Multisensor Knowledge Systems[1]

Thomas C. Henderson and Chuck Hansen

Computer Science Department
University of Utah
Salt Lake City, Utah 84112 USA

UUCS-86-114

2 September 1986

Abstract

We describe an approach which facilitates and makes explicit the organization of the knowledge necessary to map multisensor system requirements onto an appropriate assembly of algorithms, processors, sensors, and actuators. We have previously introduced the Multisensor Kernel System and Logical Sensor Specifications as a means for high-level specification of multisensor systems. The main goals of such a characterization are: to develop a coherent treatment of multisensor information, to allow system reconfiguration for both fault tolerance and dynamic response to environmental conditions, and to permit the explicit description of control.

In this paper we show how Logical Sensors can be incorporated into an object-based approach to the organization of multisensor systems. In particular, we discuss:

* a multisensor knowledge base,

* a sensor specification scheme, and

* a multisensor simulation environment.

We give an example application of the system to CAD-based 2-D vision.

Control Commands

X

Logical
Sensor Name

Selector

Program 1 ... Program n

Control
Command
Interpreter

... ...

Logical Sensor
Inputs
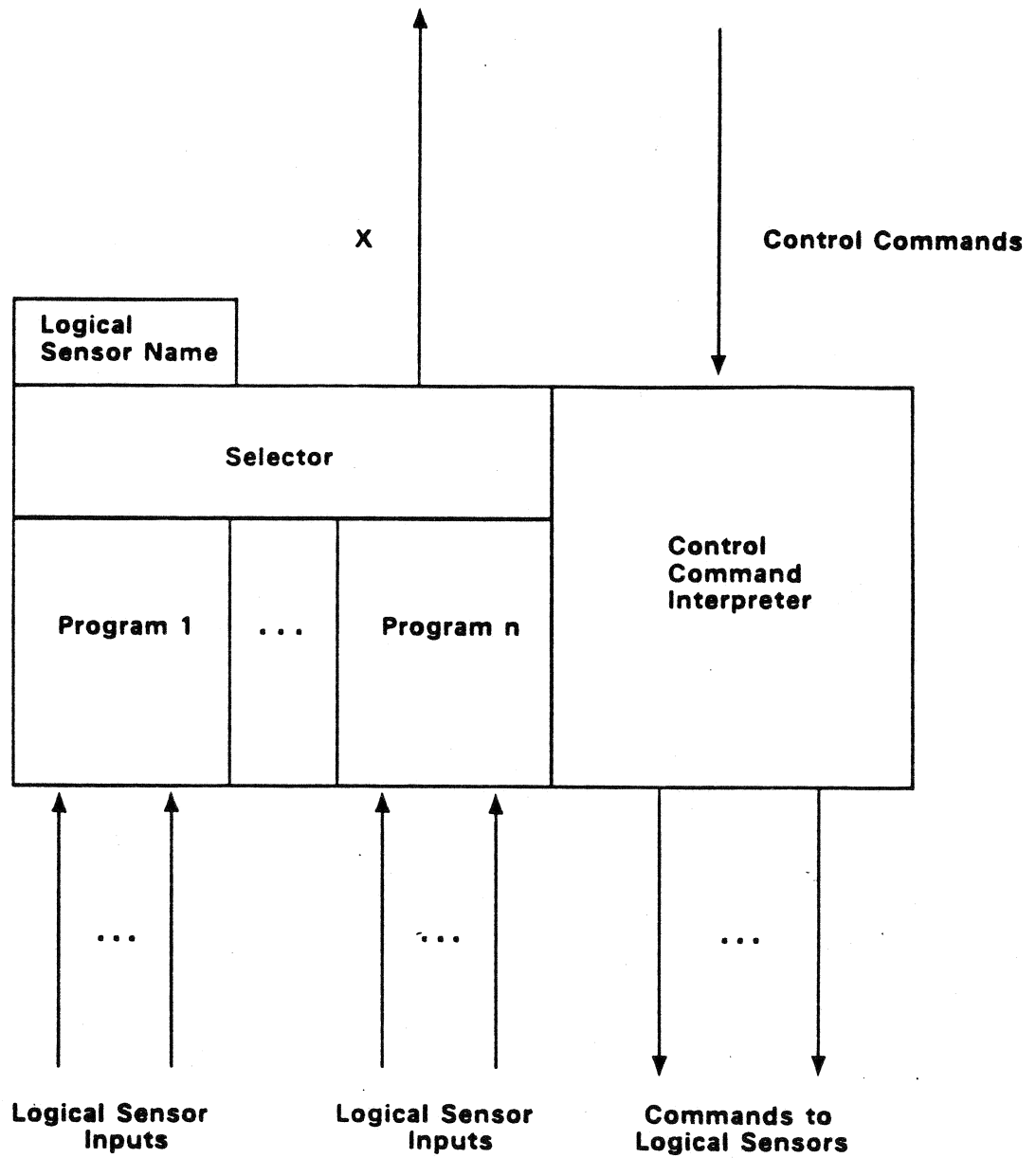
Logical Sensor
Inputs

Commands to
Logical Sensors

Figure 1. Logical Sensor Specification Building Block:
The Logical Sensor

must determine if the control commands require the execution of a different program to compute the characteristic output vector (i.e., whether dynamic reconfiguration is necessary).

Logical Sensor Specifications are useful then for any system composed of several sensors, where sensor reconfiguration is required, or where sensors must be actively controlled. The principle motivations for Logical Sensor Specifications are the emergence of significant multisensor and dynamically controlled systems, the benefits of data abstraction, and the availability of smart sensors.

In previous papers we have explored several issues of multisensor integration in the context of Logical Sensor Specifications:

* fault tolerance [8],

* functional (or applicative) style programming [16],

* features and their propagation through a network [17],

* the specification of distributed sensing and control [9, 10],

* the automatic synthesis of Logical Sensor Specifications for CAD/CAM applications [11, 12].

Related work includes that of Albus [1] on hierarchical control, Bajcsy et al. [2] on the Graphical Image Processing Language, Overton [15] on schemas, and Chiu [4] on functional language and multiprocessor implementations. For an overview of multisensor integration, see Mitiche and Aggarwal [13].

In exploring these issues, we have found that the specification of multisensor systems involves more than just sensor features. It is true that knowledge must be available concerning sensors, but it is essential to also be able to describe algorithms which use the sensor data and the hardware on which they are executed. In the rest of the paper, we describe the components of an object-based approach to developing a knowledge system to support these requirements.

2. Objects and Methods

Several distinct programming styles have been developed over the last few years, including:

* **applicative-style programming,**

* **control-based programming,**

* **logic programming,** and

* **object-based programming.**

Applicative style programming exploits function application as its main operation and regulates quite strongly the use of side-effects [6]. Historically, however, control-based programming has been the most extensively used paradigm, and focuses on the flow of control in a program. Logic programming is based on logical inference and requires the definition of the formal relations and objects which occur in a problem and the assertion of what relations are true in the solution. On the other hand, many current systems are being developed which are based on the notion of objects; this style emphasizes data abstraction combined with message passing [3, 14].

In the control-based style a program is viewed as a controlled sequence of actions on its total set of data structures. As the complexity of a system grows, it is hard to keep a clear picture of the entire sequence of actions that make up the program. This leads to the chunking of sequences into subprograms, and this is almost exclusively done for control purposes. But data structures are not decomposed into independent entities. In fact, most global data structures are shared by all subroutines.

On the other hand, the object-based style takes the view that the major concern of programming is essentially the definition, creation, manipulation and interaction of objects; that is, a set of independent and well-defined data structures. In particular, a single data structure (or instance) is associated with a fixed set of subprograms (methods), and those subprograms are the only operations defined on that object.

Such a use of data abstraction leads to design simplification which in turn makes the program more understandable, correct, and reliable. In addition, flexibility and portability are enhanced since details of objects (i.e., their representations) are hidden and can be

implemented in other ways without changing the external behavior of the object.

For our purposes, an object consists, essentially, of three parts:

1. <u>unique name</u>: this name must be distinguished from all other names in both time and space,

2. <u>type</u>: an object is an instance of a type which defines the valid set of operations and which details the nature of the resource represented, and

3. <u>representation</u>: the representation contains the information content associated with an object. This may include private data structures, references to other objects, etc.

Thus, an object is a structure with internal state (perhaps called *slots* and comprised of name/value relationships) accessed through functions (also called *methods*) defined in association with the object. This approach makes management schemes simpler and fewer, easier to implement and use; in addition, individual resources are easier to specify, create (allocate), destroy (deallocate), manipulate and protect from misuse.

It has been effectively argued many times that object-based programming is well-suited to embedded systems processing requirements. In particular, the application of this methodology to the specification of sensor systems helps to directly describe most of the important aspects of such systems:

* parallel processing,

* real-time control,

* exception handling, and

* unique I/O control.

Sensors typically require such operations as: enabling/disabling, limit setting, status checking, and periodic logging of state. That is, sensor systems must respond to out-of-limit readings and issue alarms, detect faulty sensors, and recover from failure, and these functions can be implemented in a straightforward way.

3. Multisensor Knowledge Systems

Much of our previous work on multisensor systems has concentrated on the specification of such systems and reasoning about their properties. It is necessary to be able to describe both the parameters and characteristics of individual components of multisensor systems, and to be able to deduce global properties of complete systems. Although it may be possible to deduce such properties (especially static properties like complexity, data type coercion, etc.), we believe that many interesting properties can only be determined by simulating the operation of the complete system.

Thus, we seek a representation that supports:

1. **multisensor system specification**: this describes the components and interconnection scheme of the particular system being designed,

2. **sensor, algorithm, processor and actuator knowledge representation**: this structures information about sensor characteristics (e.g., accuracy, hysteresis, dynamic range, etc.), algorithms (e.g., space and time complexity, amenity to parallel computation, stability, etc.) processors (e.g., cycle times, memory limits, address space, power requirements, etc.), and actuators (e.g., actuation principle, power requirements, etc.), and

3. **multisensor system simulation**: this permits one to monitor important parameters and to evaluate system performance.

Figure 2 shows the organization of the three capabilities within an object-oriented context. In the following subsections, we describe the Multisensor Knowledge System (MKS), an object-based approach to providing a unified answer to these three capabilities.

3.1. The Multisensor Knowledge Base

The multisensor knowledge base serves two main purposes:

1. to describe the properties of the system components (e.g., sensors, algorithms, actuators and processors), and

2. to provide class descriptions for the actual devices which are interconnected in any particular logical sensor specification.

That is, the knowledge base must describe not only generic sensors (e.g., cameras), but specific sensors (e.g., Fairchild 9000, Serial No. 28753). It is then possible to reason about sensor systems at several levels. Moreover, it is possible that two distinct specifications require some of the same physical sensors. In such a case, it is the responsibility of the

Algorithm

Segment   Edge
          Finder

Sobel   Canny

Sensor

Tactile   Visual

2-D camera   Range
             Sensor

Processor

Special   General
Purpose   Purpose

Vision   Array
System   Processor

Multisensor
Knowledge
Base

Sensor System
Specification

| Sobel Subroutine | Fairchild 3000 CCD | VICOM |

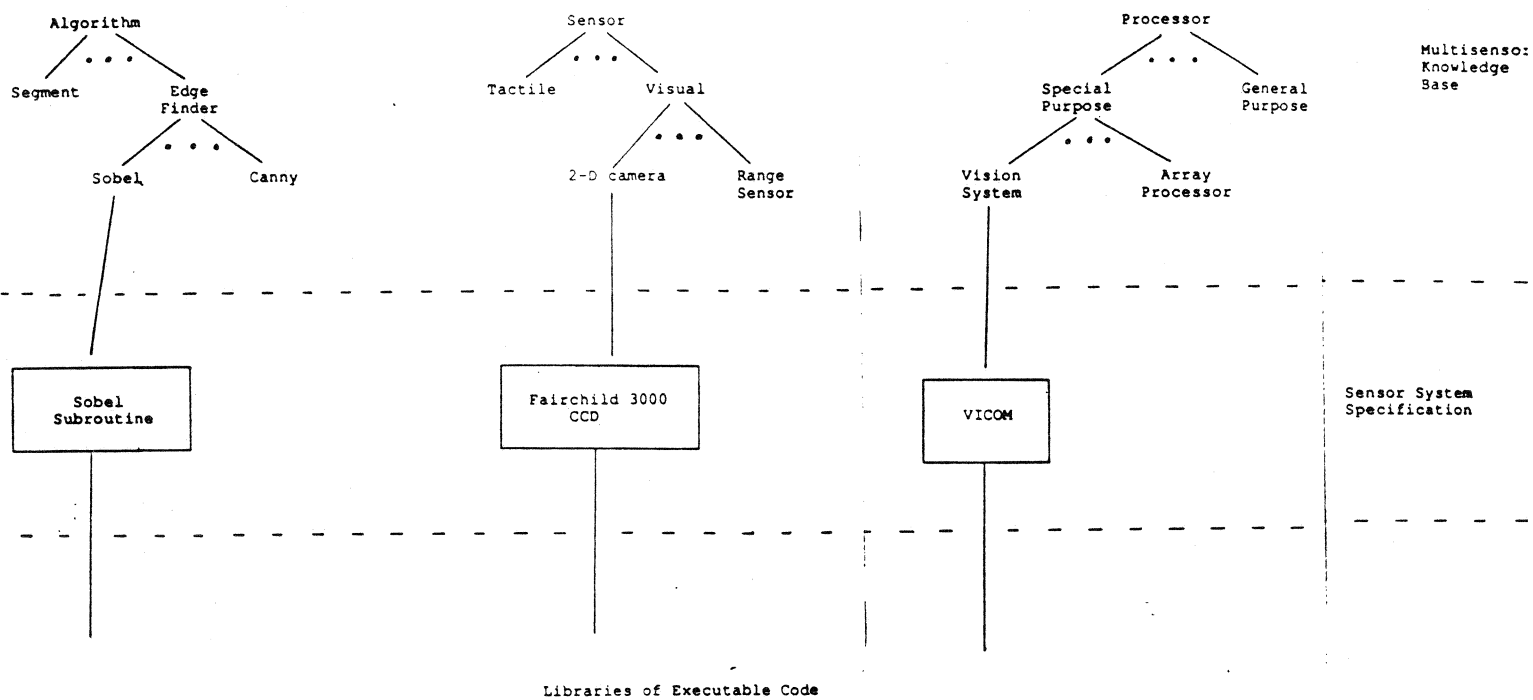Libraries of Executable Code

Figure 2 Multisensor Knowledge System

execution environment to resolve resource allocation conflicts.

We have chosen a frame-like knowledge representation. Frames relate very naturally to object-based descriptions, and, in fact, can be viewed as a class of restricted objects. It is straightforward to provide hierarchical descriptions of system components. For example, Figure 3 shows the *CCD Camera* hierarchy. The *CCD Camera* frame has two slots: element spacing and aspect ratio. These slots are specific to CCD cameras and as such do not appear as slots for *2-D cameras*. These latter have slots for scanning format, scan timing, resolution, output signal, and operating conditions. These slots are inherited by any instance of CCD camera. One level up, we find a frame for *Vision* sensors. This frame has specific slots for the spectral band and for the output type (e.g., 2-D byte array, multi-band, etc.). At the highest level of the hierarchy is the *Sensor* frame which has a slot for the physics of operation. This slot is used by any particular sensor to allow for an explanation of the physics behind the workings of the sensor. In this way, if reasoning is required about the sensor, it is possible to look in this slot for information. As can be seen, knowledge is organized such that there are more specific details lower in the hierarchy.

Note that frames are themselves implemented as objects. Thus, actual devices are instances of some class of objects. This is very concise and conveniently exploits the similarities of frames and objects.

In previous work, we have described a set of generally applicable physical sensor features [7]. The manner in which physical sensors convert physical properties to some alternative form, i.e., their transducer performance, can be characterized by: error, accuracy, repeatability, drift, resolution, hysteresis, threshold, and range. These properties can be encoded in the appropriate slots in the frames describing the sensor.

3.2. Sensor Specification

An object-based style of programming requires that the logical sensor of Figure 1 be re-described in terms of objects and methods. We shall next give the general flavor of this style, but it must be remembered that any particular sensor is actually an instance of some object class, and, in fact, inherits properties from many levels up.

Each logical sensor is completely specified as shown in Figure 4. Thus, in order to get

SENSOR

    * physics of operation

VISION

    * spectral band
    * output type

2-D CAMERA

    * scanning format
    * scan timing
    * resolution
    * output signal
    * operating conditions

CCD

    * element spacing
    * aspect ratio
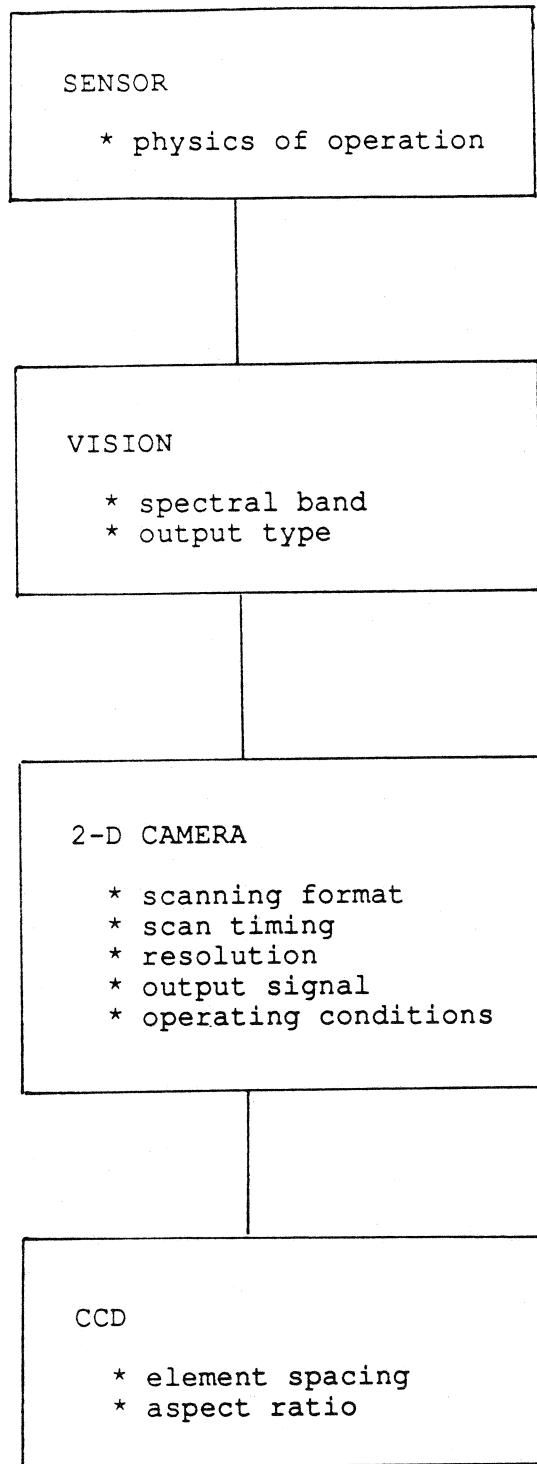
Figure 3 Organization of Camera Knowledge

```
┌─────────────────────┐      ┌─────────────────┐           Methods on
│ Characteristic      │      │ Control         │           Logical Sensors
│ Output Vector       │      │ Commands        │
└─────────────────────┘      └─────────────────┘
           ▲                          ▲
           │                          │
           │                          │
           ▼                          ▼
┌───────────────────────────────────────────────┐
│                                                │          The
│   Logical Sensor Name                          │          Logical
│                                                │          Sensor
├────────────────────────────────────────────────│          Object
│                                                │
│   Selector                                     │
│                                                │
├────────────────────────────────────────────────│
│                                                │
│   Description                                  │
│                                                │
└───────────────────────────────────────────────┘
```
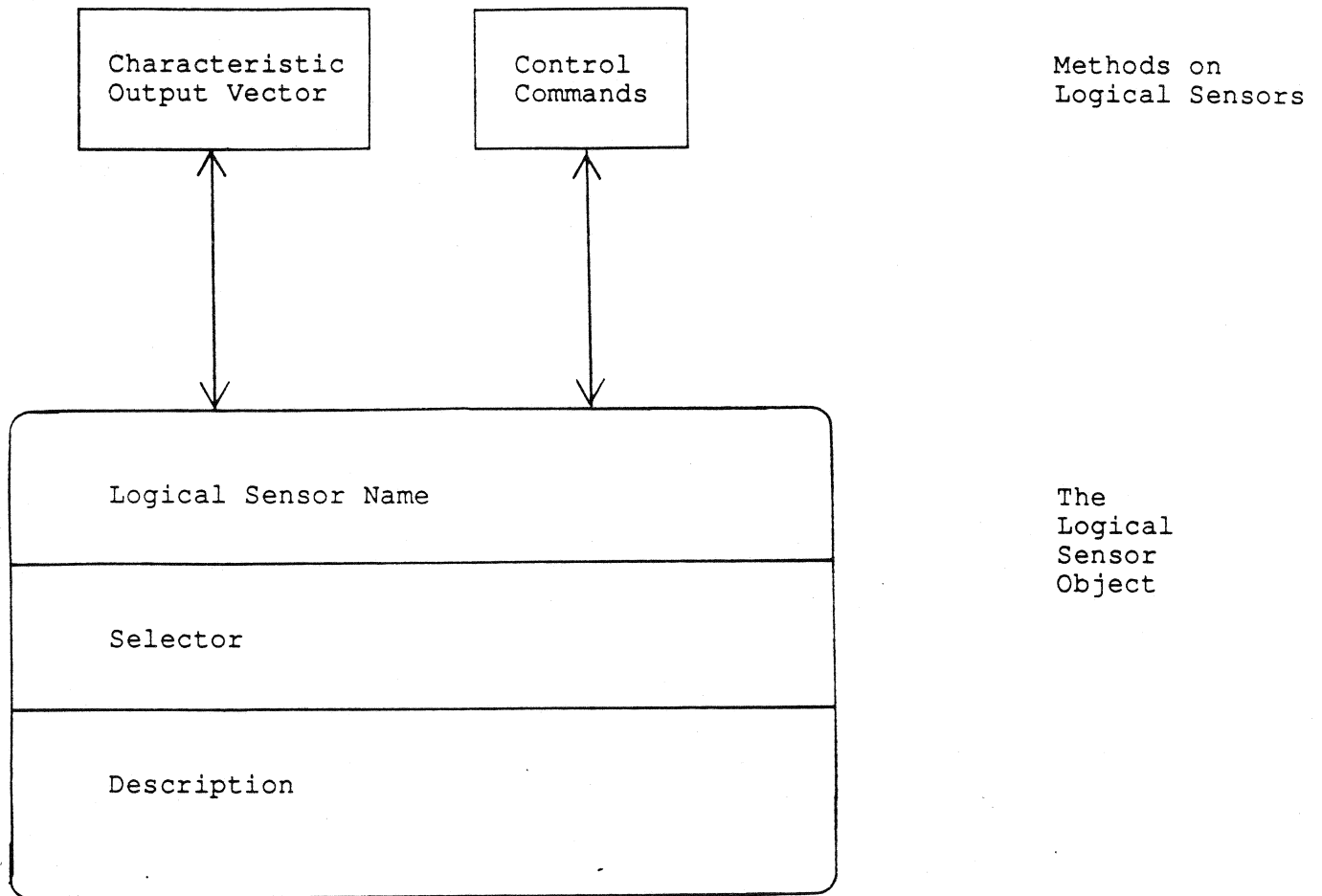
Figure 4. The Logical Sensor Object and Methods

data from a logical sensor, the *characteristic output vector method* must be invoked. Likewise, to issue control commands to the sensor (e.g., camera pan and tilt parameters), the *control commands method* must be used. The role of the *selector* is still the same as in previous logical sensor implementations, however, it now, in essence, is invoked to produce the characteristic output vector.

Such a representation makes it very easy to design sensor systems. Moreover, such specifications allow for replacement of sensors and dynamic reconfiguration by simply having the *selector* send messages to different objects. Given current object-based programming technology, such systems can be rapidly developed and permit dynamic typechecking (on objects).

Figure 5 shows the Multisensor Knowledge Base, and below the dashed line, a set of particular instances of various algorithms, sensors, etc. (drawn as circles). A logical sensor specification (indicated as a blocked in subset of the circles) defines a grouping of algorithms, sensors, etc. This newly created logical sensor is an instance of the *logical sensor object* and can be sent messages. As mentioned above, there are two methods defined on logical sensors: the *characteristic output vector method* and the *control commands method*. Thus, any logical sensor can be defined recursively in terms of other logical sensors (including itself).

Currently, our main interest is in the automatic synthesis of logical sensor specifications. Given a CAD model of an object, we would like to synthesize a specific, tailor-made system to inspect, recognize, locate or manipulate the object. Note that the synthesis of a logical sensor specification consists, for the most part, of interconnecting instances of sensors and algorithms to perform the task. This is done by writing the selector to invoke methods on other logical sensors. Given certain constrained problems, most notably the CAD/CAM environment, such a synthesis is possible.

3.3. The Simulation of Multisensor Systems

Effective simulation plays an important role in successful system development. A key requirement is the support for hierarchical specification of the system and the ability to perform stepwise refinement of the system. In addition, it is necessary to be able to efficiently emulate realtime software that will eventually be embedded in the system.

Algorithm

Segment   Edge
       Finder

Sobel   Canny

Sensor

Tactile   Visual

2-D camera   Range
        Sensor

Processor

Special   General
Purpose   Purpose

Vision   Array
System   Processor

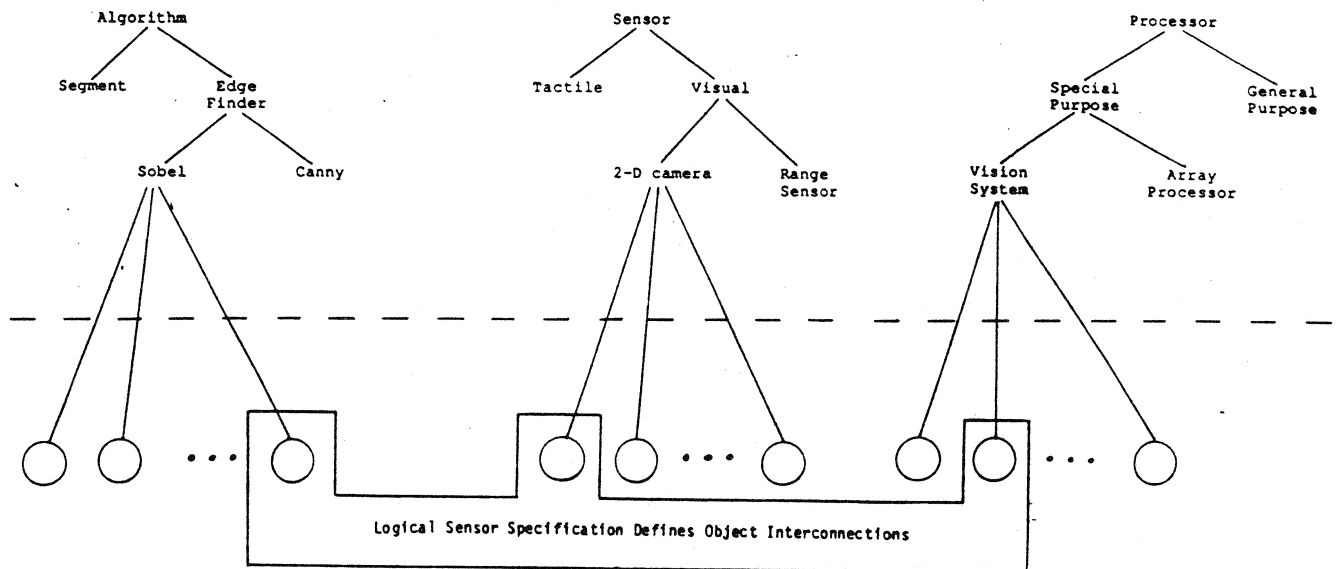Logical Sensor Specification Defines Object Interconnections

Figure 5. Logical Sensor Specification Using Object Instances

Finally, it would be quite useful to be able to embed physical components in the simulator in order to monitor the system's operation.

An object-oriented simulation methodology is well-suited to satisfy these goals. The multisensor system, that is, the system being modeled, consists of a collection of interacting physical processes. Each such process is modeled in the simulator by an object, i.e., an independent process. Interactions among physical processes are modeled through messages exchanged among the corresponding objects.

This general paradigm is currently supported in the SIMON simulator developed by Fujimoto [5, 18]. A toolkit approach is used in which the simulator is viewed as a collection of mechanisms from which application specific simulation environments are constructed. We are currently exploring the simulation of multisensor systems in the SIMON environment. Simulation can be accomplished by substituting simulation libraries for the run time libraries (see Figure 1).

A crucial aspect of the simulation is the ability to execute specific algorithms on specific hardware. SIMON permits such a direct execution technique in which application programs are executed directly on a host processor rather than through a software interpreter. Performance information is obtained through the automatic insertion of probes and timing software into the program at compile time. These probes perform whatever runtime analysis is required to accurately estimate execution time of basic blocks of code. A prototype implementation using this technique has been developed modeling the MC68010 and 68020 microprocessors. Initial data indicate that application programs may be emulated one to two orders of magnitude more efficiently over traditional register transfer level simulation, while highly accurate performance estimates can still be obtained.

4. An Example Application: CAD-Based 2-D Vision

A simple example which demonstrates some of the power of the Multisensor Knowledge System approach is that of CAD-Based 2-D Vision. The goal is to automate visual inspection, recognition and localization of parts using pattern recognition techniques on features extracted from binary images. Figure 6 shows the scheme pictorially.

The Multisensor Knowledge System stores knowledge about the algorithms, sensors,
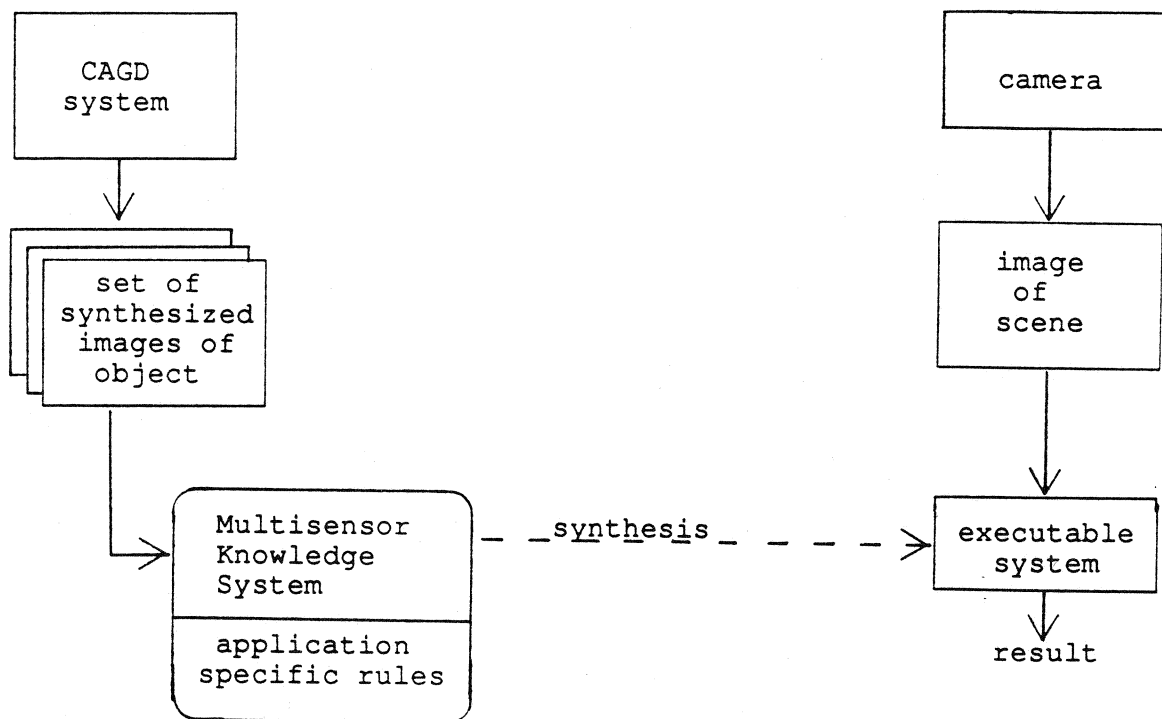
```
┌──────────┐                              ┌──────────┐
│  CAGD    │                              │  camera  │
│  system  │                              │          │
└────┬─────┘                              └────┬─────┘
     │                                         │
     ▼                                         ▼
 ┌──────────┐                             ┌──────────┐
┌┤ set of   │                             │ image    │
│││synthesized                            │ of       │
│││images of │                            │ scene    │
└┤│ object   │                            └────┬─────┘
 └────┬─────┘                                  │
      │                                        │
      │   ┌────────────────┐                   ▼
      └──▶│ Multisensor    │            ┌──────────────┐
          │ Knowledge      │─ _synthesis_ _ _ _ ▶│ executable  │
          │ System         │            │ system      │
          ├────────────────┤            └──────┬──────┘
          │ application    │                   │
          │ specific rules │                   ▼
          └────────────────┘                result
```

Figure 6. Synthesis of Part Detector

processors, etc. This knowledge is used by application specific rules. The systems to be synthesized here require that a model be created for the part to be inspected, and that a robust and (perhaps) independent set of features be chosen along with an appropriate distance metric.

The left side of the figure shows the offline training component. The new part is designed using a Computer Aided Geometric Design system. A set of images are rendered by the CAGD system giving a sample of various views of the part in different positions, orientations, and scales. These serve as a training set to the Multisensor Knowledge System.

A set of rules (or productions) performs an analysis of the views of the part to select a subset of the total set of possible features. Features are used if they are robust, independent and reliable. Once these features have been chosen, a new logical sensor object is created whose only function is to recognize the given part based on an analysis of the selected features. The part detector is then linked into a particular application (e.g., an inspection task at a specific workcell) by sending a message to the appropriate camera.

As a specific example, consider the object shown in Figure 7. It was designed with Alpha_1, an experimental solid modeling system developed at the University of Utah. For the past few years the Computer Aided Geometric Design group has been involved in a concerted effort to build this advanced modeler. Alpha_1 incorporates sculptured surfaces and embodies many theoretical and algorithmic advances. 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 B-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 for computing discrete B-splines. Subdivision, effected by the Oslo algorithm, supports various capabilities including the computation associated with Boolean operations, such as the intersection of two arbitrary surfaces. B-splines are an ideal design tool, they are simple, yet powerful. It is also the case that 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
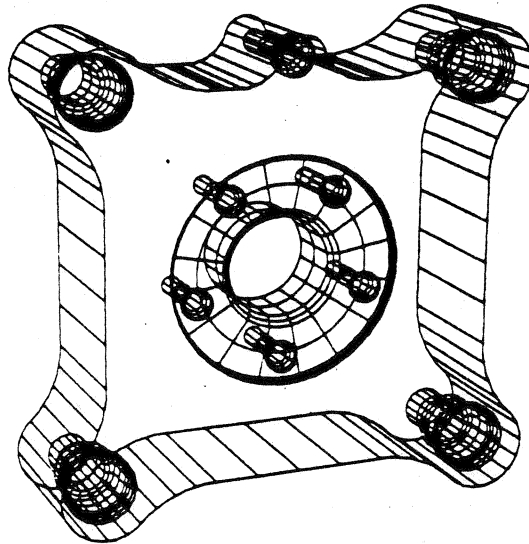
Figure 7. The Green Piece

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).

The object shown in Figure 7 was rendered at orientations of 0, 22.5 and 45 degrees. These images were analyzed. The set of possible features included:

> Area
> Perimeter to Area Ratio
> Perimeter
> X-Y Aspect Ratio
> Diameter
> Center of Mass in X
> Center of Mass in Y
> Normalized Moment 1

References

[1]   Albus, J.
        Brains, Behavior and Robotics.
        BYTE Books, Peterborough, New Hampshire, 1981.

[2]   Bajcsy, R.
        GRASP:NEWS Quarterly Progress Report.
        Technical Report Vol. 2, No. 1, The University of Pennsylvania, School of
            Engineering and Applied Science, 1st Quarter, 1984.

[3]   Booch, Grady.
        Software Engineering with Ada.
        Benjamin/Cummings Publishing Co., Menlo Park, california, 1983.

[4]   Chiu, S.L., D.J. Morley and J.F.Martin.
        Sensor Data Fusion on a Parallel Processor.
        In *Proceedings of the IEEE Conference on Robotics and Automation,* pages
            1629-1633. San Francisco, CA, April, 1986.

[5]   Fujimoto, R.M.
        The SIMON Simulation and Development System.
        In *Proceedings of the 1985 Summer Computer Simulation Conference,* pages
            123-128. July, 1985.

[6]   Henderson, T., E. Triendl and R. Winter.
        Model-Guided Geometric Registration.
        Technical Report NE-NT-D-50-80, Deutsche Forschungs- und Versuchsanstalt fuer
            Luft- und Raumfahrt, September, 1980.

[7]   Henderson, T.C. and E. Shilcrat.
        Logical Sensor Systems.
        Journal of Robotic Systems 1(2):169-193, 1984.

[8]   Henderson, T.C., E. Shilcrat and C.D. Hansen.
        A Fault Tolerant Sensor Scheme.
        In *Proceedings of the International Conference on Pattern Recognition,* pages
            663-665. August, 1984.

[9]   Henderson, T.C., C.D. Hansen, and Bir Bhanu.
        The Specification of Distributed Sensing and Control.
        Journal of Robotic Systems 2(4):387-396, 1985.

[10]   Henderson, T.C., Chuck Hansen and Bir Bhanu .
        A Framework for Distributed Sensing and Control.
        In *Proceedings of IJCAI 1985,* pages 1106-1109. Los Angeles, CA, August, 1985.

[11]   Henderson, T.C. and Steve Jacobsen.
        The UTAH/MIT Dextrous Hand.
        In *Proceedings of the ADPA Conf. on Intelligent Control Systems,* pages to
            appear. Ft. Belvoir, Va., March, 1986.

[12]   Henderson, T.C., Chuck Hansen, Ashok Samal, C.C. Ho and Bir Bhanu.
        CAGD Based 3-D Visual Recognition.
        In *Proceedings of the International Conference on Pattern Recognition*, pages to
            appear. Paris, France, October, 1986.

[13]   Mitiche, A. and J.K. Aggarwal.
        An Overview of Multisensor Systems.
        SPIE Optical Computing 2:96-98, 1986.

[14]   Organick, E.I., M. Maloney, D. Klass and G. Lindstrom.
        *Transparent Interface between Software and hardware Versions of Ada
            Compilation Units.*
        Technical Report UTEC-83-030, University of Utah, Salt Lake City, Utah, April, 1983.

[15]   Overton, K.
        Range Vision, Force, and tactile Sensory Integration: Issues and an Approach.
        In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1463.
            San Francisco, California, April, 1986.

[16]   Shilcrat, E., P. Panangaden and T.C. Henderson.
        Implementing Multi-sensor Systems in a Functional Language.
        Technical Report UUCS-84-001, The University of Utah, February, 1984.

[17]   Shilcrat, E.
        Logical Sensor Systems.
        Master's thesis, University of Utah, June, 1984.

[18]   Swope, S.M. and R.M. Fujimoto.
        SIMON II Kernel Reference Manual.
        Technical Report UUCS-86-001, University of Utah, May, 1986.