Integration of Object-Oriented Knowledge Representation with the CLIPS Rule Based System

David S. Logie and Hasan Kamil
Structural Analysis Technologies, Inc. (SAT)
4677 Old Ironsides Dr. Suite 250
Santa Clara, CA 95054
(408) 496-1120

Abstract

The paper describes a portion of the work aimed at developing an integrated, knowledge based environment for the development of engineering-oriented applications. An Object Representation Language (ORL) was implemented in C++ [2] which is used to build and modify an object-oriented knowledge base. The ORL was designed in such a way so as to be easily integrated with other representation schemes that could effectively reason with the object base. Specifically, the integration of the ORL with the rule based system CLIPS [1], developed at the NASA Johnson Space Center, will be discussed.

The object-oriented knowledge representation provides a natural means of representing problem data as a collection of related objects. Objects are comprised of descriptive *properties* and inter-*relationships*. The object-oriented model promotes efficient handling of the problem data by allowing knowledge to be encapsulated in objects. Data is inherited through an object network via the relationship links. Together, the two schemes complement each other in that the object-oriented approach efficiently handles problem data while the rule based knowledge is used to simulate the reasoning process. Alone, the object based knowledge is little more than an object-oriented data storage scheme; however, the CLIPS inference engine adds the mechanism to directly and automatically reason with that knowledge. In this hybrid scheme, the expert system dynamically queries for data and can modify the object base with complete access to all the functionality of the ORL from rules.

Introduction

This project was undertaken because of the need for a practical environment for the development of large expert systems, specifically, those involving engineering domains. In general, the motivation for this work can be summarized in the following:

- ☐ the limited expressiveness of rule-based knowledge representation, especially in engineering domains,
- ☐ the inability to build large, efficient, and comprehensive expert systems consisting of thousands of rules,
- ☐ the need to effectively store knowledge (i.e. acquired from the user, data bases, or inferred by a rule set) for later use, and
- ☐ the desire to have a common environment that could link expert systems with existing data bases and procedural programs.

Even in the preliminary stages of the development of an expert system for structural/mechanical design [3], we realized that a system with a minimum of usefulness could be comprised of thousands of rules. This fact introduced some concerns with respect to hardware and software limitations and the practicality of maintaining such an extensive knowledge base. One of the most powerful uses of this enhancement is the ability to chain rule sets. A large set of rules can be decomposed into smaller sets which reason about specific subproblems. For example, a rule could state that **if** a certain piece of knowledge is *unknown* **then** load another rule set that will infer that data. The original rule set can put itself in queue to return and continue processing, transparent to the user. Also, previously autonomous expert systems can now share data through common objects and communicate with each other through the ORL queries. As illustrated in Figure 1, a very large network of rule sets can be developed giving the illusion of a large expert system when, in fact, only a small set of rules are being processed at any one time. This capability becomes especially important on a personal or desktop computer platform. Developing, modifying, updating and verifying knowledge bases for large applications is a less formidable task when small rule sets can be edited and tested independent of the entire application.

Another advantage realized from this enhancement is that rule sets shrink considerably. This is primarily because rules for handling user queries and checking user responses are now handled by the ORL. Rule sets need only contain rules for ORL queries, the actual problem solving rules and those rules that report the results[1] . An existing set of rules can easily be modified to take advantage of the ORL capabilities[2].

Disk storage of knowledge has proven to be very useful also. In our scheme, a rule set is invoked in the context of a project. Objects are first searched for in a project specific location and then in a global storage area. In a run-time environment, modifications to the object base are only specific to a particular project. This context sensitivity allows the user to examine the effect of various responses on the recommendations or findings of an expert system by simply changing contexts. For example, in a medical diagnosis system the context would be set to refer to a particular patient.

The ultimate intention of this effort is to develop a fully integrated environment in which the same ORL query initiated from a rule can not only query the user but also result in a query to an existing data base or the invoking of a procedural program [4]. The details of where the information should be retrieved would be specified as the object base is developed through the use of property metaslots (discussed later). Optimally, this integration should be seamless to the user and function efficiently in a networked environment. With this capability, ORL/CLIPS applications could have limitless potential for practical use.

For the remainder of the paper, the use of the ORL and the object-oriented knowledge representation scheme to build practical expert systems is discussed and demonstrated.

Use of the ORL

The ORL consists of a concise set of functions for building and maintaining an object base. One of the main goals in the

---

[1] A generic reporting mechanism is being developed that may eliminate the need for the latter type of rule, thus, leaving only the rules specifically for reasoning.

[2] Existing CLIPS rules will still run without modification.

development was to keep the use of the ORL as simple as possible so that engineers or experts in other domains, without extensive computer programming experience, could develop knowledge bases and, furthermore, that non-experts could easily utilize the resulting expert systems.

The type of commands available include those for file operations, building and displaying classes and objects, querying and asserting property values, editing the object base, and an interface to the usual CLIPS command line. The file operations allow the user to set the current project, save and load objects to and from disk, reset memory resident object properties to *unknown* or to clear memory completely. Note that when running a rule set, objects are automatically loaded as needed but must be saved explicitly to permanently store any changes made by the rules.

Command line functions for building and modifying the object base include making classes and objects, making an instance of a class, copying objects, or adding and removing properties and relationships. Menu-oriented editors are available for specific modifications such as changing the name or type of a property or defining metaslots.

To access ORL commands from a rule the developer uses the "ORL" function as the first item in the right hand side pattern. The remainder of the pattern is precisely the ORL command line function and arguments. For example, to save an object to disk from a rule, one would write:

```
( ORL save < object name> ).
```

Just as in CLIPS, several destructive functions are disallowed from within a rule.

Classes and Objects

Classes and objects are the basic structures of the knowledge representation scheme. They contain descriptive properties and relationships to other classes and objects. When a property value is required in a rule set, the class or object must be queried for that specific property's value(s). Queries to classes and objects only differ in that a query to a class results in all the instances of that class being queried. In general, an ORL query from a rule takes the form:

(ORL get <class/object name> <property name(s)>)

and results in asserted facts of the form:

(<object name> <property name> <value> (certainty)[3]).

Qualifiers for the queries such as less-than, greater-than, or equal-to need to be implemented for fully functional querying; however, these types of tests are currently available in CLIPS which accounts for their low priority in the development.

In the same way, permanent assertions to the object base take the form:

(ORL assert <object name> <property name> <value> (certainty))

and result in the a fact:

(<object name> <property name> <value> (certainty))

Other queries return the instances of a class or related parts of an object. For example, to find out the instances of a class the query would be:

(ORL getinstances <class name>)

and would return facts as:

(<class name> instance <object name>)

which could be matched on the left hand side of a rule for deleting instances of a class.

Properties and Metaslots

Properties (often called 'Attributes' in similar schemes) are the mechanism by which classes and objects are described. They simply hold one or more values as they are asserted. Currently, a property may be of type integer, float, text, or boolean. Other specialized property types are being developed such as filename, equation, data

---

[3] For brevity, certainty factors will not be discussed, however, properties may optionally have a certainty applied from 1-100%

base, and program. A property will automatically handle the checking of user responses and build the appropriate CLIPS facts as values are assigned.

Defining a metaslot for a property adds a considerable amount of versatility. First, a metaslot can be used to put constraints on the values that a property can hold by specifying a list of allowable values or a range of numeric values. Other useful features include assigning initial and default values for the property and defining the prompt displayed to the the user.

Possibly, the most powerful feature of a metaslot is the ability to define a search strategy with the "Order of Sources." The USER is the default source for information when a property value is queried. Alternatively, the knowledge base developer may wish the property to assume the initial value when queried for the first time or the default value if the user responds *unknown* to a query. Also, it may be desirable to query an existing data base or invoke a procedural program to generate data.[4] These facilities may lessen the need for user interaction when the level of knowledge of user may be in question or may make it easier to develop autonomous expert systems for applications such as robotics.

Relationships

Relationships allow properties to be inherited by related classes and objects. The most common types are the *instance* and *instance_of* relationships between a class and its instance. When an instance of a class is created, the relationships between them are automatically created so that the new object can inherit properties in the class hierarchy. Other types include *is_a* and *subclass* relationships between classes (e.g., **Jet** *is_a* **Airplane, Airplane** has *subclass* **Jet**) and *part_of* and *subobject* relationships between objects (e.g., **wing-x** is *part_of* **airplane-y, airplane-y** has *subobject* **wing-x**).

As mentioned earlier, the relationships come into play when the classes and objects are queried. If a class is queried for a property value, it will automatically pass the query on to its instances. Similarly, if an object is queried for a property value which it doesn't have, it may pass the query on to related objects according to the

[4] These latter capabilities are currently under development.

current inheritance protocol. The relationship capability promotes efficient handling of data by eliminating unnecessary redundancy.

Example

The example automotive diagnosis system that was distributed with CLIPS will be used for the purpose of demonstration. First, compare the rules for querying the user for the working state of the engine. With CLIPS alone, the rule was:

```
(defrule determine-engine-state
      ?rem <- (query phase)
      (not (working-state engine ?) )
=>
      (retract ?rem)
      (printout t "What is the working state of the engine:" t)
      (printout t "  (normal/unsatisfactory/does-not-start)? ")
      (bind ?response (read) )
      (assert (working-state engine ?response) )
)
```

The user must type the complete response, correctly. Using the ORL, an object, engine, is created with the property, working-state, having a metaslot that defines the allowable values and prompt as above. The new rule is:

```
(defrule determine-engine-state
      ?rem <- (query phase)
=>
      (retract ?rem)
      (ORL get engine working-state)
)
```

The new query to the user is:

```
What is the working state of the engine?
      1. normal
      2. unsatisfactory
      3. does not start
      4. unknown or other

Selection:
```

Even this small rule set was reduced by two pages of text and several rules. Note that the original rule set made no provision for incorrectly typed responses or any other error checking. In the second case, the user ***cannot*** make a typing error and provisions

were made for *unknown* responses. It was found that rules modified to employ ORL objects and functions tend to read more naturally so that they can be more easily debugged or updated.

To fully utilize the ORL in building a useful automotive diagnosis system, a developer would define a class, auto, with related parts such as the engine or doors and then have the rule set instantiate these classes for a specific case. Also, the expert system could be divided into modules for specific problem areas such as the engine or transmission containing the expertise of specialized mechanics. Modularity makes an expert system more easily extendable. This approach is being used in-house at SAT in the development of rule sets for designing structural/mechanical components. With this approach, it was possible to develop rule modules containing basic knowledge ourselves and then consult experts in specialized areas to extend the capabilities of the knowledge based system or tailor it to a specific engineering problem.

Conclusions

The salient features of the ORL were discussed, including typical functions employed in the development and use of this object-oriented/rule-based knowledge representation scheme. The object-oriented paradigm is especially expressive in representing static, structured knowledge. The simple example of the automotive diagnosis system showed that the size of a CLIPS rule set can be significantly reduced using the ORL. Accompanying the reduction in size is improved efficiency and built-in error handling. Context sensitivity and permanent (data base like) disk storage promote flexibility in developing knowledge bases. The ultimate aim of this work will result in an integrated environment able to access data in distributed data bases and invoke procedural programs through a common user interface or from expert systems. With these capabilities there is no limit to the size of knowledge bases that can be built or the range of applicable domains to which such an integrated system could be applied.

References

1. CLIPS User's Manual, Version 4.3, Artificial Intelligence Section, Johnson Space Center, NASA, 1989.

2. Stroustrup, Bjarne, The C++ Programming Language, 1987.

3. Kamil, H., Vaish, A. K., and Berke, L., "An Expert System for Integrated Design of Aerospace Structures," Fourth International Conference on Application of Artificial Intelligence in Engineering (AIENG89), Cambridge, England, July, 1989.

4. Kamil, H. and Logie, D.S., "Toward an Integrated, Knowledge Based Engineering Environment," International Symposium on Artificial Intelligence, Robotics and Automation in Space (I-SAIRAS), Kobe, Japan, November, 1990.

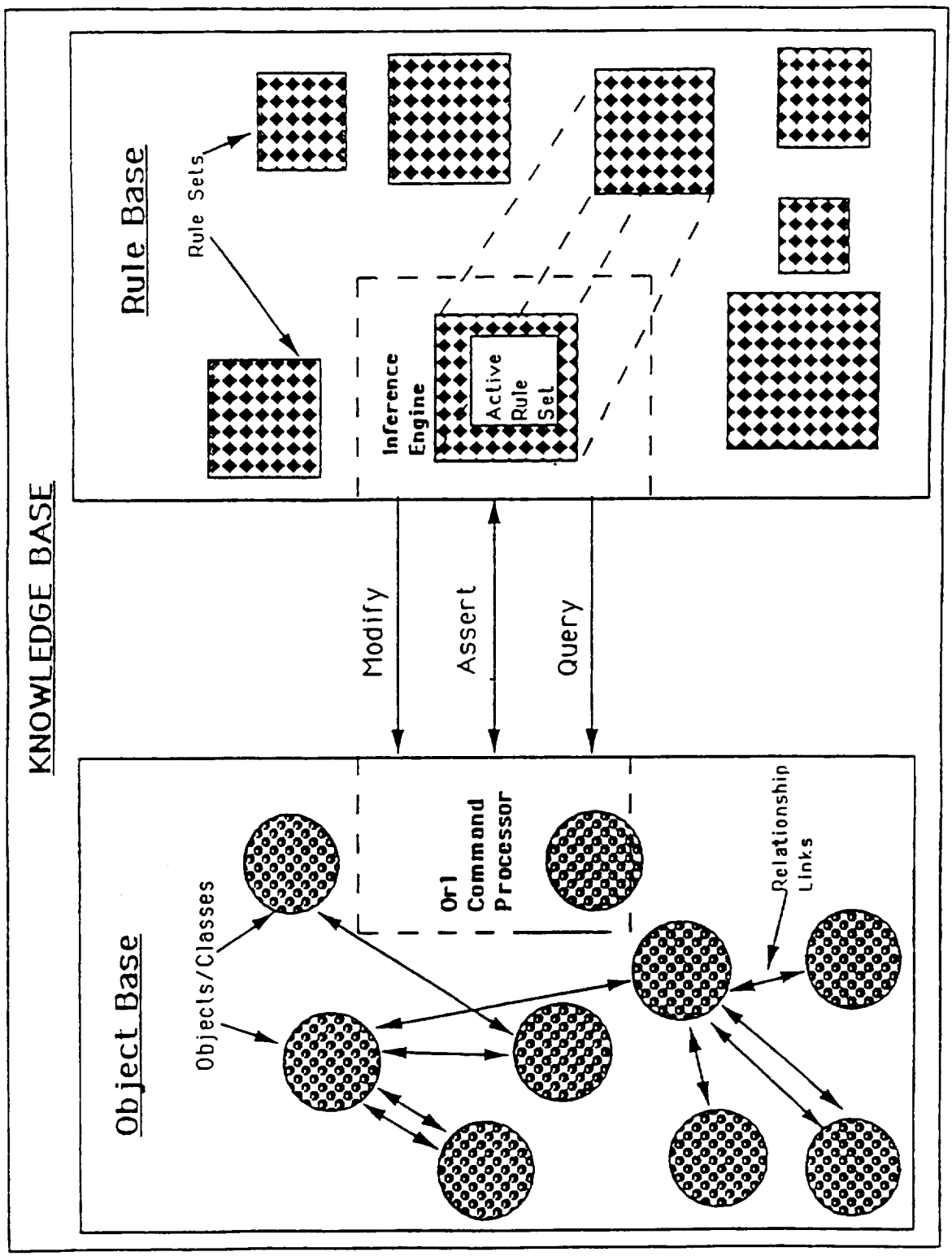

FIGURE 1. OBJECT-ORIENTED/RULE-BASED KNOWLEDGE REPRESENTATION SCHEME