DISTRIBUTED SEMANTIC NETWORKS AND CLIPS

James Snyder and Tony Rodriguez

CAD Research Unit, Design Institute
California Polytechnic State University, San Luis Obispo

Abstract. Semantic networks of frames are commonly used as a method of organizing and reasoning in many types of problems. In most of these applications the semantic network exists as a single entity in a single process environment. Advances in workstation hardware provide support for more sophisticated applications involving multiple processes, interacting in a distributed environment. In these applications the semantic network may well be distributed over several concurrently executing tasks.

This paper describes the design and implementation of a frame-based, distributed semantic network in which frames are accessed both through CLIPS expert systems and procedural C++ language programs. The application area is a knowledge-based, cooperative decision making model utilizing both rule-based and procedural experts.

INTRODUCTION

Currently, the CAD Research Unit is developing an Intelligent Computer Aided Design System (ICADS). The purpose of ICADS is to provide an intelligent environment for cooperative computer-based problem solving under the explicit control of the user using architectural design as a test environment. ICADS allows a group of distributed, intelligent agents to converse about a problem larger than any single agent's domain of expertise or knowledge, and provides advice and suggestions to the user as to the state and compliance of the current problem solution.

From a high-level point of view, ICADS is composed of three major pieces: a blackboard, a group of procedural and CLIPS-based expert systems referred to as Intelligent Design Tools (IDTs), and a semantic network of frames. The ICADS components run as distributed processes in a computer network.

To facilitate distributed expert system execution, a communication framework was developed which allows CLIPS expert systems to assert facts to another expert system under the control of a blackboard (Taylor 1990, Taylor and Myers 1990). In addition to controlling the IDT communication, the blackboard has a conflict resolver which arbitrates between IDTs. The conflict resolver establishes the system accepted values by evaluating suggestions made by IDTs and is written in CLIPS. The current working model of ICADS has six CLIPS based expert-systems. Each expert is controlled by the blackboard using the communication framework described above and are in the following architectural domains: cost, access, lighting, thermal, acoustics, and structure (Pohl 1989).

To allow C++ programs to participate in the current problem solution, they have to communicate with the blackboard and have access to the semantic network of frames at the same

level of representation as the CLIPS experts. The remainder of this paper will describe the implementation of the distributed semantic network in the CLIPS and C++ environments and illustrate sample uses of the C++ implementation used in conjunction with CLIPS-based experts.

THE CLIPS FRAME REPRESENTATION

Within the CLIPS environment, the frame-based representation used in ICADS is implemented as a set of CLIPS facts. A frame is a collection of information about a class or object. The information is represented in CLIPS with a frame header fact and any number of slot facts. Slots can define a particular value of the class or identify a relation to another class. In terms of a node-link data structure, the frame is a node and a relation is a link (Barr and Feigenbaum 1981). It is important to note that this representation does not require any modifications to CLIPS--it uses only CLIPS facts (Assal and Myers 1990). Unlike procedural paradigms, it is not necessary to locate an entire frame when a piece of information is needed. Only the pertinent slots are necessary and are accessed directly through CLIPS pattern matching. A frame is represented by a set of facts that have one or more common fields to connect them together. Each fact has a keyword in the first field to indicate the type of information it represents. The keywords are: FRAME, RELATION, VALUE. The second field has the class name which is used to connect all the instances of this class or establish a relation with another class.

CLIPS Frame Definition

Figure 1 illustrates the CLIPS fact format of a frame header. The *class* field defines the class to which the frame belongs. The *instance* field uniquely identifies the frame. Having the frame header in a CLIPS left-hand-side pattern is not always necessary, but it is useful in performing operations on the whole frame; displaying and deleting are example operations.

```
(FRAME <class> <instance>)
```

Figure 1 - Frame Fact Format

CLIPS Value Slot Definition

Figure 2 shows the CLIPS fact format of a value slot. This slot provides the values for particular attributes of a frame. It is important to note that these values are multifield values and can contain mixed types of data. Note also that the instance of the frame in contained within the value slot fact. This allows for direct pattern matching of the frame attribute values.

```
(VALUE <class> <attribute> <instance> <values>)
```

Figure 2 - Value Slot Fact Format

CLIPS Relation Slot Definition

Figure 3 illustrates the CLIPS fact format of a relation slot. This relation slot represents a **has-a** relationship. The *class1* and *instance1* fields indicate the owning class. The *class2* and *instance2* fields represent the frame instance which is pointed to.

```
(RELATION <class1> <class2> <instance1> <instance2>)
```

Figure 3 - Relation Slot Fact Format

An Example CLIPS Frame

An example architectural object is a room or space. Figure 4 contains an example space frame with several values and relations. The relations in the example indicate that the LOBBY space has four walls. If a wall of a particular space is to be referenced by a rule, a pattern similar to the example would be used.

```
(FRAME space 15)
(VALUE space name 15 LOBBY)
(VALUE space perimeter 15 108)
(RELATION space wall 15 1)
(RELATION space wall 15 2)
(RELATION space wall 15 3)
(RELATION space wall 15 4)
```

Figure 4 - An Example Architectural Frame

An Example Rule Using a CLIPS Frame

Suppose the building code states that the area of all bathrooms must be greater than or equal to twenty square feet. The rule in Figure 5 checks to see if the area of a bathroom is less than twenty square feet. If the area of the space is too small an error message is printed. This simple example illustrates how values slots can be used. Relation slots are used in a simpler manner; the type of relation is included in the left-hand-side pattern and can be used to reason about groups of objects at the same time. For example, a rule could specify a pattern which references all the walls which belong to the BATHROOM space.

```
(defrule building-code-check
  (FRAME space ?id)
  (VALUE space name ?id BATHROOM)
  (VALUE space area ?id ?x&:(< ?x 20))
 =>
  (printout t "The bathroom is too small" crlf)
)
```

Figure 5 - An Example Rule Using a Frame

C++ IMPLEMENTATION OF FRAMES

The purpose of the frame classes is to provide an object-oriented representation for the frame facts that are used by the CLIPS IDTs. In order to allow C++ programs to work with the current semantic network that is represented in CLIPS, a representation paralleling the CLIPS frame data structures was implemented using C++ objects. Using the class method interface, frames can be built explicitly and then added to the network; the user is responsible for creating any necessary objects to insert into the frame--relations and values are examples. After the frame has been built correctly it is then entered into the semantic network using net class

methods. The following sections explain the structure of the C++ classes used to represent a semantic network of frames.

The Net Class

The net class is basically a container class that represents the entire semantic network. It is composed of frame objects. The net class provides several methods for the addition, removal, and modification of frames, as well as, query methods that allow questions to be asked about the network. Figure 6 shows the class structure of the net class. The *frames* data member is a dictionary of frame objects. A dictionary is an associative array where a name is mapped to an actual instance of an object. In this case, the frame instance name is mapped to a frame instance C++ object.

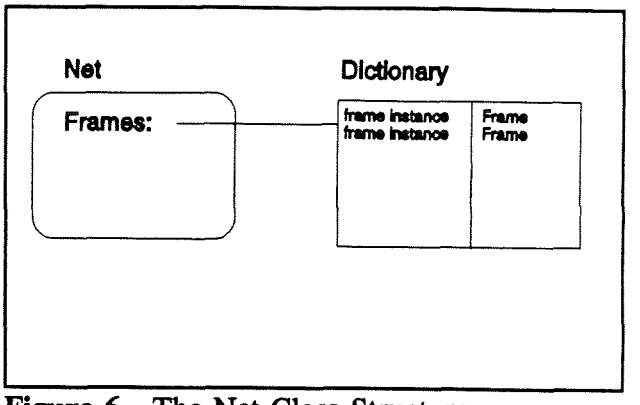

Figure 6 - The Net Class Structure

The Frame Class

The frame class is the central component of the semantic network. A frame is uniquely identified by its name and instance number and contains value and relation slots illustrated by Figure 7. The frame class has several methods that permit the addition, and deletion of value and relation slots, in addition to methods that return information about the frame itself.

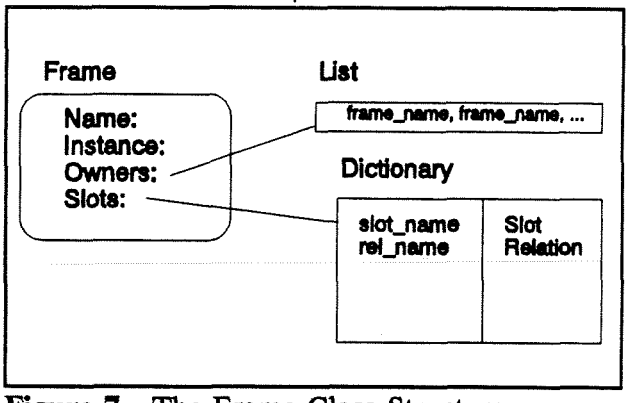

Figure 7 - The Frame Class Structure

The Slot Class

The slot class is an abstract class and is used to derive new classes. This provides a standard interface for all the slot classes. With this class the network can always make certain assumptions about the methods it can call in any derived slot class. As Figure 8 shows there are no actual data members present in this class--this is why it is abstract.

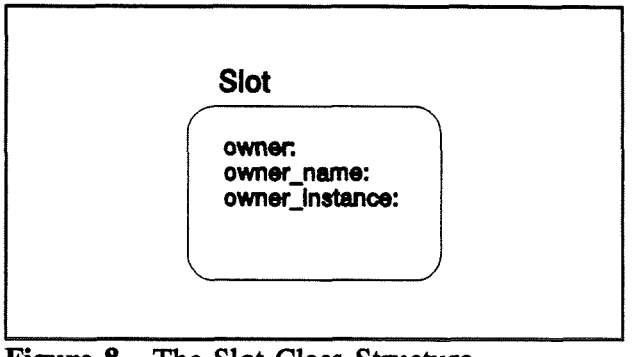

Figure 8 - The Slot Class Structure

The Value Class

The value class is the C++ representation of the CLIPS value fact and is derived from the slot abstract class. This class holds multifield values in the form of strings, integers, and floating points and provides methods for the manipulation of these multifield items. The structure represented in Figure 9 shows the slot class abstract members as well as some additional members--this is an example of inheritance in C++.

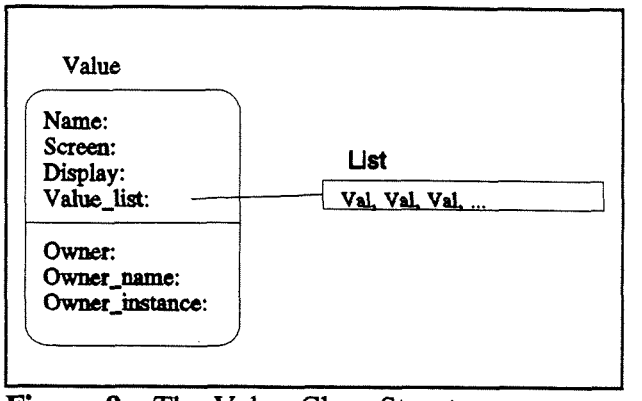

Figure 9 - The Value Class Structure

The Relation Class

The relation class, like the value class, is also a direct representation of a CLIPS fact--the relation fact; it is also derived from the slot class. This relation represents a "has-a" relationship, and when this object is inserted as a slot in a frame object, it defines the relationship between two frames. In addition, when a relation is created, the frame pointed to by the relation is notified that a relationship has been established. This allows a frame to know what frames have relations pointing to it. Figure 10 shows the structure of the relation class.

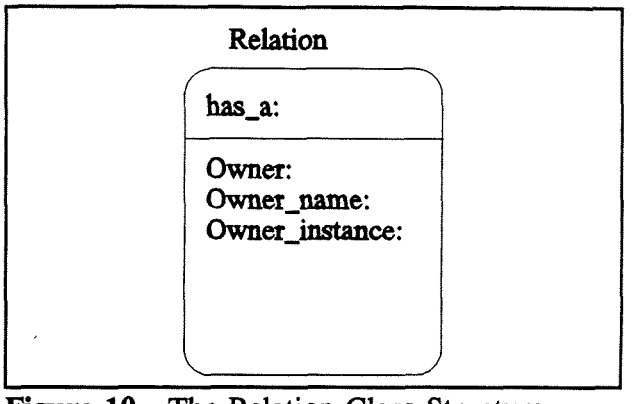

Figure 10 - The Relation Class Structure

The Val Class

The val class is used to represent all the types of data that could be used in a value class (i.e. strings, integers, floating points). Instances of this class are used by the value class to store values for value facts; the value class has a list of val objects as one of its data items. Figure 11 depicts the structure of the val class.

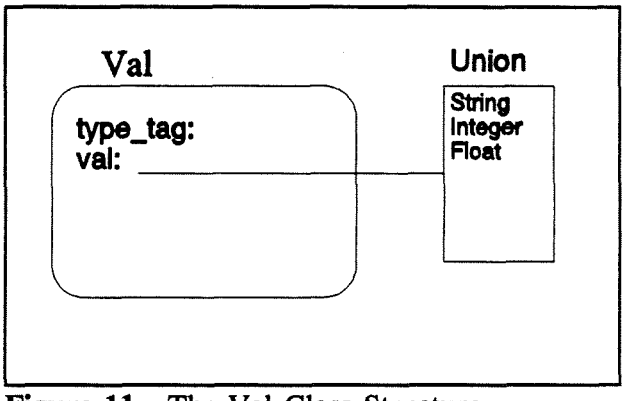

Figure 11 - The Val Class Structure

THE FRAME PARSER

An alternative to using class methods for constructing a network is the frame parser. The frame parser was designed to accept an input language that specifies CLIPS facts in the same format as the CLIPS representation uses and is listed in Appendix A. Using the frame parser relieves the user from having to create new frame class instances, reducing the complexity of the coding effort. This language does not provide queries about the network structure; the user must rely on the class methods.

In addition to the fact information, an action is prepended to the fact to indicate what action is to be taken with the fact. There are three supported actions: ADD, MODIFY, and DELETE. For example, the keyword ADD is used to add a new frame to the semantic network.

EXAMPLE USES OF THE C++ FRAME CLASSES

Several applications have been written which use the C++ frames in a procedural environment. The applications are suited to an environment such as C++ much more than an expert system shell like CLIPS; these types of programs are not easily expressed using CLIPS.

The Design Interface

One of the primary objectives of the Design Interface was to view changes in the semantic network as they occurred; the user should be able to specify the desired slots to view, and the Design Interface would automatically update the display as values change.

As facts come from the blackboard, they are parsed using the frame parser, described previously, and are stored in the semantic network of the Design Interface. In addition, a module in the Design Interface tries to match the incoming fact with a user-specified set of slots. If there is a match, the values are then displayed or updated, which ever case is appropriate.

The Pre-Design Module

Within the ICADS model the Pre-Design Module (PDM) is used to construct a building design starting with collections of objects that represent the spaces that are to be included in the final building structure as denoted by the Project Design Object Frames (PDO). The PDOs are loaded into the PDM via the frame parser which was described above.

Once the PDO frames are loaded into the system the PDM can now make queries about these frames. The spaces frames (PDO frames that represent spaces) are displayed as circles on the screen that the user can select and position as desired. Criteria are then specified by the user to assist the PDM in choosing a central space for the layout of the structure.

After the layout is agreed upon by user the building layout is sent to a CAD system as draw commands for display. A geometry interpreter notices these draw commands and sends messages, that are basically equivalent to the grammar specified by the frame parser, to a blackboard. Once these statements are received by the blackboard and a representation of the semantic network is built in CLIPS, the same messages are sent to any IDTs that require them.

This technique allows IDTs to be written in either C++ or CLIPS. In the case of CLIPS the messages sent from the blackboard are placed in the fact list and the CLIPS rules will

fire accordingly. If C++ is used the messages are handled by the frame parser which will build the same semantic network structure using the frame classes. These frame classes can then be used by functions written in C++.

CONCLUSIONS

Executing expert systems in a distributed environment has allowed for an increased level of complexity to be introduced. By having multiple representations, both procedural and expert system, available to programmers, the appropriate strengths of each paradigm can be used for a particular application.

The experiences of the ICADS project have proven both methods of representation to be useful. The sample applications discussed previously would not be as easily implemented in the CLIPS environment. Although, the introduction of the object paradigm to CLIPS 5.0 may remove the need to implement the C++ frames under certain circumstances. However, CLIPS 5.0 will never completely replace the C++ frame representation; the C++ representation allows existing applications to interface into the semantic network in an intuitive and direct manner.

REFERENCES

Assal, H. and L. Myers (1990). *An Implementation of a Framebased Representation in CLIPS.* First CLIPS Conference Proceedings, Houston, Texas. pp. 570-580.

Barr, A. and E. Feigenbaum (1981). *Frames and Scripts.* The Handbook of Artificial Intelligence. Vol. I., William Kaufmann, Stanford, CA.

Coyne, R. C., M. A. Rosenman, A. D. Radford, M. Balachandran and J.S. Gero (1990). *Knowledge-Based Design Systems.* Addison-Wesley, Reading.

Giarratano J., G. Riley (1989). *Expert Systems: Principles and Programming.* PWS-Kent, Boston.

Myers, L. and J. Pohl (1991). *Computer-Based Intelligent Design Assistance: Concepts and Strategies.* First International Conference on Artificial Intelligence in Design. Edinburgh, Scotland, U.K.

NASA (1989). *CLIPS Reference Manual: Version 4.3 of CLIPS.* Artificial Intelligence Section, Lyndon B. Johnson Space Center. Houston, TX.

Pohl, J., L. Myers, A. Chapman, L. Chirica, J. Snyder, H. Assal, J. Taylor, C. Johnson, D. Johnson (1990). *Knowledge-Based CAAD and the CLIPS Expert System Shell.* Technical Report, CADRU-04-90, CAD Research Unit, Design Institute, Cal Poly, San Luis Obispo, CA.

Pohl, J., L. Myers, A. Chapman, J. Snyder, H. Chauvet, J. Cotton, C. Johnson, D. Johnson (1991). *ICADS Working Model Version 2 and Future Directions.* Technical Report, CADRU-05-91, CAD Research Unit, Design Institute, Cal Poly, San Luis Obispo, CA.

Pohl, J., L. Myers, A. Chapman, J. Cotton (1989). *ICADS: Working Model Version 1.* Technical Report, CADRU-03-89, CAD Research Unit, Design Institute, Cal Poly, San Luis Obispo, CA.

Taylor, J. (1990). *A Framework for Multiple Cooperating Agents in an Intelligent Computer-Aided Design Environment.* (Masters Thesis). School of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA.

Taylor, J. and L. Myers (1990). *Executing CLIPS Expert Systems in a Distributed Environment. First CLIPS Conference Proceedings*, Houston, Texas. pp. 686-695.

APPENDIX A - Frame Parser Grammar

```
fact:
        YADD addactions
|       YMOD modactions
|       YDEL delactions
;

addactions:
        YFRAME frame_part
|       YVALUE value_part
|       YRELATION relation_part
;

modactions:
        YVALUE value_part
;

delactions:
        YFRAME frame_part
|       YVALUE value_part
|       YRELATION relation_part
;

frame_part:
        class instance
;

value_part:
        class attribute instance value_list
;

relation_part:
        class class instance instance
;

value_list:
        value
|       value_list value
;

value:
        YSTR
|       YINT
|       YREAL
;

class:
        YSTR
;

instance:
        YINT
;

attribute:
        YSTR
;
```