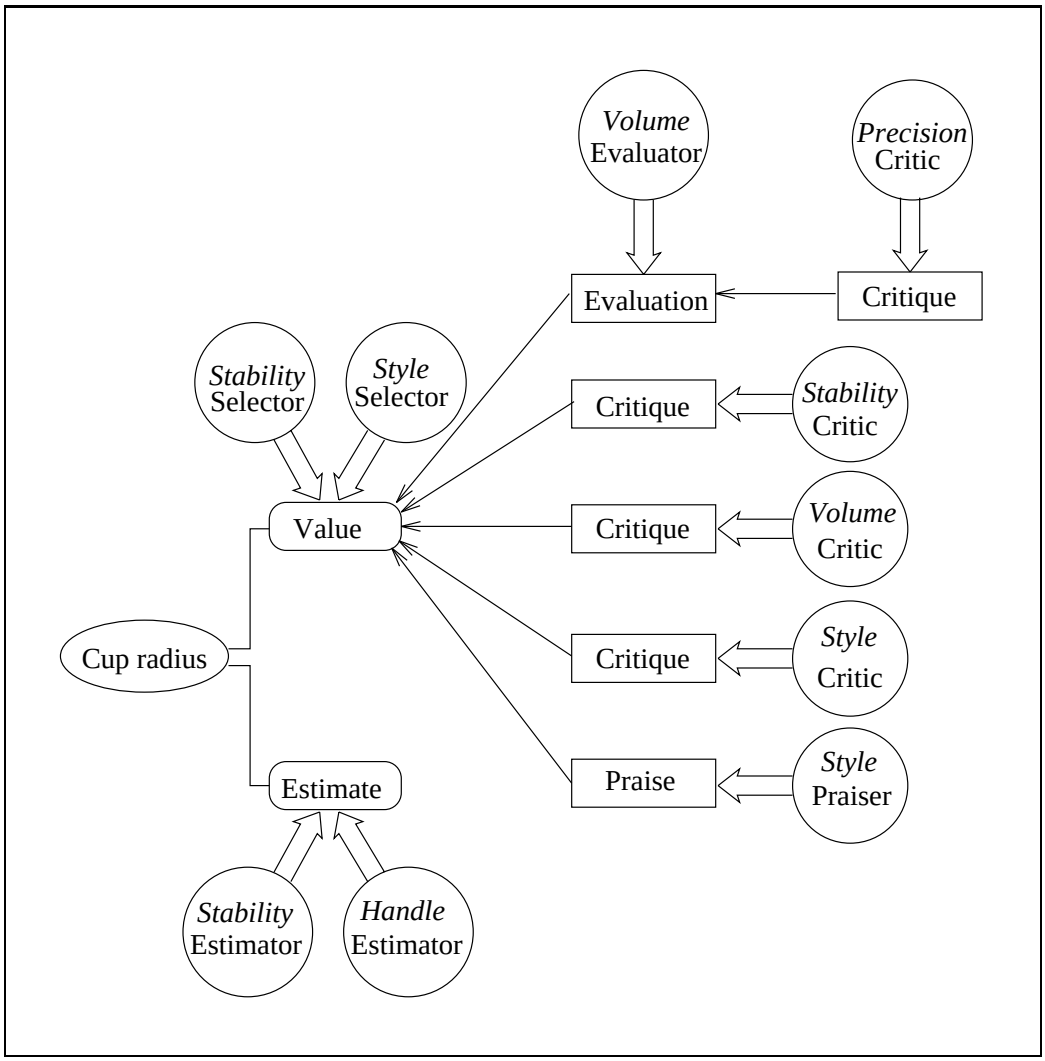

Figure 8.1: Cup radius parameter block in COSINE

form was a simpler task than modifying the SINE code.

SINE used COOL (CLIPS Object-Oriented Language) to define a class hierarchy of agents, design objects, points of view, and targets. This enabled inheriting common features of classes when defining objects. For example, when defining a selector agent, the data and procedures common to all selectors was simply inherited.

COSINE uses only the rule based portion of CLIPS and does not have any classes and objects. While making the system simpler in terms of understandability, this does not allow for inheritance. Although all agents have similar rules, there is no way of inheriting the common code. Many rules are repeated in all agents with minor modification. This is a major drawback.

An advantage of the simplicity of COSINE is that it is easy to spot rule patterns that occur in all agents of the same type. These similar patterns allow for the building of a knowledge acquisition tool so that designers of agents do not need to write rules, but enter the knowledge using a graphical user interface. The knowledge acquisition tool generates rules automatically. Although such a tool has not been built yet, it is part of the ongoing research on SiFA based systems.

Sample rules from COSINE can be found in Appendix A and sample outputs can be found in Appendix B.

8.2 The Agents

The agents in COSINE are a collection of rules. Each agent has rules for design, redesign, conflict detection, and negotiation. Conflict indication, classification, negotiation strategy selection, and refinement are trivial and are included together with the other rules.

Rules in each group have different *salience* to control the sequencing of the various tasks. The salience of a rule is an indication of its importance. When there are

multiple rules with their left hand sides satisfied, the one with the highest salience fires first. In general, design rules have the lowest salience, conflict detection rules have the next level of salience, then come the redesign rules, and the negotiation rules have the highest salience. This way of assigning salience guarantees that while design is going on, if there is a conflict it will be detected before any further design is done. It also assures that after conflict detection the negotiation will start, and that during negotiation no other conflicts will be detected until the current conflict is resolved.

After the rules for each agent have been loaded into CLIPS, the system does not differentiate between agents. So rules from any agent have the potential to fire at any moment. This means that the rules from different agents can work in an interlaced manner, simulating a distributed environment. One agent does not have to wait for another agent to finish its work. It starts working as soon as the preconditions for one of its rules are satisfied.

There is no explicit agenda to coordinate agents. The agenda mechanism built into CLIPS to control rule firings is used. Any time a rule's preconditions are satisfied or there is a modification of the facts on which the rule's preconditions match, the rule has the potential to fire. This is especially suitable for conflict indication and detection. Any time there is a potential conflict, the preconditions of the detection rule are satisfied and the rule fires.

8.3 The Blackboard

The working memory of CLIPS is used to simulate a blackboard. All design decisions are represented as facts and are available to all agents at all times. This means that all entities are stored on the blackboard. The messages are also sent between agents through this blackboard, but they are read only by the agent to which they are addressed.

8.4 Summary

The implementation aspects of COSINE were presented in this chapter in order to demonstrate how a SiFA-based design system might be implemented using a rule based approach.

The next chapter contains a brief evaluation of the work presented in this thesis.

Chapter 9

Evaluation

The evaluation of this work is rather difficult since it is mostly exploratory and the main goal is to extend the SiFA paradigm to include a model of conflicts and negotiations.

The goals of this work were previously identified and expressed in the form of the following questions:

- *Have all conflicts types among all possible pairs of agents been identified and negotiation behaviors proposed for those conflicts?*

  The proposed conflict hierarchy, we believe, covers all meaningful SiFA conflicts. Although negotiation graphs for all conflict types have not been presented, COSINE has demonstration all conflict types and their resolution with simple negotiation strategies. Sample runs of COSINE are in appendix B.

- *Do the classification of conflicts and the corresponding resolution strategies allow inheritance from abstract to concrete classes in the manner proposed by Klein's model?*

  The structure of the conflict hierarchy assures that the nodes higher in the hierarchy have all the general aspects of the nodes below them. The hierarchy is an is-a hierarchy, so a leaf node inherits all aspects of its parent node. The

resolution strategies have not been fully associated with the conflict hierarchy. Most of the negotiation process has been discussed in relation to the hierarchy but the execution of the negotiation strategy has not been linked to the conflict hierarchy.

- *Has the proposed model been successfully implemented?*

  COSINE implements the proposed SiFA model successfully. Sample rules and demonstration runs can be found in the appendices. Although COSINE does not take advantage of the possibility of using inheritance in defining the agents, especially their negotiation knowledge, the implementation is simple enough to allow for automatic generation of the code by a knowledge acquisition tool that provides a graphical front end to the designer of a SiFA system.

- *Has SINE's functionality been increased?*

  It is not easy to evaluate this since the implementation was not based on the SINE platform. COSINE is not a platform to build SiFAs, rather it is a rule based implementation of a SiFA based system. It is possible to replicate most of the functionality of a SINE-based SiFA system using a purely rule based approach like COSINE. The inverse is not true though. The negotiation knowledge built into the COSINE agents are not readily available in SINE agents. Also, the knowledge of the agents in the SINE platform are not as clearly isolated and declarative as they are in COSINE.

- *Does the model, and implementation, allow for the integration of future improvements such as the ability of agents to learn during negotiation?*

  The model does not have any restrictive features, neither does the implementation. The model does not make any claims about learning and can definitely be extended to include it. The implementation would allow history keeping,

and learning simple constants, constraints, preferences, etc, but learning new methods, that are currently implemented as rules, may not be possible.

- *Are the negotiation behaviors of agents understandable to humans?*

  The sample runs of COSINE are very easy to follow. The agents are operating in an interleaved fashion, and it would be possible to have more than one conflict detection and negotiation going on at once if the conflicts did not involve any related entities, i.e., if they were criticism conflicts in different parameter blocks. In the current state of SINE though, one conflict is resolved before a second conflict is detected and its negotiation started. The only reason for this is to make COSINE easier to understand for human experts.

- *Have any patterns of communication among certain pairs of agent types been identified?*

  Since the negotiation behavior of the agents are still very simple, there are very definite observed patterns of communication between the agents. This is a result of the extremely restricted vocabulary and strategies currently available to the agents rather than the patterns that emerge by the nature of the conflicts. Still, it is possible for the same patterns to be observed even if the agents are given more powerful negotiation strategies and a broader vocabulary.

Almost all of the goals set forth for this thesis have been met by this body of work as seen from the answers given to the questions above. It is not possible to show more solid results than the demonstration of the ideas presented in chapter 7 using excerpts from sample runs of COSINE. This is due to the fact that the work on SiFA's is still new and the model keeps evolving. A full implementation of a SiFA-based system that is capable of solving real life design problems does not yet exist, therefore it is not possible to say that SiFA-based systems perform better than others in terms of

any metric. Even if the SiFA paradigm does not prove to be useful in a practical sense, we claim that it is a very interesting research tool that enables the study of all aspects of multi-agent systems, especially conflict management.

The next chapter discusses some of the possible future research directions using the SiFA paradigm.

Chapter 10

Future Research

The SiFA paradigm has proved to be a very good tool to study multi-agent design systems and has revealed many of the underlying issues in building such systems.

There are many directions that require further research in order to understand the full extent of the power and the shortcomings of SiFA-based design systems.

In this chapter, we discuss some of the future research issues that immediately follow from and complement the present work on SiFAs.

We hope that further work on SiFAs will produce more evidence for the applicability of the model proposed in this thesis, probably with further extensions, to design problems from a wide spectrum of domains.

10.1 High Level Entities

The parameter block was introduced in this work without giving a lot of detail as to what exactly the high level entities represent.

Third level entities, where meta-level knowledge about the design is supposed to reside, were not defined in specific terms. It is not very clear what, or in what form, the information should be stored in the third level criticisms, praises, and evaluations in order to help the design process. More work is needed to formalize the contents of the third level entities.

Although the entities with levels of reference higher than three are possible in our model of the parameter block, more research needs to be done concerning these entities. Are these entities meaningful in the context of design? If so, what role do they play and what knowledge can be stored in these entities? Does this knowledge help the design problem solving in any way?

10.2 Negotiation Strategies

Although COSINE successfully resolves the conflicts among its agents and designs wine glasses, the negotiation behavior of the agents, that is, the strategies they use, are very primitive. The simplicity of the negotiation process among the agents is one of the promises of the SiFA paradigm, but still, the strategies used by COSINE agents are over-simplistic.

More work is needed to construct negotiation strategies. It would be very elegant if these strategies were independent of an agent's design knowledge so that they could be "plugged into" any agent. These strategies would probably be dependent on the agent type, and maybe on the level of reference of its target. A library of negotiation strategies for selectors, critics, evaluators, estimators, and praisers on all levels of reference would allow rapid construction of SiFAs with powerful negotiation techniques.

In order to construct various negotiation strategies, the language used among agents needs to be extended beyond the current vocabulary of alternatives and relaxations. Agents need to have more expressive power. Agents probably do not need to understand every other agent, just the ones it can conflict with. The vocabularies of a selector will most likely be very different from the critic of an evaluation of an estimate. The details of the vocabulary have still to be worked out.

10.3 History Keeping

The experience with COSINE has clearly shown that SiFAs need to have history keeping capabilities in order to solve design problems efficiently.

Even very simple history keeping would improve the system drastically. For example, if an agent keeps a record of its proposals that led to a conflict, it can avoid those in the future. There are many other opportunities to improve the system with history keeping.

Since there is no agent hierarchy in the SiFA paradigm, no agent has a global view of the design process. Hence, all history keeping is at an agent level. Agents can remember their past interactions with other agents, but no agent can keep a global history of the whole design process. If an agent hierarchy where some agents control others is introduced in the future, then global history keeping becomes possible.

Keeping history is just a simple way for agents to learn as the design process goes on, and to improve the overall performance of the system with time. There are other opportunities for agents to learn during the design by using various learning techniques as discussed in the next section.

10.4 Learning in SiFA Based Design Systems

Future research on learning using the SiFA paradigm will have two positive results. The first is that adding learning capabilities to SiFAs will drastically improve the performance of SiFA based design systems. Secondly, the SiFA paradigm will act as an exceptional tool to investigate learning in multi-agent design systems in general, not just those with SiFAs. This work will also contribute to distributed learning research.

In the SiFA framework, there are many elements that support learning. These

are:

- Knowledge provided through criticisms, praises, and evaluations;
- Traces of sequences of design decisions;
- Traces of negotiations;
- Analysis of conflict situations.

The things that might be learned in the SiFA framework are:

- Dependencies between design parameters;
- Support in favor of or against a decision;
- Design actions and design rules;
- Preferences in selection tasks;
- Sequences of actions and plans;
- Preconditions and postconditions for rules;
- Predictions of actions of other agents;
- Negotiation strategies;
- Types of conflicts;
- Knowledge about other agents.

Learning should improve the performance of the design system. The aspects of performance that can be improved in the SiFA paradigm are:

- The quality of the final design;

- Making the design process shorter;
- Better conflict anticipation, avoidance, and resolution.

Learning in a system with agents can have two distinct results. The first would be to improve the local performance of agents. The second is an improvement of the overall system through organizational learning where the newly acquired knowledge of an agents does not directly contribute to its local goals but has the effect of improving the overall performance.

10.5 Automatic Generation of SiFAs

The development of COSINE has shown that the agents with the same function have a lot of rules that are almost identical. This is not surprising, but it shows that instead of hand-coding the agents, a graphical knowledge acquisition tool can be built where the designer of a SiFA based design system enters knowledge that is specific to an agent and the rest of the agent is built automatically.

Work in this direction would mainly have practical importance. It would help to build large-scale SiFA systems. There would be some theoretical gain as well since this work would show what is common to groups of agents and help us understand SiFAs better.

10.6 Summary

Although a lot of work has been done to investigate conflicts and negotiation in a design problem using the SiFA paradigm, the possible directions for future research presented in this chapter will provide valuable insights into the design process in general.

The next chapter gives an overall summary of this thesis and provides some conclusions.

Chapter 11

Conclusions

In this chapter, a brief summary of the main ideas presented in the thesis will be given, highlighting the contributions to the field of artificial intelligence in design. The contributions will be followed by some conclusions about the work.

11.1 Main Contributions

The first contribution of this thesis has been to extend the SiFA paradigm with the concept of a parameter block. The parameter block enables knowledge about the design to be presented explicitly, and defines very precisely the knowledge that is required in the design process. It also allows multiple levels of meta-knowledge to be represented, through the idea of levels of reference.

The second main contribution is the analysis of conflicts in the SiFA framework. Since SiFAs provide the building blocks of a multi-agent design system, the conflicts between them represent the primitives of conflict situations in any design.

The conflict hierarchy first divides conflicts into two broad categories: incompatibility conflicts and criticism conflicts. These are further divided according to the kinds of entities involved in the conflict. This hierarchy enables general methods for handling the various tasks involved in the negotiation process, i.e., a method for detecting incompatibility conflicts will work for a whole group of conflicts under the

incompatibility conflicts branch of the hierarchy.

The third major contribution of the thesis is the negotiation model defined for the SiFA paradigm. The negotiation process was divided into many steps, conflict indication, detection, classification, strategy selection, refinement, and execution. Most of these tasks turned out to be relatively simple problems for a SiFA to solve since the kinds of conflicts an agent could be involved in are few, and are very well specified. Once these steps were identified, the negotiation knowledge that should be present in an agent was also identified. Each agent had to have knowledge and methods to be able carry out all of these negotiation steps.

These contributions were also demonstrated by a simple wine glass design system COSINE which used these ideas to come up with a design, taking into account many different and conflicting points of view.

11.2 Discussion

The SiFA-based model for design systems provides a very powerful tool to build and also study design systems.

The model allows a designer to build a system that will consider many different points of view while designing an artifact, but the designer does not have to predict all possible conflicts between these points of view and resolve them while building the design system. The designer needs to provide the design knowledge from every point of view, and the system itself will take care of the conflicts.

This way of building design systems removes the burden of checking rule bases for consistency and also makes maintenance much easier, since the knowledge in different agents does not have to be conflict free. The separation of conflict knowledge from the design knowledge in the agents also has very important consequences in terms of the maintainability and the understandability of a system.

The main disadvantage of the SiFA model from a practical perspective is that the fine grained agent size is less efficient than using larger agents, but the understanding gained by SiFA research will show what functionality should be grouped together to produce more efficient systems.

Apart from the practical issues of having the SiFA model for building design systems, the framework proposed in the thesis also provides a new perspective to study design systems, multi-agent systems, conflicts and negotiations.

SiFAs have proved to be very helpful in gaining insight into many design related research issues and they have revealed new research paths that should be explored further.

Appendix A

Sample Rules From COSINE

This appendix includes sample rules from the source code of COSINE for two selectors, three critics, one evaluator, one estimator, and one praiser, all from the cup radius parameter block for the wine glass design. All of the rules making up these agents are presented here. COSINE has other agents that are not included in this appendix.

The names of the agents as they are referred to in the rules are given in parantheses in the comments after the full name of the agents.

```
;---------------------------------
;cup radius stability selector (S1)
;---------------------------------
;design rules

(defrule S1-cr-from-br
  ?par <- (param (name cr) (owner nil|S1) (val ?cr))
  (param (name br) (val ?br&~nil))
  (pref (agent S1) (list $?list))
  (test (neq (* ?br (first ?list)) ?cr))
  =>
  (bind ?newcr (* ?br (first ?list)))
  (modify ?par (val ?newcr) (owner S1))
  (say S1 Set cup radius value to ?newcr cm.))
```

```
;redesign rules

(defrule S1-re-cr-from-br (declare (salience 20))
  ?par <- (param (name cr) (owner S1) (val ?cr&~nil))
  (param (name br) (val ?br&~nil))
  (pref (agent S1) (list $?list))
  (test (neq (* ?br (first ?list)) ?cr))
  =>
  (bind ?newcr (* ?br (first ?list)))
  (modify ?par (val ?newcr) (owner S1))
  (say S1 Redesigned cup radius value to ?newcr cm.))

;conflict detection rules

(defrule S1-detect (declare (salience 10))
  (param (name cr) (val ?cr&~nil) (owner ?owner&~nil&~S1))
  (param (name br) (val ?br&~nil))
  (pref (agent S1) (list $?list))
  (test (> (/ (abs (- (* ?br (first ?list)) ?cr)) ?cr) 0.1))
  =>
  (assert (conflict (initiator S1) (partner ?owner)))
  (say S1 Conflict detected with (name ?owner) .))

(defrule S1-no-conflict (declare (salience 40))
  (param (name cr) (val ?cr&~nil) (owner ?owner&~nil&~S1))
  (param (name br) (val ?br&~nil))
  (pref (agent S1) (list $?list))
  (test (< (/ (abs (- (* ?br (first ?list)) ?cr)) ?cr) 0.1))
  ?cf <- (conflict (initiator S1) (partner ?owner))
  =>
  (retract ?cf)
  (say S1 Conflict with (name ?owner) resolved.))

;conflict resolution rules

(defrule S1-tell-alternative (declare (salience 30))
  ?msg <- (msg (to S1) (from ?from) (primitive ask)
       (subject alternative) (param1 v1))
  ?pref <- (pref (agent S1) (list $?list))
  (param (name cr))
```

```
  (test (> (length ?list) 1))
  =>
  (modify ?pref (list (rest$ ?list)))
  (say S1 Using alternative value.)
  (retract ?msg))

(defrule S1-no-alternative (declare (salience 30))
  ?msg <- (msg (to S1) (from ?from) (primitive ask)
       (subject alternative) (param1 v1))
  (pref (agent S1) (list $?list))
  ?param <- (param (name cr))
  (test (= (length ?list) 1))
  =>
  (say S1 No alternative values.)
  (assert (msg (from S1) (to ?from) (primitive ask) (subject relax)
       (param1 v1)))
  (say S1 Ask (name ?from) to relax constraints.)
  (retract ?msg))

(defrule S1-ask-alternative (declare (salience 30))
  (conflict (initiator S1) (partner ?partner))
  =>
  (assert (msg (from S1) (to ?partner) (primitive ask)
       (subject alternative) (param1 v1)))
  (say S1 Ask (name ?partner) for an alternative value.))

(defrule S1-dont-care (declare (salience 30))
  ?msg <- (msg (to S1) (from ?from) (primitive tell)
       (subject dont-care) (param1 v1))
  ?conf <- (conflict (initiator S1) (partner ?from))
  =>
  (retract ?msg)
  (retract ?conf)
  (assert (msg (from S1) (to ?from) (primitive ask) (subject retract)
       (param1 v1)))
  (say S1 Ask (name ?from) to retract the value of cup radius.)
  (say S1 Conflict with (name ?from) resolved.))
```

```
;--------------------------------------
;cup radius value stability critic (C1)
;--------------------------------------
;criticism and conflict detection rules

(defrule C1-cup-unstable (declare (salience 10))
  (param (name cr) (val ?cr&~nil) (owner ?owner))
  (param (name br) (val ?br&~nil))
  (pref (agent C1) (list $?list))
  (test (> (- ?cr ?br) (first ?list)))
  =>
  (assert (ct (of v1) (owner C1) (pov stability) (body too-high)))
  (say C1 Conflict detected with (name ?owner) .)
  (assert (conflict (initiator C1) (partner ?owner)))
  (say C1 Criticism asserted.
       Cup radius value is too high from stability pov.))

(defrule C1-cup-stable (declare (salience 40))
  (param (name cr) (val ?cr&~nil) (owner ?owner))
  (param (name br) (val ?br&~nil))
  (pref (agent C1) (list $?list))
  (test (<= (- ?cr ?br) (first ?list)))
  ?ct <- (ct (of v1) (owner C1) (pov stability) (body too-high))
  ?cf <- (conflict (initiator C1) (partner ?owner))
  =>
  (retract ?ct)
  (say C1 Criticism retracted.
       Cup radius value is not too high from stability pov anymore.)
  (retract ?cf)
  (say C1 Conflict with (name ?owner) resolved.))

;conflict resolution rules

(defrule C1-ask-alternative (declare (salience 30))
  (ct (of v1) (owner C1) (body ?tag))
  (param (name cr) (owner ?owner))
  =>
  (assert (msg (from C1) (to ?owner) (primitive ask)
       (subject alternative) (param1 v1)))
  (say C1 Ask (name ?owner) for an alternative value.))
```

```
(defrule C1-relax (declare (salience 30))
  ?msg <- (msg (to C1) (from ?from) (primitive ask) (subject relax)
       (param1 v1))
  ?pref <- (pref (agent C1) (list $?list))
  (test (> (length ?list) 1))
  =>
  (modify ?pref (list (rest$ ?list)))
  (say C1 Constraint relaxed.)
  (retract ?msg))

(defrule C1-no-relaxation (declare (salience 30))
  ?msg <- (msg (to C1) (from ?from) (primitive ask) (subject relax)
       (param1 v1))
  (pref (agent C1) (list $?list))
  (test (= (length ?list) 1))
  =>
  (assert (msg (from C1) (to ?from) (primitive tell) (subject relax)
       (param1 v1) (param2 none)))
  (say C1 Tell (name ?from) that no relaxation is possible.)
  (retract ?msg))

(defrule C1-dont-care (declare (salience 30))
  ?msg <- (msg (to C1) (from ?from) (primitive tell)
       (subject dont-care) (param1 v1))
  ?conf <- (conflict (initiator C1) (partner ?from))
  ?ct <- (ct (of v1) (owner C1))
  =>
  (retract ?msg)
  (assert (msg (from C1) (to ?from) (primitive ask) (subject retract)
       (param1 v1)))
  (say C1 Ask (name ?from) to retract the value of cup radius.)
  (retract ?conf)
  (retract ?ct)
  (say C1 Conflict with (name ?from) resolved.))

;-----------------------------------
;cup radius value volume critic (C2)
;-----------------------------------
;criticism and conflict detection rules
```

```
(defrule C2-cup-too-small (declare (salience 10))
  (param (name cr) (val ?cr&~nil) (owner ?owner))
  (pref (agent C2) (list $?list))
  (test (< (* 0.67 (pi) (* ?cr ?cr ?cr)) (first ?list)))
  =>
  (assert (conflict (initiator C2) (partner ?owner)))
  (say C2 Conflict detected with (name ?owner) .)
  (assert (ct (of v1) (owner C2) (pov volume) (body too-low)))
  (say C2 Criticism asserted. Cup radius value is too low from
       volume pov.))

(defrule C2-cup-too-big (declare (salience 10))
  (param (name cr) (val ?cr&~nil) (owner ?owner))
  (pref (agent C2) (list $?list))
  (test (> (* 0.67 (pi) (* ?cr ?cr ?cr)) (second ?list)))
  =>
  (assert (conflict (initiator C2) (partner ?owner)))
  (say C2 Conflict detected with (name ?owner) .)
  (assert (ct (of v1) (owner C2) (pov volume) (body too-high)))
  (say C2 Criticism asserted. Cup radius value is too high from
       volume pov.))

(defrule C2-cup-size-ok (declare (salience 40))
  (param (name cr) (val ?cr&~nil) (owner ?owner))
  (pref (agent C2) (list $?list))
  (test (>= (* 0.67 (pi) (* ?cr ?cr ?cr)) (first ?list)))
  (test (<= (* 0.67 (pi) (* ?cr ?cr ?cr)) (second ?list)))
  ?ct <- (ct (of v1) (owner C2) (pov volume) (body ?body))
  ?cf <- (conflict (initiator C2) (partner ?owner))
  =>
  (retract ?ct)
  (say C2 Criticism retracted.
       Cup radius value is not ?body anymore from a volume pov. )
  (retract ?cf)
  (say C2 Conflict with (name ?owner) resolved.))

;conflict resolution rules

(defrule C2-ask-alternative (declare (salience 30))
```

```
  (ct (of v1) (owner C2) (body ?tag))
  (param (name cr) (owner ?owner))
  =>
  (assert (msg (from C2) (to ?owner) (primitive ask)
       (subject alternative) (param1 v1)))
  (say C2 Ask (name ?owner) for an alternative value.))

(defrule C2-no-relaxation (declare (salience 30))
  ?msg <- (msg (to C2) (from ?from) (primitive ask) (subject relax)
       (param1 v1))
  =>
  (assert (msg (from C1) (to ?from) (primitive tell) (subject relax)
       (param1 v1) (param2 none)))
  (say C1 Tell (name ?from) that no relaxation is possible.)
  (retract ?msg))

(defrule C2-dont-care (declare (salience 30))
  ?msg <- (msg (to C2) (from ?from) (primitive tell)
       (subject dont-care) (param1 v1))
  ?conf <- (conflict (initiator C2) (partner ?from))
  ?ct <- (ct (of v1) (owner C2))
  =>
  (retract ?msg)
  (assert (msg (from C2) (to ?from) (primitive ask) (subject retract)
       (param1 v1)))
  (say C2 Ask (name ?from) to retract the value of cup radius.)
  (retract ?conf)
  (retract ?ct)
  (say C2 Conflict with (name ?from) resolved.))

;------------------------------
;cup radius style selector (S2)
;------------------------------
;design rules

(defrule S2-default-cr-style
  ?par <- (param (name cr) (val nil))
  (pref (agent S2) (list $?list))
  =>
  (modify ?par (val (first ?list)) (owner S2))
```

```
  (say S2 Cup radius set to default value of (first ?list) cm.))

;redesign rules

(defrule S2-new-cr-style (declare (salience 20))
  ?par <- (param (name cr) (owner S2) (val ?cr&~nil))
  (pref (agent S2) (list $?list))
  (test (neq ?cr (first ?list)))
  =>
  (modify ?par (val (first ?list)) (owner S2))
  (say S2 Cup radius set to new value of (first ?list) cm.))

;conflict resolution rules

(defrule S2-dont-care (declare (salience 30))
  ?care <- (S2 dont care)
  ?msg <- (msg (to S2) (from ?from) (primitive ask)
       (subject alternative) (param1 v1))
  =>
  (retract ?care)
  (retract ?msg)
  (assert (msg (from S2) (to ?from) (primitive tell)
       (subject dont-care) (param1 v1)))
  (say S2 Tell (name ?from) that I don't care about value of
       cup radius.))

(defrule S2-retract (declare (salience 30))
  ?msg <- (msg (to S2) (from ?from) (primitive ask) (subject retract)
       (param1 v1))
  ?par <- (param (name cr) (owner S2))
  ?pref <- (pref (agent S2))
  =>
  (retract ?msg)
  (modify ?par (val nil) (owner nil))
  (retract ?pref)
  (say S2 Value of cup radius retracted.))

(defrule S2-tell-alternative (declare (salience 30))
  ?msg <- (msg (to S2) (from ?from) (primitive ask)
       (subject alternative) (param1 v1))
```

```
  ?pref <- (pref (agent S2) (list $?list))
  (param (name cr))
  (test (> (length ?list) 1))
  =>
  (retract ?msg)
  (modify ?pref (list (rest$ ?list)))
  (say S2 Using alternative value.))

(defrule S2-no-alternative (declare (salience 30))
  (msg (to S2) (from ?from) (primitive ask)
       (subject alternative) (param1 v1))
  (pref (agent S2) (list $?list))
  ?param <- (param (name cr))
  (test (= (length ?list) 1))
  =>
  (say S2 No alternative values.)
  (say S2 Switching to dont care mode.)
  (assert (S2 dont care)))

;---------------------------------------
;cup radius value volume evaluator (Ev1)
;---------------------------------------
;evaluation rules

(defrule Ev1-rough-evaluate-good
  (param (name cr) (val ?cr&~nil))
  (pref (agent Ev1) (list $?list))
  ?eval <- (eval (of v1) (op ?op) (percentage ?per))
  (test (< (- (* 0.67 (pi) ?cr ?cr ?cr) (first ?list)) (second ?list)))
  (test (or (neq ?per good) (neq ?op eq)))
  (not (precision evaluation required))
  =>
  (modify ?eval (of v1) (owner Ev1) (pov volume) (op eq)
  (percentage good))
  (say Ev1 Evaluation asserted. Cup radius value is good from a
       volume pov.))

(defrule Ev1-rough-evaluate-bad
  (param (name cr) (val ?cr&~nil))
  (pref (agent Ev1) (list $?list))
```

```
  ?eval <- (eval (of v1) (op ?op) (percentage ?per))
  (test (> (- (* 0.67 (pi) ?cr ?cr ?cr) (first ?list)) (second ?list)))
  (test (or (neq ?per bad) (neq ?op eq)))
  (not (precision evaluation required))
  =>
  (modify ?eval (of v1) (owner Ev1) (pov volume) (op eq)
  (percentage bad))
  (say Ev1 Evaluation asserted. Cup radius value is bad from a
       volume pov.))

(defrule Ev1-precise-evaluate
  ?req <- (precision evaluation required)
  (param (name cr) (val ?cr&~nil))
  (pref (agent Ev1) (list $?list))
  ?eval <- (eval (of v1) (owner Ev1) (pov volume) (op ?op)
 (percentage ?oldper))
  =>
  (bind ?per (round (- 100 (* (/ (abs (- (* 0.67 (pi) ?cr ?cr ?cr)
   (first ?list))) (first ?list)) 100))))
  (if (neq ?oldper ?per) then
    (modify ?eval (op eq) (percentage ?per))
    (say Ev1 Evaluation asserted. Cup radius value's quality is
 ?per percent.)))

;conflict resolution rules

(defrule Ev1-precision-required (declare (salience 30))
  ?msg <- (msg (to Ev1) (from ?from) (primitive ask)
       (subject precision) (param1 ev1))
  =>
  (retract ?msg)
  (assert (precision evaluation required))
  (say Ev1 Changing to precision evaluation mode.))

;-------------------------------------------------
;cup radius value evaluation precision critic (C3)
;-------------------------------------------------
;criticism and conflict detection rules

(defrule C3-imprecise-evaluation (declare (salience 10))
```

```
  (eval (of v1) (owner ?owner&~nil) (percentage ?per))
  (test (not (numberp ?per)))
  =>
  (assert (conflict (initiator C3) (partner ?owner)))
  (say C3 Conflict detected with (name ?owner) .)
  (assert (ct (of ev1) (owner C3) (pov precision)
      (body too-imprecise)))
  (say C3 Criticism asserted. Cup radius value evaluation is
       too imprecise.))

(defrule C3-precise-evaluation (declare (salience 40))
  (eval (of v1) (owner ?owner&~nil) (percentage ?per))
  (test (numberp ?per))
  ?ct <- (ct (of ev1) (owner C3) (pov precision)
     (body too-imprecise))
  ?cf <- (conflict (initiator C3) (partner ?owner))
  =>
  (retract ?ct)
  (say C3 Criticism retracted.
       Cup radius value evaluation is not imprecise anymore.)
  (retract ?cf)
  (say C3 Conflict with (name ?owner) resolved.))

;conflict resolution rules

(defrule C3-ask-precision (declare (salience 30))
  (ct (of ev1) (owner C3) (body too-imprecise))
  (eval (of v1) (owner ?owner) )
  =>
  (assert (msg (to ?owner) (from C3) (primitive ask)
       (subject precision) (param1 ev1)))
  (say C3 Ask (name ?owner) to offer a more precise evaluation.))

;---------------------------------------
;cup radius handleability estimator (Es1)
;---------------------------------------
;estimation rules

(defrule Es1-using-stem-length
  ?par <- (param (name cr) (est nil))
```

```
  (param (name sl) (val ?sl&~nil))
  (pref (agent Es1) (list $?list))
  =>
  (bind ?est (* (first ?list) ?sl))
  (modify ?par (est ?est) (estowner Es1))
  (say Es1 Cup radius estimated to be ?est cm.))

;reestimation rules

(defrule Es1-reusing-stem-length (declare (salience 20))
  ?par <- (param (name cr) (est ?est&~nil) (estowner Es1))
  (param (name sl) (val ?sl&~nil))
  (pref (agent Es1) (list $?list))
  (test (neq ?est (* (first ?list) ?sl)))
  =>
  (bind ?est (* (first ?list) ?sl))
  (modify ?par (est ?est) (estowner Es1))
  (say Es1 Cup radius re-estimated to be ?est cm.))

;conflict detection rules

(defrule Es1-detect (declare (salience 10))
  (param (name cr) (est ?cr&~nil) (estowner ?owner&~nil&~Es1))
  (param (name sl) (val ?sl&~nil))
  (pref (agent Es1) (list $?list))
  (test (> (/ (abs (- (* ?sl (first ?list)) ?cr)) ?cr) 0.1))
  =>
  (assert (conflict (initiator Es1) (partner ?owner)))
  (say Es1 Conflict detected with (name ?owner) .))

(defrule Es1-no-conflict (declare (salience 40))
  (param (name cr) (est ?cr&~nil) (estowner ?owner&~nil&~Es1))
  (param (name sl) (val ?sl&~nil))
  (pref (agent Es1) (list $?list))
  (test (< (/ (abs (- (* ?sl (first ?list)) ?cr)) ?cr) 0.1))
  ?cf <- (conflict (initiator Es1) (partner ?owner))
  =>
  (retract ?cf)
  (say Es1 Conflict with (name ?owner) resolved.))
```

```
;conflict resolution rules

(defrule Es1-ask-alternative (declare (salience 30))
  (conflict (initiator Es1) (partner ?partner))
  =>
  (assert (msg (from Es1) (to ?partner) (primitive ask)
       (subject alternative) (param1 es1)))
  (say Es1 Ask (name ?partner) for an alternative estimate.))

(defrule Es1-tell-alternative (declare (salience 30))
  ?msg <- (msg (to Es1) (from ?from) (primitive ask)
       (subject alternative) (param1 es1))
  ?pref <- (pref (agent Es1) (list $?list))
  (test (> (length ?list) 1))
  =>
  (retract ?msg)
  (modify ?pref (list (rest$ ?list)))
  (say Es1 Using alternative estimate.))

(defrule Es1-no-alternative (declare (salience 30))
  ?msg <- (msg (to Es1) (from ?from) (primitive ask)
       (subject alternative) (param1 es1))
  (pref (agent Es1) (list $?list))
  (test (= (length ?list) 1))
  =>
  (retract ?msg)
  (say Es1 No alternative estimates.))

;-----------------------------------
;cup radius value style praiser (P1)
;-----------------------------------
;praise rules

(defrule P1-cup-beautiful (declare (salience 10))
  (param (name cr) (val ?cr&~nil) (owner ?owner))
  (pref (agent P1) (list $?list))
  (test (and (> ?cr (first ?list)) (< ?cr (second ?list))))
  =>
  (assert (pr (of v1) (owner P1) (pov style) (body beautiful)))
  (say P1 Praise asserted. Cup radius value makes cup beautiful
```

```
       from style pov.))

(defrule P1-cup-beautiful-no-more (declare (salience 10))
  (param (name cr) (val ?cr&~nil) (owner ?owner))
  (pref (agent P1) (list $?list))
  (test (or (< ?cr (first ?list)) (> ?cr (second ?list))))
  ?pr <- (pr (of v1) (owner P1))
  =>
  (retract ?pr)
  (say P1 Praise retracted. Cup radius value does not make the cup
       beautiful from style pov anymore.))

;conflict detection rules

(defrule P1-detect (declare (salience 10))
  (pr (of v1) (owner P1))
  (ct (of v1) (owner ?owner) (pov style))
  =>
  (say P1 Conflict detected with (name ?owner) .)
  (assert (conflict (initiator P1) (partner ?owner))))

(defrule P1-no-conflict (declare (salience 40))
  ?cf <- (conflict (initiator P1) (partner ?partner))
  (not (ct (of v1) (owner ?partner) (pov style)))
  =>
  (retract ?cf)
  (say P1 Conflict with (name ?partner) resolved.))

;conflict resolution rules

(defrule P1-ask-removal (declare (salience 30))
  (conflict (initiator P1) (partner ?partner))
  =>
  (assert (msg (from P1) (to ?partner) (primitive ask)
       (subject retract) (param1 ct)))
  (say P1 Ask (name ?partner) to remove its criticism.))
```

Appendix B

Sample Output From COSINE

This appendix contains a sample run of COSINE showing examples of some of the possible conflict situations and how they are resolved through negotioation.

First the `(batch "wine.bat")` command is entered at the `CLIPS>` prompt. This loads in all the necessary files for the wine glass design. Then, the design process is started with the `(design)` command.

```
CLIPS> (batch "wine.bat")
CLIPS> (design)

Base Thickness Style Selector: Set base thickness to default
value of 0.3 cm.

Cup Thickness Style Selector: Set cup thickness to default value
of 0.1 cm.

Cup Radius Style Selector: Cup radius set to default value of
3 cm.

Cup Radius Value Volume Critic: Conflict detected with Cup Radius
Style Selector.

Cup Radius Value Volume Critic: Criticism asserted. Cup radius
value is too high from volume pov.
```

```
Cup Radius Value Volume Critic: Ask Cup Radius Style Selector for
an alternative value.

Cup Radius Style Selector: Using alternative value.

Cup Radius Style Selector: Cup radius set to new value of 2.5 cm.

Cup Radius Value Volume Critic: Ask Cup Radius Style Selector for
an alternative value.

Cup Radius Style Selector: Using alternative value.

Cup Radius Style Selector: Cup radius set to new value of 2.0 cm.

Cup Radius Value Volume Critic: Criticism retracted. Cup radius
value is not too-high anymore from a volume pov.

Cup Radius Value Volume Critic: Conflict with Cup Radius Style
Selector resolved.

Cup Radius Value Style Critic: Criticism asserted. Cup radius
value makes cup ugly from style pov.

Cup Radius Value Style Praiser: Praise asserted. Cup radius value
makes cup beautiful from style pov.

Cup Radius Value Style Praiser: Conflict detected with Cup Radius
Value Style Critic.

Cup Radius Value Style Praiser: Ask Cup Radius Value Style Critic
to remove its criticism.

Cup Radius Value Style Critic: Criticism retracted.

Cup Radius Value Style Praiser: Conflict with Cup Radius Value
Style Critic resolved.

Stem Length Stability Selector: Set stem length to default value
of 4 cm.
```

```
Base Radius Stability Selector: Set base radius to 1.0 cm.

Cup Radius Stability Selector: Conflict detected with Cup Radius
Style Selector.

Cup Radius Stability Selector: Ask Cup Radius Style Selector for
an alternative value.

Cup Radius Style Selector: No alternative values.

Cup Radius Style Selector: Switching to dont care mode.

Cup Radius Style Selector: Tell Cup Radius Stability Selector
that I don't care about value of cup radius.

Cup Radius Stability Selector: Ask Cup Radius Style Selector to
retract the value of cup radius.

Cup Radius Stability Selector: Conflict with Cup Radius Style
Selector resolved.

Cup Radius Style Selector: Value of cup radius retracted.

Stem Radius Strength Selector: Set stem radius value to 0.4 cm.

Stem Radius Strength Selector: Conflict detected with Stem Length
Stability Selector.

Stem Radius Strength Selector: Ask Stem Length Stability Selector
for an alternative value.

Stem Length Stability Selector: Using alternative value.

Stem Length Stability Selector: Redesigned stem length value to
3 cm.

Stem Radius Strength Selector: Conflict with Stem Length
Stability Selector resolved.

Stem Radius Strength Selector: Redesigned stem radius value to
```

```
0.3 cm.

Cup Radius Stability Estimator: Cup radius estimated to be
2.0 cm.

Cup Radius Handleability Estimator: Conflict detected with Cup
Radius Stability Estimator.

Cup Radius Handleability Estimator: Ask Cup Radius Stability
Estimator for an alternative estimate.

Cup Radius Stability Estimator: Using alternative estimate.

Cup Radius Stability Estimator: Cup radius re-estimated to be
1.5 cm.

Cup Radius Handleability Estimator: Conflict with Cup Radius
Stability Estimator resolved.

Cup Radius Stability Selector: Set cup radius value to 3.0 cm.

Cup Radius Value Volume Critic: Conflict detected with Cup Radius
Stability Selector.

Cup Radius Value Volume Critic: Criticism asserted. Cup radius
value is too high from volume pov.

Cup Radius Value Volume Critic: Ask Cup Radius Stability Selector
for an alternative value.

Cup Radius Stability Selector: Using alternative value.

Cup Radius Stability Selector: Redesigned cup radius value to
2.3 cm.

Cup Radius Value Volume Critic: Ask Cup Radius Stability Selector
for an alternative value.

Cup Radius Stability Selector: Using alternative value.
```

```
Cup Radius Stability Selector: Redesigned cup radius value to
2.2 cm.

Cup Radius Value Volume Critic: Criticism retracted. Cup radius
value is not too-high anymore from a volume pov.

Cup Radius Value Volume Critic: Conflict with Cup Radius
Stability Selector resolved.

Cup Radius Value Style Critic: Criticism asserted. Cup radius
value makes cup ugly from style pov.

Cup Radius Value Style Praiser: Conflict detected with Cup Radius
Value Style Critic.

Cup Radius Value Style Praiser: Ask Cup Radius Value Style Critic
to remove its criticism.

Cup Radius Value Style Critic: Criticism retracted.

Cup Radius Value Style Praiser: Conflict with Cup Radius Value
Style Critic resolved.

Cup Radius Value Style Praiser: Praise asserted. Cup radius value
makes cup beautiful from style pov.

Cup Radius Value Stability Critic: Conflict detected with Cup
Radius Stability Selector.

Cup Radius Value Stability Critic: Criticism asserted. Cup radius
value is too high from stability pov.

Cup Radius Value Stability Critic: Ask Cup Radius Stability
Selector for an alternative value.

Cup Radius Stability Selector: No alternative values.

Cup Radius Stability Selector: Ask Cup Radius Value Stability
Critic to relax constraints.
```

```
Cup Radius Value Stability Critic: Constraint relaxed.

Cup Radius Value Stability Critic: Criticism retracted. Cup
radius value is not too high from stability pov anymore.

Cup Radius Value Stability Critic: Conflict with Cup Radius
Stability Selector resolved.

Cup Radius Value Volume Evaluator: Evaluation asserted. Cup
radius value is good from a volume pov.

Cup Radius Value Evaluation Precision Critic: Conflict detected
with Cup Radius Value Volume Evaluator.

Cup Radius Value Evaluation Precision Critic: Criticism
asserted. Cup radius value evaluation is too imprecise.

Cup Radius Value Evaluation Precision Critic: Ask Cup Radius
Value Volume Evaluator to offer a more precise evaluation.

Cup Radius Value Volume Evaluator: Changing to precision
evaluation mode.

Cup Radius Value Volume Evaluator: Evaluation asserted. Cup
radius value's quality is 88 percent.

Cup Radius Value Evaluation Precision Critic: Criticism
retracted. Cup radius value evaluation is not imprecise anymore.

Cup Radius Value Evaluation Precision Critic: Conflict with Cup
Radius Value Volume Evaluator resolved.

Here is the final design:

Cup radius: 2.2 cm.
Cup thickness: 0.1 cm.
Base radius: 1.0 cm.
Base thickness: 0.3 cm.
Stem length: 3 cm.
Stem radius: 0.3 cm.
```

References

[Austin 62] J. L. Austin. *How to Do Things With Words*. Oxford University Press, Oxford, England, 1962.

[Bond & Gasser 88] A. H. Bond & Les Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1988.

[Brown & Chandrasekaran 92] D. C. Brown and B. Chandrasekaran. Investigating Routine Problem Solving. Christopher Tong & Duvvuru Sriram (Eds.) *Artificial Intelligence in Engineering Design*, Volume 1, Academic Press, Inc., 1992, pp.221-249.

[Brown et al. 94] D. C. Brown, B. V. Dunskus & D. L. Grecu. Using Single Function Agents to Investigate Negotiation. *AAAI Workshop on Models of Conflict Management in Cooperative Problem Solving*, 1994.

[Douglas et al. 93] R. E. Douglas, D. C. Brown & D. C. Zenger. A Concurrent Engineering Demonstration and Training System for Engineers and Managers. *International Journal of CADCAM and Computer Graphics, special is-*

sue on "AI and Computer Graphics", I. Costea (Ed.), Hermes, Vol. 8, No. 3, 1993, pp.263-301.

[Dunskus 94] Bertram V. Dunskus. *Single Function Agents and Their Negotiation Behavior in Expert Systems*. Master's Thesis, Worcester Polytechnic Institute, Worcester, MA, 1994.

[Giarratano & Riley 93] J. C. Giarratano & G. Riley. *CLIPS Reference Manual*. Volumes 1 & 2. NASA Lyndon B. Johnson Space Center Information Center Information Systems Directorate, Software Technology Branch, Version 6.0, June 2nd 1993.

[Goodwin 93] R. Goodwin. *Formalizing properties of agents*. Technical Report CMU-CS-93-159, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[Huhns 87] M. N. Huhns (ed.) *Distributed Artificial Intelligence*, Morgan Kaufmann, 1987.

[Klein 91] M. Klein. Supporting Conflict Resolution in Cooperative Design Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 6, November/December 1991, pp. 1379-1390.

[Klein & Lu 90] M. Klein & S. C.-Y. Lu. Conflict Resolution in Cooperative Design. *The International Journal for AI in Engineering*. No.4, 1990, pp. 168-180.

[Klein & Lu 91] M. Klein & S. C.-Y. Lu. Insights into Cooperative Group Design: Experience with the LAN Designer System. G. Rzevski & R. A. Adey (Eds.), Proceedings of the 6th International AI in Engineering Conference, Oxford, UK, Elsevier, July 1991, pp. 143-162.

[Finin et al.] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, J. McGuire, D. McKay, S. Shapiro, R. Pelavin, and C. Beck. *Specification of the KQML Agent Communication Language*, Official Document of the DARPA Knowledge Sharing Initiative's External Interfaces Working Group, 1993.

[Lander & Lesser 91] S. E. Lander & V. R. Lesser. Customizing Distributed Search Among Agents with Heterogeneous Knowledge. *Proceedings of the 5th International Symposium on AI Applications in Manufacturing & Robotics*, Cancun, Mexico, December 1991.

[Polat et al. 93] F. Polat, Shashi Shekar, & H. A. Guvenir. A Negotiation Platform for Cooperating Multi-agent Systems. *Concurrent Engineering: Research and Applications*, Vol. 1, No. 3, Academic Press, September 1993, pp. 179-187.

[Searle 69] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, England, 1969.

[Sycara 87] K. P. Sycara. Finding Creative Solutions in Adversarial Impasses. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, 1987, pp. 997-1003.

[Sycara 88] K. P. Sycara. Resolving Goal Conflicts via Negotiation. *Proceedings of the AAAI-88*, Volume 1. St. Paul, MN, 1988, pp. 245-250.

[Sycara 90] K. P. Sycara. Cooperative Negotiation in Concurrent Engineering Design. *Cooperative Engineering Design*, Springer Verlag Publications, 1990, pp. 269-297.

[Shoham 90] Y. Shoham. *Agent-Oriented Programming*. Technical Report STAN-CS-1335-90, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[Talukdar & Cardozo 88] S. Talukdar & E. Cardozo. Building Large-Scale Software Organizations. (Ed.) M. D. Rychener, *Expert Systems for Engineering Design*, Academic Press, Inc., 1988, pp. 241-256.

[Victor et al. 93] 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. *Applications of AI in Engineering VIII. Vol. 1: Design, Methods and Techniques*. G. Rzevski, J. Pastor & R. A. Adey (Eds.), Proceedings of the 8th International

AI in Engineering Conference, Toulouse, France. Elsevier, June 1993, pp. 83-96.

[Victor & Brown 94] 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. Russel (Eds.), Proceedings of the 9th International AI in Engineering Conference, Pennsylvania, USA. Computational Mechanics Publications, July 1994, pp. 173-179.

[Werkman & Barone 91] K. J. Werkman & M. Barone. Evaluating Alternative Connection Designs Through Multiagent Negotiation. D. Sriram, R. Logcher, S. Fukuda (Eds.), *Computer Aided Cooperative Product Development*. Lecture Notes Series, No. 492, Springer Verlag, 1992, pp. 298-333.

[Wooldridge & Jennings 94] Michael Wooldridge & Nicholas R. Jennings. Intelligent Agents: Theory and Practice. Submitted to *Knowledge Engineering Review*, 1994.