CBCV: A CAD-Based Vision System (I)

Thomas C. Henderson, John Evans, Lane Grayston, Allen Sanderson, Leihg Stoller, Eliot Weitz;
University of Utah, Department of Computer Science, Salt Lake City

1. Introduction

Computer Aided Design, or CAD as ist is better known, has many advantages to offer in the design and development of manufactured items. Computer vision, on the other hand, has yet to make major inroads into the manufacturing domain. We believe that a closer tie between geometric models and computer vision will lead to greater application of computer vision techniques in industry and to a more efficient and effective manufacturing process.

In this report, we examine the role of Computer Aided Geometric Design models in providing support for computer vision techniques. In particular, we examine the requirements placed on CAD systems to achieve useful vision functions, andplanar faces. A very specific interpretation tree is generated for an object using a set of object specific rules. The rules were selected by hand rather than generated automatically. There doesn't appear to be any algorithmic approach for the ap, and on the geometric object under consideration.

We give detailed examples of both CAD and computer vision systems, as well as synthesis techniques for automatically deriving visual inspection, object recognition, and pose determination modules. We use the Alpha-1 CAGD system, developed by *Rich Riesenfeld* and *Elaine Cohen* and their colleagues at the University of Utah for all of our design and 3D data manipulation. Their system is a boundary representation, B-spline based modeler, and provides exceptional design and analysis capabilities.

For the computer vision system, we use the IKS (Image Kernel System). This developed out of an early set of vision tools acquired from *Bill Havens* at the University of British Columbia. The majority of the current IKS system, however, has been developed over the last few years at the University of Utah. The functions range from low-level image to image processing routines to 3D intrinsic characteristic functions for the analysis of 3D range data.

The knowledge-based component of the system described here has been developed on top of PCLS (Portable Common Lisp Standard) in FROBS (FRames and OBjectS). These systems were developed by *Bob Kessler* and the PASS (Portable Ai SystemS) group at the University of Utah.

2. Computer Aided Geometric Design

2.1. General Considerations

Computer vision has been an active research area for over 25 years. In the past, emphasis was on low level processing such as intensity and signal processing to perform edge detection. More recently, models of objects and knowledge of the working environment have provided the basis for driving vision systems. This is known as model-based vision. The pursuit of the fully automated assembly environment has fueled interest in model-based computer vision and object manipulation. This in volves building a 3-D model of the object, matching the sensed environment with the known world and determinig the position and orientation of the recognized objects. The goal is to provide a solution to the problem of visual recognition in a well-known domain.

In the automation environment, recognition schemes and representations have typically been constructed using ad hoc techniques. Although objects used in the assembly process are designed with a CAD system, generally there is no direct link from the CAD system to the robotic Workcell. This means the recognition systems are constructed independently of the CAD model database. What is desired is a systematic approach for both the generation of representations and recognition strategies based on the CAD models. Such a system provides an integrated automation environment. The system is composed of several components: a CAD system, a milling system, a recognition system and a manipulation system. In this paper, the automatic generation of recognition strategies based on the CAGD model is studied. It has also been determined that the use of shape, inherent in CAGD models, can also be used to drive the recognition process. Others have been studying portions of this system. Recent work by *Ho* has focused on the generation of computer vision models directly from a CAGD model [2], [15].

The work described here investigates the use of geometric knowledge in constructing 2D recognition codes. These codes provide a robust mechanism for recognition and localization of two-dimensional objects (occluded as well as non-occluded) in typical manufacturing scenes.

One of the first researchers to study the automatic synthesis of general recognition strategies was *Goad* [11]. He was concerned with automatic programming for 3-D model based vision. His work generated a recognition scheme for matching edges based on a general sequential matching algorithm. His algorithm proceeded in three steps: (1) predict a feature, (2) observe (match) a feature, and (3) back-project (refine the object hypothesis based on step 2). These three steps form a template which is used by the automatic programming phase. He used a unit sphere to gather loci of view angles (camera positions) which represent orientations of the object. The only features used were straight edges from intensity images and the search trees were generated from a template and ordered by hand rather than automatically. His system didn't consider partial occlusion. However, this was a major contribution since it was one of the first attempts to automate the generation of recognition schemes.

Another influential project was the 3DPO system by *Bolles* and *Horaud* [4]. This work is the 3-D generalization of the Local Feature Focus method [3]. Their system annotates a CAD model producing what is called the extended CAD model. From this model, feature analysis is performed to determine unique features from which to base hypotheses. The focus feature in their system is the dihedral arc. When the recognition system finds a dihedral arc, it looks for nearby features which are used to discriminate between model arcs with similar attributes. From these, an object's pose is hypothesized and subsequently verified.

Recently, *Ikeuchi* has explored the use of interpretation trees for representation of recognition strategies [16]. His system uses the concept of visible faces to generate generic representative views, called aspects. From this set of aspects, an interpretation tree is formed which discriminates among the different aspects. His system uses a variety of object features such as: EGI, face inertia, adjancency information, face shape, and surface characteristics. Most of these features are based on planar faces. A very specific interpretation tree is generated for an object using a set of object specific rules. The rules were selected by hand rather than generated automatically. There doesn't appear to be any algorithmic approach for the application of the rules to discriminate between the aspects. The branching on the tree seems to be a function of the particular aspects chosen rather than being based on the geometric information in the model.

Hansen and *Henderson* [13] have also proposed a CAD-based 3D recognition and localization scheme based on strategy trees. That system isn't dependent on a certain class of features but rather can be extended to include many classes of features. The system also performs automatic selection of features based on a set of constraints: feature filters. These features are used to form a strategy tree which provides a scheme for hypothesis formation, corroborating evidence gathering and object verification. The flexibility of this approach makes it significantly different from related work.

Our main goal of is the automatic synthesis of recognition system specifications for CAD-based 2D and 3D computer vision [12], [13], [22]. Given a CAD model of an object, a specific, tailor-made system to recognize and locate the object is synthesized.

To attain this goal, the following problems have been studied:

1. Geometric Knowledge Representation: The use of geometric data is central to a strong recognition paradigm. Weak methods can only be avoided when better information is available. The Alpha_1 B-spline model allows the modeling of freeform sculptured surfaces. To obtain the geometric features of interest for 3-D recognition, techniques for the transformation to a computer vision representation have been developed.

2. Automatic Feature Selection: The part to be recognized or manipulated must be examined for significant features which can be reliably detected and which constrain the object's pose as much as possible. Moreover, such a set of features must cover the object from any possible viewing angle. In solving the feature selection problem, a technique is available for synthesizing recognition systems. This produces much more efficient, robust, reliable and comprehensible systems.

3. Recognition System Synthesis: Once a robust, complete and consistent set of features has been selected, a recognition strategy is automatically generated. Such a strategy takes into account the strongest features and how their presence in a scene constrains the remaining search. The features and the corresponding detection algorithms are welded, as optimally as possible, into a search process for object identification and pose determination. The automatic synthesis of search strategies is a great step forward toward the goal of automated manufacturing. Generation of strategies is constrained, not only by the feature selection process but, by the actal task to be accomplished. Thus, strategies for a specific task might not be as strong when applied to a different task; strategies are task specific.

The remainder of this paper explains how these three components can be exploited to automate the process of selecting proper features and recognition schemes for specific goals. Algorithms are described which were developed for feature selection and which give supporting evidence for their formulation.

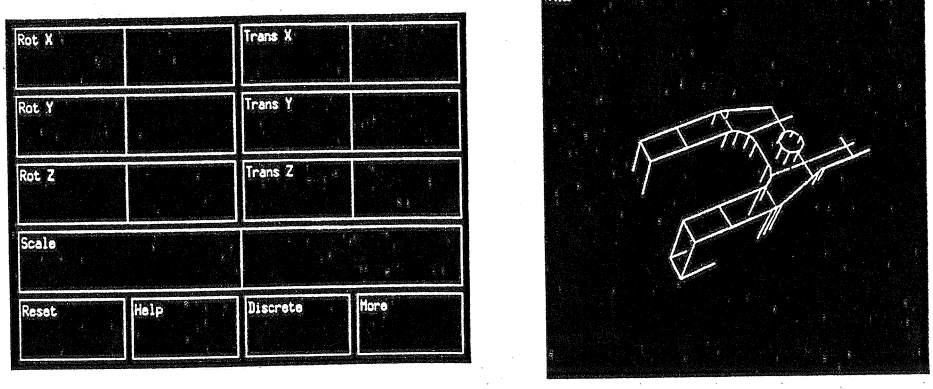
Computer vision utilizes object models in a different manner than computer graphics or CAGD. In CAGD, the models must contain information about the 3-D object for rendering, performing finite element analysis, milling and other processes. Computer vision is concerned with recognition of the objects from sensory data. CAGD models must contain information for the local design operations such as what shape to extrude or what is the profile curve for a sweep operation. Features used in construction of models are implicitly rather than explicitly used in the CAGD representation. For example, a dihedral edge formed from two adjoining surfaces isn't modeled as an edge per se but as two surfaces with adjacency information.

Constructive solid geometry (CSG) and boundary representations are the best understood and currently most important representation schemes in computer aided design. Present day 3-D wireframe models used in CAD and model-based vision have many deficiencies including ambiguity – it is easy to build a wireframe model that can be surfaced in several ways [19]. In CSG, the basic idea is that complicated solids can be represented as various ordered "additions" and "subtractions" of simpler solids by means of modified versions of Boolean set operators-union, difference and intersection [18]. For inherent boundary representations a number of different approaches are used. These include Coons patches, bicubic surface patches, Bezier methods and B-splines [1].

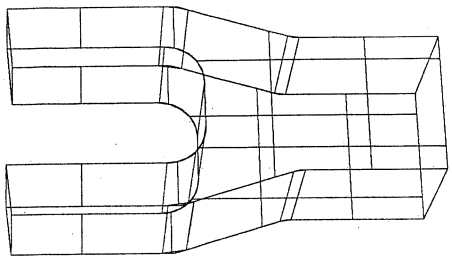
Most Geometrical Modeling Systems (GMS) use a limited class of primitives such as rectilinear blocks and conic surfaces (cylinders, cones and spheres). Although these suffice to design a large number of conventional unsculptured parts, a GMS which includes sculptured solids is highly desirable. Also since the sculptured design is surface oriented, it is easier to incorporate it in a boundary based system. In general, boundary modelers tend to support stepwise construction of the models more easily than CSG modelers but require greater data storage. CSG modelers are inadequate for modeling sculptured parts: they have no capability at all for constructing and using sculptured surfaces as part of the boundary of the solid model. Some advantages of boundary representation are: there are many known surface models available from which to choose [1]: the mathematics of surface representation is well developed and complex shapes can often be represented with a single primitive [8], [21]; and it results in an intuitive model. A minor disadvantage is that it may be difficult to ensure the validity of a boundary representation of a set. On the other hand, CSG representations are not unique in general, since a solid may be constructed in many ways; the final result may not be easily visualized by looking at the primitives. However, the CSG representation is concise, validity is guaranteed and such a representation can be easily converted to a boundary representation. The comparison of CSG and boundary representation methods can be found in [19], [20].

Until recently it was not possible to carry out Boolean operations on sculptured surfaces. Work by *Thomas* [21] attempts to combine the best attributes of CSG and surface-based representation systems by using subdivision techniques developed by *Cohen* et al. [10]. He uses a uniform boundary representation. The "primitives" are solids bounded by B-spline surfaces. As compared to the other work in solid modeling, his method does not require that the objects being combined have closed boundaries; they must only satisfy a weak completion criterion. Thus this method results in a powerful shape description system which allows the combination of primitives using set operations into arbitrarily-complex objects bounded by curved surfaces and the production of a model which represents such objects. Adjacency information about surface points and the intersection curve between two surfaces as a polyline can be obtained. Although he has used B-spline surfaces, his techniques are applicable to any surface representation scheme [8]. All this work has been incorporated in the Alpha_1 system [9]. (More details about Alpha_1 are presented below.) Thus, the advantages of both CSG and sculptured surface representation can be obtained in the shape representation of objects and the combination of objects via set operations. As a result of these significant advances in CAGD, we decided to use the Alpha_1 system for exploring the computer vision application.

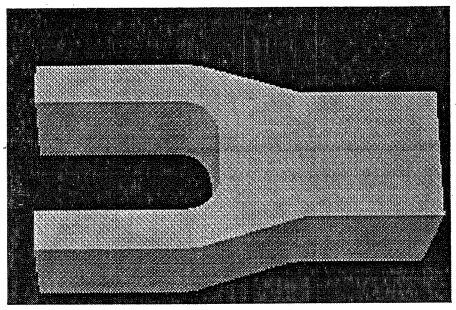
Alpha_1 is an experimental CAGD based solid modeler system incorporating sculptured surfaces [9]. It allows in a single system both high quality computer graphics and freeform surface representation and design. It uses a rational polynomial spline representation of arbitrary degree to represent the basic shapes of the models. The rational spline includes all spline polynomial representations for which the denominator is trivial. Nontrivial denominators lead to all conic curves. Alpha_1 uses the Oslo algorithm [10] for computing discrete B-splines. Subdivi-



1 *Design and Viewing Control Windows*



2 *Surface Grid for Main Shape of Linkage*



3 *Rendering of Main Shape of Linkage*

sion, effected by the Oslo algorithm, supports various capabilities including the computation associated with Boolean operations, such as the intersection of two arbitrary surfaces [21]. B-splines are an ideal design tool, they are simple yet powerful; many common shapes can be represented exactly using rational B-splines. For example, all of the common primitive shapes used in CSG systems fall into this category. Other advantages include good computational and representational properties of the spline approximation: the variation diminishing property, the convex hull property and the local interpolation property. There are techniques for matching a spline-represented boundary curve against raw data. Although the final result may be an approximation, it can be computed to any desired precision (which permits nonuniform sampling). At present, tolerancing information is not included in the object specification in Alpha_1 system.

Given the CAGD model (perhaps by combining several modeling paradigms), a corresponding set of vision models (with some control structure) is generated. Once these models are available, they provide the basis for standard 2-D and 3-D scene analysis. An early example of such an interactive system is the ACRONYM system [5], [6] designed for applications in computer vision and manipulation. The world is described to ACRONYM as volume elements and their spatial relationships and as classes of objects and their subclass relationships. It uses a hybrid CSG and general sweep scheme for the representation of rigid solids. The representations are CSG-like trees whose leaves are generalized cylinders. Like PADL (a geometric modeling system [7]) it allows variation in size, limited variation in structure and variation in structural relationships of the modeled objects. However, in ACRONYM, it may be difficult to design algorithms for computing properties of objects.

The Alpha_1 modeler allows the user to design an object by giving a sequence of commands. These commands define the geometry of the object. At the same time, the result of each command can be viewed in a separate window. Although specific Alpha_1 commands are given here, we use them to describe a more general philosophy of design. The underlying motivation is to exploit wherever possible the geometric modeling system functions to provide the information required to support computer vision applications.

2.2. Front Suspension Shock Linkage

To illustrate our modeling philosophy, we will use the design of a front suspension shock linkage. This part was designed by *Samuel H. Drake* at the University of Utah, and was actually milled and used in a small off-road vehicle built for the SAE Mini-Baja student competition.

The design is specified in such a way so as to facilitate the machining and automatic inspection of the part. The annotated specification is now given. First, an X window is provided so that the result of every geometry creating command can be viewed. A control window is also provided for viewing transformations (see Figure 1).

```
% creates geometry for the front suspension
shock linkage for Mini Baja vehicle
grab xgen; – Grab an X window to display the
results
```

The next commands bring in the (Lisp) definitions of design features (e.g., pockets, holes, etc.) and set up reference axes.

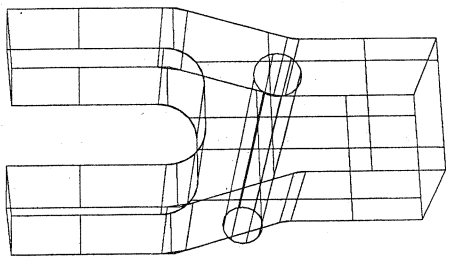
```
{
load features; – Load feature definitions
setSrfNorms(T);
Xref := XAxis; – Define coordinate axes
Yref := YAxis;
Zref := ZAxis;
};
```

Now, a set of bounding lines can be defined; the *reverseObj* command is used to keep the normal pointing into the shape (i.e., each bounding line has an orientation).

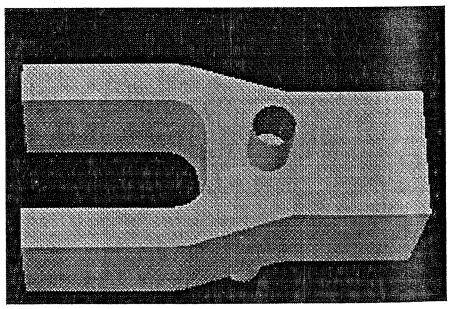
```
{
UpLinkConstLine1 := lineVertical(-0.5);
UpLinkConstLine2 := lineVertical(2.5);
UpLinkConstLine3 := lineHorizontal(0.75);
UpLinkConstLine4 := lineHorizontal
                    (−0.75);
UpLinkConstLine5 := reverseObj(lineHori-
                    zontal(0.5));
UpLinkConstLine6 := lineHorizontal
                    (−0.5);
UpLinkConstLine7 := lineHorizontal(0.44);
UpLinkConstLine8 := reverseObj(lineHori-
                    zontal(−0.44));
UpLinkConstLine9 := reverseObj(lineVer-
                    tical(0.875));
UpLinkConstLine10 := lineVertical(1.5);
```

The following commands define the remaining points, arcs, and segments needed to specify the shape. Note that the shape is defined in the x-y plane.

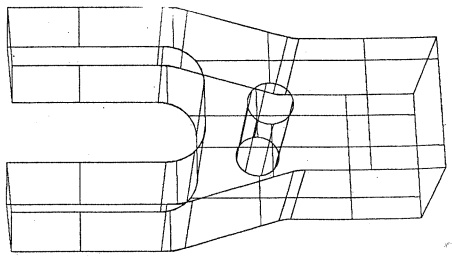
```
UpLinkConstArc1 := arcRadTan2Lines
                   (0.275, UpLinkConst
                   Line7, UpLinkConst
                   Line9)$
UpLinkConstArc2 := arcRadTan2Lines
                   (0.275, UpLinkConst
                   Line9, UpLinkConst
                   Line8)$
UpLinkConstPt1 := centerOfArc (UpLink
                   ConstArc1);
UpLinkConstPt2 := centerOfArc (UpLink
                   ConstArc2);
UpLinkConstCir1 := circleCtrRad (UpLink
                   ConstPt1, 0.585)$
UpLinkConstCir2 := circleCtrRad (UpLink
                    ConstPt2, 0.585)$
UpLinkConstPt3 := ptIntersect2Lines
                   (UpLinkConstLine2,
                   UpLinkConstLine5);
UpLinkConstPt4 := ptIntersect2Lines
                   (UpLinkConstLine5, Up
                   LinkConstLine10);
UpLinkConstPt5 := ptIntersect2Lines
                   (UpLinkConstLine3, Up
                   LinkConstLine1);
UpLinkConstLine11 := linePtCircle (UpLink
                     ConstPt4, UpLink
                     ConstCir1, T);
UpLinkConstLine12 := linePtCircle (UpLink
                     ConstPt5, UpLink
                     ConstCir1, nil);
UpLinkConstPt6 := ptIntersectCircleLine
                   (UpLinkConstCir1,
                   UpLinkConstLine11);
UpLinkConstPt7 := ptIntersectCircleLine
                   (UpLinkConstCir1,
                   UpLinkConstLine12);
```



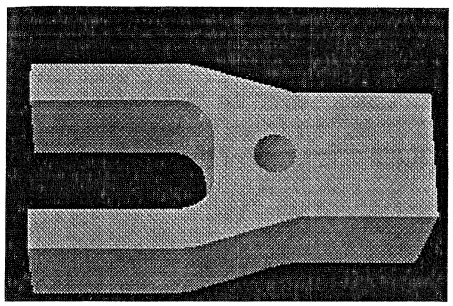
4 *Surface Grid of Shape with Solid for Hole*



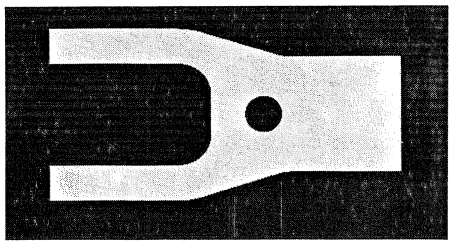
5 *Rendering of Shape with Solid for Hole*



6 *Surface Grid of Boolean Combined Surfaces*



7 *Rendering of Linkage*



8 *Orthogonal Rendering of the Linkage*

```
UpLinkConstArc3 := arcRadTan2Lines
                   (0.275, UpLinkConst
                   Line5, UpLinkConst
                   Line11)$
UpLinkConstArc4 := arcEndCenterEnd (Up
                   LinkConstPt6, UpLink
                   ConstPt1, UpLinkConst
                   Pt7)$
UpLinkConstPt8 := ptIntersect2Lines (Up
                   LinkConstLine1, UpLink
                   ConstLine7);
UpLinkConstPt9 := ptIntersect2Lines (Up
                   LinkConstLine1, UpLink
                   ConstLine8);
UpLinkConstPt10 := ptIntersect2Lines (Up
                   LinkConstLine1, Up
                   LinkConstLine4);
UpLinkConstPt11 := ptIntersect2Lines (Up
                   LinkConstLine6, Up
                   LinkConstLine10);
UpLinkConstLine13 := linePtCircle (UpLink
                   ConstPt10, UpLink
                   ConstCir2, T);
UpLinkConstLine14 := linePtCircle (UpLink
                   ConstPt11, UpLink
                   ConstCir2, nil);
UpLinkConstPt12 := ptIntersectCircleLine
                   (UpLinkConstCir2, Up
                   LinkConstLine13);
UpLinkConstPt13 := ptIntersectCircleLine
                   (UpLinkConstCir2, Up
                   LinkConstLine14);
UpLinkConstArc5 := arcEndCenterEnd (Up
                   LinkConstPt12, UpLink
                   ConstPt2, UpLinkConst
                   Pt13)$
UpLinkConstArc6 := arcRadTan2Lines
                   (0.275, reverseObj(Up
                   LinkConstLine14), Up
                   LinkConstLine6)$
UpLinkConstPt14 := ptIntersect2Lines (Up
                   LinkConstLine6, Up
                   LinkConstLine2);
```

The object's profile can now be defined:

```
UpLinkProfile1: = profile (UpLinkConstPt3,
UpLinkConstArc3, – Create the 2D shape
UpLinkConstArc4, UpLinkConstPt5,
UpLinkConstPt8, UpLinkConstArc1,
UpLinkConstArc2, UpLinkConstPt9,
UpLinkConstPt10, UpLinkConstArc5,
UpLinkConstArc6, UpLinkConstPt14,
UpLinkConstPt3)$
```

Now we define the bounding z positions and extrude the shape in 3D:

```
UpLinkExPt1 := pt(0.0, 0.0, 0.55); – Define
               the z extent and extrude the
               shape
UpLinkExPt2 := pt(0.0, 0.0, -0.55);
UpLinkShape := extrude (UpLinkProfile1,
               UpLinkExPt1, UpLinkExPt2,
T, T)$
};
```

Thus, the basic shape is defined (see Figures 2 and 3). Next we add a hole:

```
UplinkFixHole2 := objTransform (hole(ori-
                  gin, 0.3125, 0.15, 0.0, T),
                  tx(1.325), tz(0.075))$ –
                  Define a hole feature
```

We can view the hole shape with the profile shape as shown in Figures 4 and 5. However, to actually put the hole through the profile surface, we need to perform a Boolean subtraction. First, we store the two shapes (as defined by the variables UpLinkShape and UpLinkFixHole2) to a file named UpLinkTest Op.a1.

```
dumpa1File (list(UpLinkShape, UpLinkFix
Hole2), "UpLinkTestOp.a1")$
```

At the shell command level, we then perform the Boolean set operation which subtracts the hole from the main shape:

```
cs>set_op<UpLinkTestOp.a1>UpOut.a1
```

The final object is shown in Figures 6 and 7.

2.3. Model Synthesis

The geometric specification described in the previous section facilitates the direct extraction of object features. However, many modeling systems do not allow this. Therefore, the easiest way to provide models for a 2D vision system is to render an image from the CAD model and provide that image as the training set to the 2D vision system.

That is what we have done here. However, the viewing parameters must be carefully selecteds in order to produce an orthogonal view of the object. Figure 8 shows such a rendered version of the linkage. This constitutes the input, then, to the computer vision training techniques.

Note that this method also permits the analysis of arbitrary surface geometries, whereas a syntactic approach to discovering surface features in the Alpha_1 specification, may be quite complicated.

3. The FROBS Knowledge and Rule Base

The knowledge-base component of CBCV is written in FROBS (FRames + OBjectS) which is an object-oriented frames package that runs on top of CommonLips and provides:

- object oriented programming
- frame based programming
- daemons
- rule based programming.

An overview of the system is given here; for more details, see the FROBS Manual [17].

3.1. Overview of FROBS

The basic building block of the FROBS package is called a module. Modules consist of a class FROB and all of its associated methods. This provides for total method and data access hiding with no distinction between methods and slots. The organization of class FROBs can be viewed as a tree structure, although more complicated schema-type structures are possible through multiple inheritance. FROB class instances are leaves of the tree.

The class frob is used to define the structure of instance frobs of that class. It is also the frob that daemons and methods are defined over. Inheritance of methods is done through the class frobs. A special feature of the class frob is that it is an instance of itself. It can be used like any other instance of the class. Figure 9 shows how an algorithm class FROB and a subclass FROB are defined.

FROBS are used to build both the knowledge-based vision system and the application system it synthesizes. This allows templates in the knowledge-based system to be directly used in the application system. The concept of logical sensors is implemented easily using objects to form logical sensors [14]. Class FROBS represent logical sensor templates to be instantiated for application system synthesis.

Most importantly, the FROBS package provides forward chaining rules as well as slot daemons. Slot daemons are useful for automatic data consistency checking and hidden slot calculations. The forward chaining rules provide the mechanism needed to create the knowledge base.

3.1.1. System Support

The knowledge-based system must have utilities for supporting the networking of logical sensors and objects. These utilities provide the foundation from which the system is built. Higher level utilities are built on top of

```
(def-class algorithm nil
    :slots (name
            size
            language
            machine

(def-class feature-calculator ({class algorithm})
    :slots (feature-type
            focus-type))
```

9 *Example of FROB class Definition*

```
(def-rule select-lff
        :type ((?req requirements))
        :prem((not (member 'lff (applications ?req)))
                (equal 'recognition (task ?req)))
        :conc ((assert-val ?req 'applications
                (cons 'lff (applications ?req)))
        (make class system-specs
                :task 'recognizer
                :method 'lff
                :time (time ?req)
                :space (space ?req)
                :accuracy (accuracy ?req))))
```

10 *A FROB Forward Chaining Rule*

lower level ones for sophisticated system operations. The lowest level utility functions should have a maximal amount of flexibility since it is not known what or how more powerful constructs built upon them will be used. In the prototype system they are implemented as methods attached to FROB classes which define major components of the system. These classes and their methods form the templates from which application systems are synthesized. The application specific rules use knowledge of these templates to apply the line interpretation rules by relying on the transparent nature of the methods to handle lower level hardware or operating system specific tasks.

An example is the FROB representing the class of cameras. Knowledge about operating this class of camera is represented in a "run" method which is local to the class. It executes operating system commands which are not of concern to the object using the method. A "run" method is also provided to other sensors which have other operating system commands which are transparent to the caller of the method.

3.1.2. Language Issues

Since the application system is created from FROBS in the same environment as the knowledge-based system, the application system runs in the Common Lisp environment. To require that all of the algorithms in the system be written in Common Lisp would be a severe restriction to its flexibility. The object-based approach allows algorithms written in any language to be incorporated into the system as an algorithm object.

Methods are used to run the algorithm and provide it with the necessary I/O. Since the internal representation of the object is transparent to I/O from the outside, algorithms written in any language can be incorporated into the system as long as there are low level utilities in the system to support the methods which run them.

3.1.3. Object Communication Protocol

When designing a CBCV vision system using objects, there must be a well defined way for one object sensor to pass information to another. Logical sensors address this problem in an abstract sense, but a specific protocol must be chosen which has the flexibility to accept all kinds of data. The protocol is represented in the slots and methods of the logical sensor objects. There must be a way to pass information from machine to machine as well as an efficient way to pass information in the Lisp environment itself. We separate the two as different types of information passing, file piping and S-expression passing.

Passing S-expressions between objects is a trivial task. All that is required is a slot in the algorithm object which stores the expression to be passed. This slot is read by any object requiring the expression as input. To perform file piping on any host in the system, the simplest approach is to use the Unix pipe facility which allows executables to work as filters passing their output to the next program in the pipe. This is the easiest implementation since it is supported by the remote shell command "rsh" which is used to perform tasks on remote machines. It requires, however, that most programs written for the knowledge-based system be written in filter form on machine supporting Unix. This is not an unreasonable requirement since Unix is a widely supported operating system and it is good modular style to have a system designed with filters. Other programs can be run as well as filters although it is up to the user to supply names and flags in slots of the object which contains the program. Only filters are handled "automatically" by the piping method.

At some point in the CBCV system's operation, information from a remote machine will have to be read by an object in the Lisp environment or vice-versa. This requires some special processing on the part of the methods performing the pipe. To send the S-expression output of a Lisp algorithm to a non-Lisp algorithm, certain conventions must be adopted. The program receiving the S-expression must know that its input is in such a form. Each algorithm in the knowledge base must have information regarding what format it expects its input to be in and what format is produced as output. This is done with methods using slot information in the algorithm object. These methods determine what format conversions are necessary for information piped between algorithms. Information transfer between machines is performed when objects have slots indicating that their executables are on different machines.