Logical Sensor Specification[1]

Charles Hansen, Thomas C. Henderson, Esther Shilcrat and Wu So Fai
Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

Abstract

Multi-sensor systems require a coherent and efficient treatment of the information provided by the various sensors. We propose a framework in which the sensors can be abstractly defined in terms of computational processes operating on the output from other sensors. Various properties of such an organization are investigated.

1 Introduction

Most pattern recognition systems to date have been designed around a single sensor or a small number of sensors, and ad hoc techniques have been used to integrate them into the complete system and for operating on their data. In the future, however, systems must operate in a reconfigurable multi-sensor environment; for example, there may be several cameras (perhaps of different types), active range finding systems, tactile pads, etc. The **M**ulti-sensor **K**ernel **S**ystem (MKS) has been proposed as an efficient and uniform mechanism for dealing with data taken from several diverse sensors[3, 4, 1]. MKS has three major components: low-level data organization, high-level modeling, and logical sensor specification. This paper addresses the problem of logical sensor specification.

The specification of logical graphics input devices has been investigated by Foley [2], Pfaff [9], and Rosenthal [10], and we have shown how MKS can provide similar functions in a multi-sensor environment. Basically, a logical (or abstract) sensor can be viewed as a software filter whose inputs are either system defined devices (such as cameras) or the output streams of other logical sensors. Thus, MKS supports a mix of

[1]This work was supported in part by the System Development Foundation and NSF Grants ECS-8307483 and MCS-82-21750.

hardware and software sensors in straightforward way. The logical sensor specification defines a dependency relation and permits efficient system reconfiguration in case of the failure of a particular sensor. Such a sensor specification is in fact a sensor definition language, and thus, a semantics of sensor definitions can be given, providing the user a useful analysis tool.

There are several principal motivations for logical sensor specification:

- *emergence of multi-sensor systems*: many applications require several sensors; e.g., robotics workcells, distributed problem solving networks, etc. Often, the sensors are of diverse types (cameras, tactile pads, active range finders), and a coherent data acquisition and integration system is required.

- *benefits of data abstraction*: the specification of a sensor is separated from its implementation. The multi-sensor system is then much more portable in that the specifications remain the same over a wide range of implementations. Moreover, alternative mechanisms can be specified to produce the same sensor information but perhaps with different precision or at different rates. Thus, several dimensions of sensor granularity can be defined. Further, the stress on modularity not only contributes to "intellectual manageability" [11] but is also an essential component of the system's reconfigurable nature. The inherent hierarchical structuring of logical sensors further aids system development.

- *availability of smart sensors*: the lowering cost of hardware combined with developing methodologies for the transformation from high level algorithmic languages to silicon have made possible a system view in which hardware/software divisions are transparent. It is now possible to incorporate fairly complex algorithms directly into hardware. Thus, the substitution of hardware for software (and vice versa) should be transparent above the implementation level.

In the remainder of this paper, we present some of the advantages and disadvantages of logical sensor specification, but it should be remembered that this is only one part of the Multi-sensor Kernel System, and that the specification of logical sensors must accord with the rest of the system.

2 The Data Flow View of Logical Sensors

Before we give a detailed syntactic description of a logical sensor specification language, an overview of logical sensor specification is in order. A logical sensor is defined in terms of three parts:

1. A set of input sources. Each element of the set must either be itself a logical sensor, or the empty set.

Figure 1: Graphical View of a Logical Sensor

2. A computation unit over the input sources. Currently such computation units are software programs, but in the future, hardware units may also be used. The examples of computation units in this paper will be programs.

3. A characteristic output vector. This is basically a vector of types which serves as a description of the output vectors that will be produced by the logical sensor. Thus, the output of a logical sensor is a stream of vectors, each of which is of the type declared by that logical sensor's characteristic output vector. When an output vector is of the type declared by a characteristic output vector (i.e., the cross product of the vector element types), we say that the output vector is an "instantiation" of that characteristic output vector.

Alternatively, we present the following inductive definition of a logical sensor, where 1. is the base case:

1. A computation unit, with specified output type (the characteristic output vector), which requires no input sources.

2. A computation unit, with specified output type, whose input sources are logical sensors.

Figure 1 gives a graphical presentation of this notion. The characteristic output vector declared for this logical sensor is (x-loc:real, y-loc:real, z-loc:real, curvature:integer). We present two examples to clarify the definition of logical sensors, and in particular to show how the inputs to a logical sensor are defined in terms of other logical sensors and how the program accepts input from the source logical sensors, performs some computation on them and returns as output a set (stream) of vectors of the type defined by the characteristic output vector. Figure 2 shows the logical sensor specification for a *Camera* which happens to have no other logical sensor inputs. The specification for

Figure 2: The Logical Sensor Specification of a *Camera*

Figure 3: The Logical Sensor Specification of *Range_Finder*.

a stereo camera range finder called *Range_Finder* is given in Figure 3. The program *stereo* takes the output of the two cameras and computes vectors of the form (x,y,z) for every point on the surface of an object in the field of view.

The idea is that a logical sensor can specify either a device driver program which needs no other logical sensor input, but rather gets its input directly from the physical device and then formats it for output in a characteristic form, or a logical sensor can specify that the output of other logical sensors be routed to a certain program and the result packaged as indicated. Thus, logical sensors play the role of "software filters" which can be built as a useful set of modular tools in the system.

Having described how logical sensors are developed and operate, we now define a logical sensor to be a **network** composed of one or more sub-networks, where each sub-network is a logical sensor. The computation units of the logical sensor are the nodes of the network. The network forms a rooted directed acyclic graph. The graph is rooted because, taken entirely, it forms a complete description of a single logical sensor (versus, for example, being a description of two logical sensors which share sub-networks). We also say that it is rooted because there exists a path between each sub-network and a computation unit of the final logical sensor. Logical sensors may not be defined in terms of themselves, that is, no recursion is allowed, and hence the graph is acyclic.

All communication within a network is accomplished via the flow of data from one sub-network to another. No explicit control mechanism, such as the use of shared variables, alerts, interrupts, etc., is allowed. The use of such control mechanisms decreases the degree of modularity and independent operation of sub-networks. Hence the networks described by the logical sensor specification language are data flow networks, and have the following properties [8]:

1. A network is composed of independently, and possibly concurrently, operating sub-networks.

2. A network, or some of its sub-networks, may communicate with its environment via possibly-infinite input or output streams.

3. Sub-networks are modular.

3 A Syntax for Logical Sensor Specification

We have shown that a logical sensor has the following properties:

1. A logical sensor is a network composed of sub-networks which are themselves logical sensors. This composition is tantamount to saying that the input sources to a logical sensor are logical sensors, or null. Allowing null input permits **physical**

sensors which have only an associated program (the device driver) to be described as a logical sensor. Thus, uniformity of sensor treatment can be obtained.

2. A logical sensor may be defined only in terms of other, previously defined, logical sensors.

3. A computation unit is an integral part of the definition of a logical sensor.

4. A logical sensor produces output of the type declared by its characteristic output vector, and the declaration of the characteristic output vector is also an integral part of the definition of a logical sensor.

It should be noted that there may be alternate input paths, subtrees, to a particular sensor. These alternate subtrees form a different sub-network for a particular logical sensor. That is, the alternate subtrees provide a different input path should a node in the sub-network fail. Thus, there may be one or more paths through which a logical sensor produces data, but regardless of the path taken, the output will be of the type declared by the logical sensor's characteristic output vector.

With these points in mind, a syntax for describing the logical sensor system can be formed:

```
1. <logical_sensor_list>        ::= <logical_sensor>
                                        {;<logical_sensor>}.;

2. <logical_sensor>             ::= <logical_sensor_name>
            <characteristic_output_vector>
        <alternate_subtrees>;

3. <logical_sensor_name>        ::= identifier;

4. <characteristic_output_vector>     ::= <name_type_list>;

5. <name_type_list>                   ::= identifier : type
                                          {,<name_type_list>};

6. <alternate_subtrees>               ::= <input_list>
                                          <computation_unit_name>
                                          {|<alternate_subtrees>};

7. <input_list>                       ::= <> | <sensor_list>;
```

```
 8. <sensor_list>                    ::= <logical_sensor_name> [<index_list>]
                                         {,<sensor_list>};

 9. <index_list>                     ::= integer {,<index_list>};

10. <computation_unit_name>          ::= identifier;
```

We have developed both a user interface and a parser for this specification language. The user interface allows the user to interactively enter sensors into the system. The program prohibits the user from defining sensors based on non-existent sensor names but allows the user to see existing sensors, their various alternate sub-trees and the types of their output vectors . Furthermore, the program computes the dependency relation between sensors, and this relation can be used for dynamic reconfiguration. For example, should a sensor in the current subtree fail, the system could switch to an alternate subtree which provides the necessary input streams and therefore maintains the sensor's functionality. From this interactive sensor specification, a sensor program is produced which conforms to the syntax given above.

The parser takes a sensor program as given above and parses for syntactic correctness. The parser was developed using the compiler tools LEX and YACC on the UNIX (a registered trademark of Bell Labs) system. Currently, the parser simply checks the input language for syntax errors and does no semantic checking. Much of the semantics of the language is enforced through the interactive interface. For example, undefined sensor names, uniqueness of names, alternate subtrees, etc. are detected during input. This parser can be further used to produce machine independent interpretable code to define the manner in which the system should be run.

4 The Semantics of Logical Sensors

Below we present a high level description of the operational semantics (i.e., the execution effect) for each rule of the grammar:

1. A logical sensor list declaration establishes a series of logical sensor definitions as being known to the system. This is used to ensure that logical sensors are defined only in terms of other, previously defined, logical sensors.

2. A logical sensor declaration provides an associated name for the logical sensor used for identification purposes, a characteristic output vector to declare the type of output for that logical sensor, and establishes the subordinate logical sensors (the alternate subtrees).

3. A logical sensor name declaration associates a (unique) identifier for the logical sensor.

4. A characteristic output vector declaration establishes the type of output for the logical sensor.

5. A name-type list declaration establishes the precise nature of the output type as declared by the characteristic output vector. It consists of a cross product of types, with an associated name.

6. An alternate subtree declaration establishes a series of input sources, computation unit name tuples, thus making known which logical sensors and computation units are part of the definition of the logical sensor being declared.

7. An input list declaration establishes which legal input sources (either none, or a series of logical sensors) are to be used as input to the computation unit.

8. A sensor list declaration associates logical sensor names to an index list and is used to check that logical sensors specified as input sources have previously been declared.

9. An index list is a list of integers.

10. A computation unit name declaration establishes the name of the actual program which will execute on the declared input sources.

We are also currently working on providing more formal semantics for the logical sensor specification language. Many works provide denotational semantics (i.e., semantic schemes which associate with each construct in the language an abstract mathematical object) for general data flow networks [6, 7, 8]. When such semantics have been given for the networks represented by logical sensors, we will be able to formally prove desired network properties, such as [6]:

1. A network can execute forever.

2. If one of the sub-networks within a network stops at some time for an extraneous reason, the entire network would stop. (We say that a sub-network stops only if each of its alternate subtrees stops.)

We will also be able to prove that the output of a specified logical sensor has particular properties of interest (e.g., that its type matches that of the characteristic output vector).

5 The Multi-sensor Kernel System

The Multi-sensor Kernel System (MKS) provides more than just a mechanism for the specification of logical sensors. Our approach is to first take the vectors produced by the logical sensors and organize them into a useful low-level representation, the @b(spatial proximity graph). The nodes of the spatial proximity graph are the vectors themselves, and an edge exists between two nodes if the vectors they represent are within a specified distance. Many high-level modeling techniques can be supported by such a representation. Feature-based models can be directly matched to vectors from the sensors, but structural models will, in general, require some kind of processing of the spatial proximity graph [5, 1].

In the runtime environment, alternate subtrees combined with the dependency relation of the logical sensors provide a mechanism for efficient system reconfiguration should a sensor fail. A failed sensor can be handled by means of a device time-out interval, a failure bit propagated through the network, etc., and all sensors which depend on the failed sensor in their currently operating sub-network must be notified. These sensors can switch to an alternate subtree defined at the specification level. Moreover, the alternate subtree may be invoked at the closest hierarchical level, thus lessening the impact of the failed sensor. Through this scheme, dynamic system reconfiguration can be achieved.

6 Conclusion

The formal specification of logical sensors allows the organization of many diverse kinds of sensors and supports data abstraction. This increases the facility with which dynamic configuration of a sensor system can be achieved and hides implementation details. Thus, software functions can be migrated to hardware in a straightforward way. Moreover, within such a system it is possible for a higher level planning module to automatically specify logical sensors in order to obtain relevant sensor data in a systematic way.

References

[1] Wu So Fai. A multi-sensor integration and data acquisition system. Master's thesis, University of Utah, Salt Lake City, Utah, June 1983.

[2] J.D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

[3] Thomas C. Henderson and Wu So Fai. A multi-sensor integration and data acquisition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 274–280. IEEE, June 1983.

[4] Thomas C. Henderson and Wu So Fai. Pattern recognition in a multi-sensor environment. Computer Science UUCS 83-001, University of Utah, July 1983.

[5] Thomas C. Henderson and Wu So Fai. Some experiments with the 3-d hough shape transform. Computer Science UUCS 83-002, University of Utah, July 1983.

[6] G. Kahn. The semantics of a simple language for parallel programming. In *Proceedings of IFIP*, pages 471–475, 1974.

[7] G. Kahn and D. MacQueen. Coroutines and networks of parallel processes. In *Proc. IFIP 77*, pages 993–998, 1977.

[8] R.M. Keller. Denotational models for parallel programs with indeterminate operators. In E.J. Neuhold, editor, *Formal Descriptions of Programming Concepts*, pages 337–366. North Holland Publishing Co., 1978.

[9] H. Kuhlmann Pfaff, G. and H. Hanusa. Constructing user interfaces based on logical input devices. *Computer*, 15(11):62–69, November 1982.

[10] D.S. Rosenthal, J.C. Michener, G. Pfaff, R. Kessener, and M. Sabin. The detailed semantics of graphics input devices. *Computer Graphics*, 16(3):33–38, July 1982.

[11] N. Wirth. On the compostion of well-structured programs. In E.N. Yourdan, editor, *Classics in Software Engineering*, pages 153–172, London, 1979. Yourdon Press.