Instrumented Sensor System Architecture

Mohamed Dekhil and Thomas C. Henderson

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

Abstract

Sensor systems are becoming ubiquitous throughout society, yet their design, construction and operation are still more of an art than a science. In this paper, we define, develop, and apply a formal semantics for sensor systems that provides a theoretical framework for an integrated software architecture for modeling sensor-based control systems. Our goal is to develop a design framework which allows the user to model, analyze and experiment with different versions of a sensor system. This includes the ability to build and modify multisensor systems and to monitor and debug both the output of the system and the affect of any modification in terms of robustness, efficiency, and error measures. The notion of *Instrumented Logical Sensor Systems* (ILSS) that are derived from this modeling and design methodology is introduced. The instrumented sensor approach is based on a sensori-computational model which defines the components of the sensor system in terms of their functionality, accuracy, robustness and efficiency. This approach provides a uniform specification language to define sensor systems as a composition of smaller, predefined components. From a software engineering standpoint, this addresses the issues of modularity, reusability, and reliability for building complex systems. An example is given which compares vision and sonar techniques for the recovery of wall pose.

This work was supported in part by NSF grant CDA 9024721 and a gift from Hewlett Packard Corporation.

1 Introduction

In any closed-loop control system, sensors are used to provide the feedback information that represents the current status of the system and the environmental uncertainties. Building a sensor system for a certain application is a process that includes the analysis of the system requirements, a model of the environment, the determination of system behavior under different conditions, and the selection of suitable sensors. The next step in building the sensor system is to assemble the hardware components and to develop the necessary software modules for data fusion and interpretation. Finally, the system is tested and the performance is analyzed. Once the system is built, it is difficult to monitor the different components of the system for the purpose of testing, debugging and analysis. It is also hard to evaluate the system in terms of time complexity, space complexity, robustness, and efficiency, since this requires quantitative measures for each of these measures.

In addition, designing and implementing real-time systems are becoming increasingly complex because of many added features such as fancy graphical users interfaces (GUIs), visualization capabilities and the use of many sensors of different types. Therefore, many software engineering issues such as reusability and the use of COTS (Commercial Off-The Shelf) components (Profeta, 1996), real-time issues (Hu *et al.*, 1995; Schneider *et al.*, 1994; Simon *et al.*, 1993), sensor selection (Giraud & Jouvencel, 1994), reliability (Kapur *et al.*, 1996; Kim & Subbaraman, 1997; Stewart & Khosla, 1997), and embedded testing (Weller *et al.*, 1990) are now getting more attention from system developers.

In a previous paper, we proposed to use formal semantics to define performance characteristics of sensor systems (Dekhil & Henderson, 1996a). In this paper, we address these and other problems related to sensor system modeling and evaluation. We start by presenting a theoretical frame-

work for modeling and designing sensor systems based on a formal semantics in terms of a virtual sensing machine. This framework defines an explicit tie between the specification, robustness and efficiency of the sensor system by defining several quantitative measures that characterize certain aspects of the system's behavior. Figure 1 illustrates our proposed approach which provides static analysis (e.g., time/space complexity, error analysis) and dynamic handles that assist in monitoring and debugging the system.

1.1 Sensor Modeling

Each sensor type has different characteristics and functional description. Therefore it is desirable to find a general model for these different types that allows modeling sensor systems that are independent of the physical sensors used, and enables studying the performance and robustness of such systems. There have been many attempts to provide "the" general model along with its mathematical basis and description. Some of these modeling techniques concern error analysis and fault tolerance of multisensor systems (Brooks & Iyengar, 1993; Dekhil & Sobh, 1997; Iyengar & Prasad, 1994; Nadig *et al.*, 1993; Prasad *et al.*, 1991; Prasad *et al.*, 1994). Other techniques are model-based and require a priori knowledge of the scanned object and its environment (Durrant-Whyte, 1988; Groen *et al.*, 1993; Joshi & Sanderson, 1994). These techniques help fit data to a model, but do not provide the means to compare alternatives. Task-directed sensing is another approach to devise sensing strategies (Briggs & Donald, 1994; Hager & Mintz, 1991; Hager & Mintz, 1989), but again, it does not provide measures to evaluate the sensor system in terms of robustness and efficiency.

Another approach to modeling sensor systems is to define sensori-computational systems associated with each sensor to allow design, comparison, transformation, and reduction of any sen-

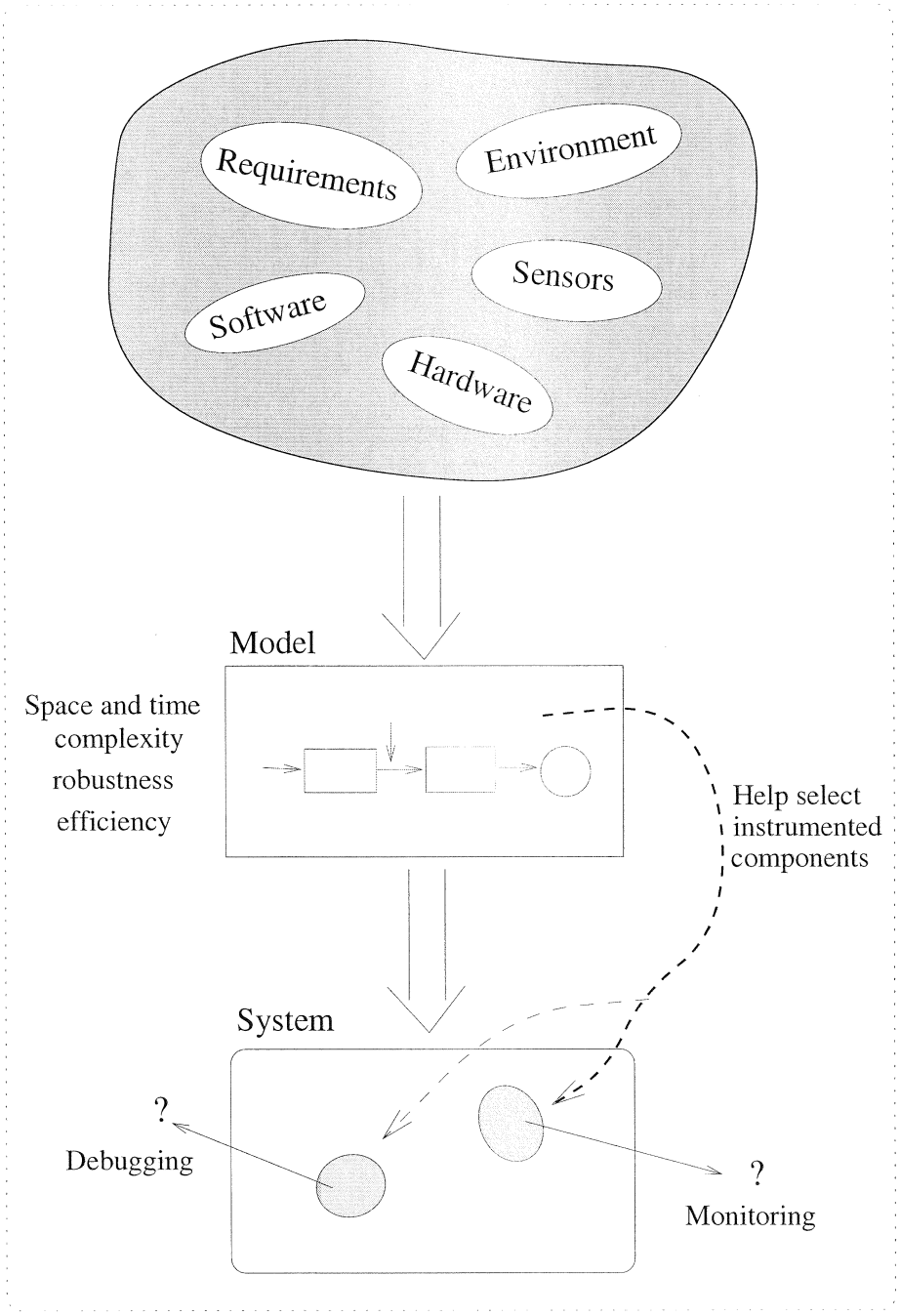

Figure 1: The proposed modeling approach.

sory system (Donald, 1995). In this approach the concept of information invariants is used to define some measure of information complexity. This approach provides a very strong computational theory which allows comparing sensor systems, reducing one sensor system to another, and measuring the information complexity required to perform a certain task. However, as stated by Donald, the measures for information complexity are fundamentally different from performance measures. Also, this approach does not permit one to judge which system is "simpler," "better," or "cheaper."

To that end, we introduce the notion of an *Instrumented Logical Sensor System* (ILSS) which represents our methodology for incorporating design tools and allows static and dynamic performance analysis, on-line monitoring, and embedded testing. Figure 2 shows the components of our framework. First (on the left), an Instrumented Logical Sensor Specification is defined, as well as $\mathcal{F}$, a set of functions which measure system properties of interest. This specification is derived from a mathematical model, simulation results, or from descriptions of system components. Analysis of some aspects of the ILSS are possible (e.g., worst-case complexity of algorithms). Next (the center of the figure), an implementation of the system is created; this can be done by hand or automatically generated in a compile step (note that the original Logical Sensor Specifications(Henderson & Shilcrat, 1984) could be compiled into Unix shell script or Function Equation Language (FEL), an applicative language). Either way, the monitoring, embedded testing or taps are incorporated into the system implementation. Finally (the right hand side), validation is achieved by analyzing the system response and performance measures generated during system execution. In this way, there are some semantic constraints on the values monitored which relate the system output measures to the original question posed for the specification.

Currently, an ILSS library is under development as part of an interactive graphical programming environment called "CWave" used to design and execute real-time control systems (refer to

"http://easy.cs.utah.edu/cwave/index.htm" for more information about the CWave project). Currently, we have a theoretical framework and validation strategy with a partial implementation within CWAVE. CWave is a graphical program specification language that has been created to design measurement systems and has been funded by HP. CWave has been applied to broad robot systems (e.g., Lego robot warehouse demos) in our software engineering projects class here at Utah. Finally, CWave is a specification language and can be linked to simulation tools, or executed in an interpreted mode, or compiled for incorporation in embedded systems.

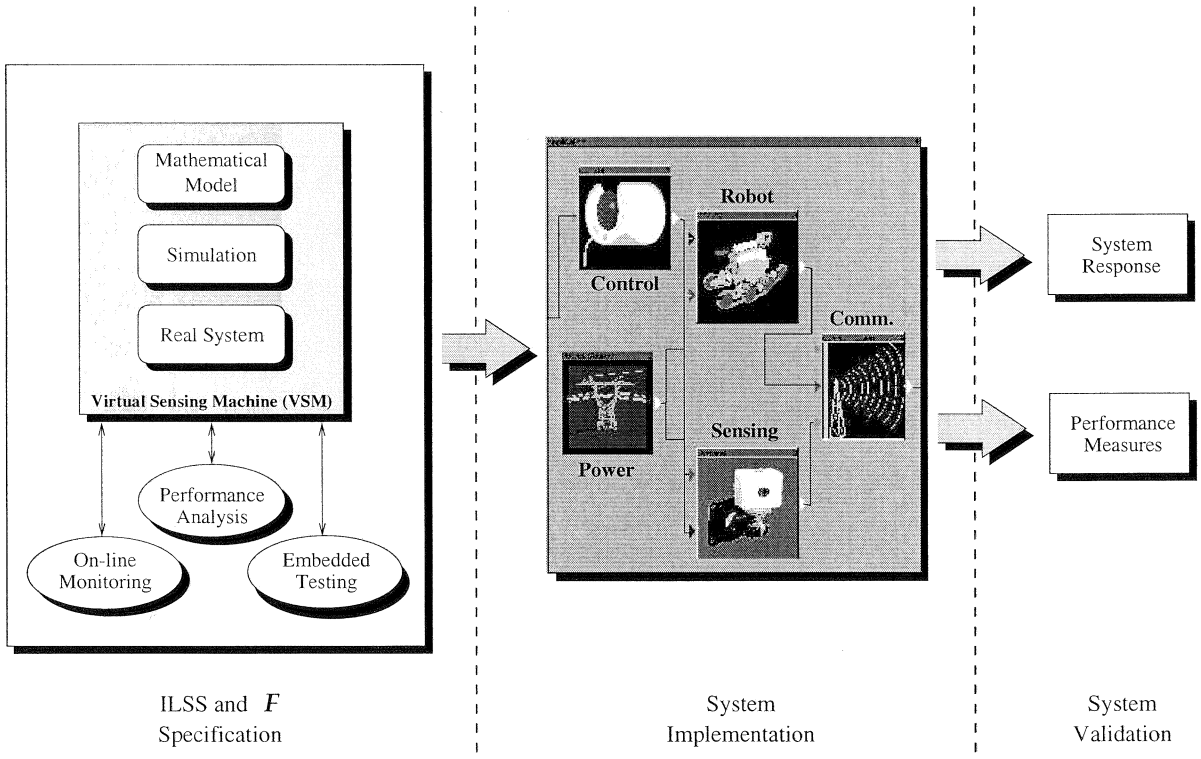

Figure 2: The Instrumented Logical Sensor System Components.

2 Performance Semantics of Sensor Systems

The use of sensors in safety critical applications, such as transportation and medicine, requires a high level of reliability. However, increased robustness and reliability of a multisensor system requires increased cost through redundant components and more sensor readings and computation. In contrast, increasing the efficiency of the system means less redundant components, fewer sensor readings and less computation. Performance analysis is crucial to making an informed tradeoff between design alternatives.

Performance analysis consists of a static analysis of a specification of the system and its parameters as well as a dynamic analysis of the system's run-time behavior. The static analysis can be based on some formal description of the syntax and semantics of the sensor system, while the dynamic analysis requires on-line monitoring of some quantitative measures during run-time.

Our goal is to achieve strong performance analysis and provide information which allows the user to make informed choices concerning system tradeoffs. This involves a sensor system model which permits quantitative measures of time and space complexity, error, robustness, and efficiency, and which facilitates analysis, debugging and on-line monitoring.

Formal semantics of programming languages provides techniques to describe the meaning of a language based on precise mathematical principles. These formal techniques should provide the following: precise machine-independent concepts, unambiguous specification techniques, and a rigorous theory to support reliable reasoning (Gordon, 1979). The main types of formal semantics are: *denotational semantics* which concerns designing denotations for constructs, *operational semantics* which concerns the specification of an abstract machine together with the machine behavior when running the program, and *axiomatic semantics* which concerns axioms and rules of

inference for reasoning about programs.

Our view is that performance semantics should allow us to compute measures of interest on program structures. Denotational semantics is the closest to our view since, according to (Ashcroft, 1982), to specify the semantics of a language denotationally means to specify a group of functions which assigns mathematical objects to the program and to parts of programs (modules) in such a way that the semantics of a module depends only on the semantics of the submodules. Thus, given a set of programs, $\mathcal{P}$, from a language, and an operating context, $\mathcal{C}$, the semantics is a set of functions

$$\mathcal{F} = \{f_i\}$$

where

$$f_i : \mathcal{P} \times \mathcal{C} \rightarrow \Re$$

where $\Re$ is the measurement domain.

The static semantics defines structural measures over the syntax of $p \in \mathcal{P}$. This includes standard measures such as maximum depth of the program graph, branching measures, data structure properties, storage estimates and standard computational complexity measures. Note that these can be determined without reference to $\mathcal{C}$ (i.e., $f : \mathcal{P} \rightarrow \Re$). This can be extended to include functions of the operational context $\mathcal{C}$, including sensor models, accuracy, precision, redundancy and replacement, as well as operating system effects, communication strategies and protocols, and processor properties.

The dynamic semantics include validity measures and operational characteristics. Validity measures permit the comparison of behavior models to actual run-time performance (monitors), while operational characteristics are simply measures of run-time values (taps). The values of a tap or monitor are represented as a sequence $X = (x_n : n \in \mathcal{N})$; x_n is the n^{th} value produced by the tap

or monitor

$$X : \mathcal{N} \rightarrow S$$

where S is the structure produced by the tap or monitor.

The selection of functions in $\mathcal{F}$ depends directly on the user's needs and are defined so as to answer specific questions. Standard questions include actual running times, space requirements, bottlenecks, etc., and a complex application can be investigated in a top down manner – the user may define new measurement functions on lower level modules once information is gained at a higher level. This forces the user to identify crucial parameters and to measure their impact. For example, a computer vision application may be data dependent, say on the number of segmented objects or their distribution in the image. Thus, the user is coerced into a better understanding of the significant value regimes of these parameters and may develop monitors to ensure that the application stays within a given range, or that it dynamically switches algorithms when a particular parameter value occurs (e.g., more than 1000 segmented objects occur in the image). The main point is that the user can construct executable versions of the $f_i \in \mathcal{F}$ to ensure the validity of the controller as it runs.

Although computational complexity provides insight for worst case analysis, and for appropriate population distribution models, average case analysis can be performed, we propose here what might be termed *empirical case analysis* which allows the user to gain insight into the system without requiring a detailed analytical model of the entire application and its context. Very few users exploit formal complexity analysis methods; we believe that empirical case analysis is a very useful tool.

2.1 Simple Example: Time Vs. Robustness Using Sonar Readings

Suppose that we want to determine how many sonar readings to use to get a robust range estimate, but would like to trade off against the time taken to sample. This simple example demonstrates the motivation of the proposed approach and how it can be used to select between alternatives. In this example we have a "classical" tradeoff between speed (time to accomplish a certain task) and robustness (a combination of accuracy and repeatability). Assume that the sonar has been calibrated to eliminate any environmental effects (e.g., wall type, audio noises, etc.). The variables in this case are the accuracy of the physical sonar sensor and the number of readings taken for the same position.

Assuming the time to take one reading is t, the error standard deviation is σ, and the probability of a bad reading is Pr_b, taking one reading yields minimum time and worst accuracy. By adding a filter (e.g., averaging) and taking multiple readings, accuracy increases and time also increases. Therefore, we need quantitative measures to decide how many readings are needed to achieve the required accuracy (measured in terms of the standard deviation of the error) within a time limit.

Using the formalism presented earlier, the semantics of this problem can be defined using the set of functions $\mathcal{F} = \{time, error, repeatability\}$. In the case of using a single reading these functions can be written as:

$$time(single) = t$$

$$error(single) = \frac{\sigma}{\sqrt{(1 - Pr_b)}}$$

$$repeatability(single) = 1 - Pr_b$$

Now, if we take the average of n readings, the semantics can be written as:

$$time(average) = nt + \tau_n$$

$$error(average) = \frac{\sigma}{\sqrt{n * (1 - Pr_b)}}$$

$$repeatability(average) = 1 - Pr_b^n$$

where τ_n is the time to calculate the average of n readings, and $\tau_1 = 0$.

In this simple example we were able to get estimates of the required measures using mathematical models. However, we did not consider the changes in the environment and how it affects these measures. In this case, the set of functions $\mathcal{F}$ are mappings from the cross product of the program $\mathcal{P}$ and the operating context C to the measurement domain $\Re$, that is

$$f_i : \mathcal{P} \times \mathcal{C} \rightarrow \Re$$

To solve this problem, we either have to model the environmental effects and include it in our model, or we may need to conduct simulations if a mathematical model is not possible. Simulation is a very useful tool to approximate reality, however, in some cases even simulation is not enough to capture all the variables in the model, and real experiments with statistical analysis may be required to get more accurate results. Thus, the formal functions can be operationalized as monitors or taps in the actual system.

3 Sensor System Specification

The ILSS approach is based on *Logical Sensor Systems* (LSS) introduced by Henderson and Shilcrat (Henderson & Shilcrat, 1984). LSS is a methodology to specify any sensor in such a way that hides its physical nature. The main goal behind LSS was to develop a coherent and efficient presentation of the information provided by many sensors of different types. This representation provides

a means for recovery from sensor failure and also facilitates reconfiguration of the sensor system when adding or replacing sensors (Henderson *et al.*, 1985).

We define the ILSS as an extension to the LSS and it is comprised of the following components (see Figure 3):

1. *ILS Name*: uniquely identifies a module.
2. *Characteristic Output Vector (COV)*: strongly typed output structure. We have one output vector (COV_{out}) and zero or more input vectors (COV_{in}).
3. *Commands*: input commands to the module ($Commands_{in}$) and output commands to other modules ($Commands_{out}$).
4. *Select Function*: selector which detects the failure of an alternate and switches to another alternate (if possible).
5. *Alternate Subnets*: alternative ways of producing the COV_{out}. It is these implementations of one or more algorithms that carry the main functions of the module.
6. *Control Command Interpreter (CCI)*: interpreter of the commands to the module.
7. *Embedded Tests*: self testing routines which increase robustness and facilitate debugging.
8. *Monitors*: modules that check the validity of the resulting COVs.
9. *Taps*: hooks on the output lines to view different COV values.

These components identify the system behavior and provide mechanisms for on-line monitoring and debugging. In addition, they give handles for measuring the run-time performance of the system.

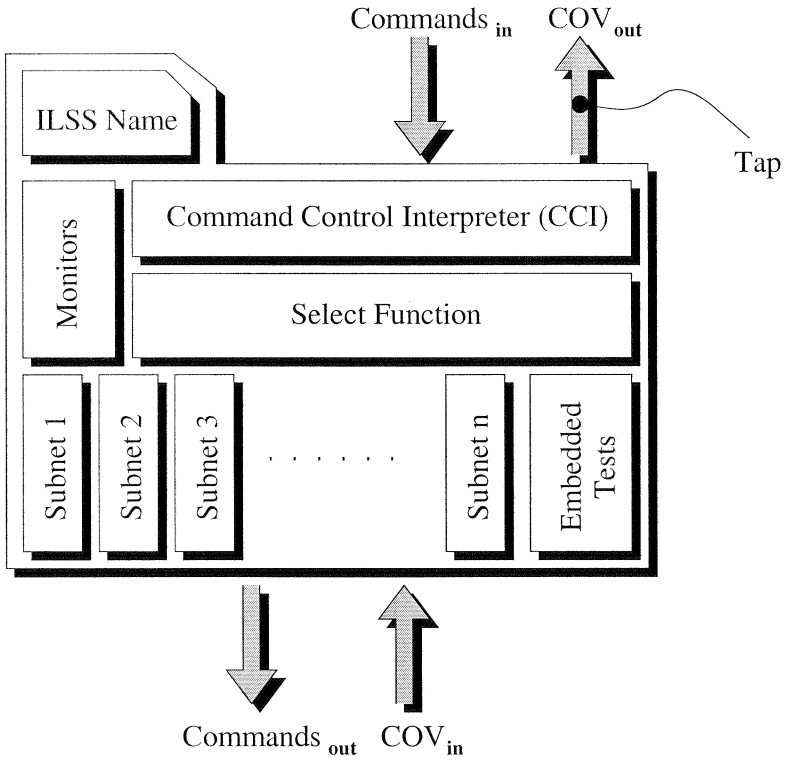

Figure 3: The extended logical sensor module.

Monitors are validity check stations that filter the output and alert the user to any undesired results. Each monitor is equipped with a set of rules (or constraints) that governs the behavior of the COV under different situations.

Embedded testing is used for on-line checking and debugging proposes. Weller proposed a sensor processing model with the ability to detect measurement errors and to recover from these errors (Weller *et al.*, 1990). This method is based on providing each system module with verification tests to verify certain characteristics in the measured data and to verify the internal and output data resulting from the sensor module algorithm. The recovery strategy is based on rules that are local to the different sensor modules. We use a similar approach in our framework called *local embedded*

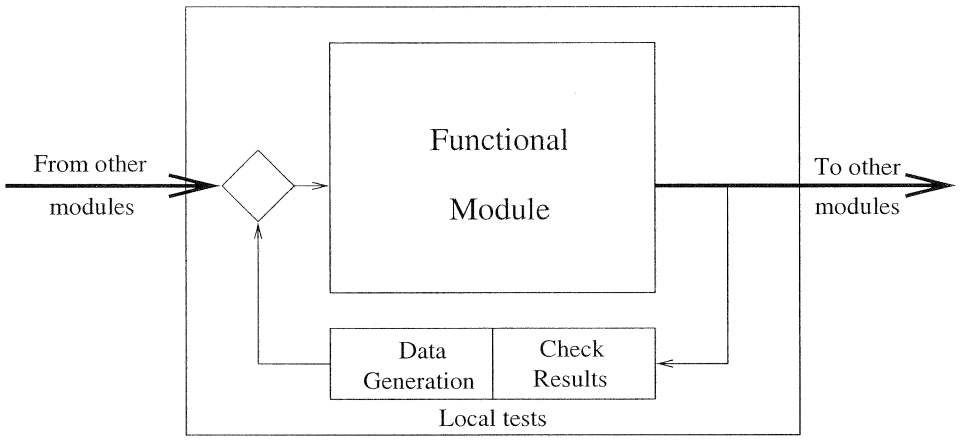

Figure 4: Local embedded testing.

testing in which each module is equipped with a set of tests based on the semantic definition of that module. These tests generate input data to check different aspects of the module, then examine the output of the module using a set of constraints and rules defined by the semantics. Also these tests can take input data from other modules if we want to check the operation for a group of modules.

Figure 4 illustrates the idea of local embedded testing. Local embedded testing increases the robustness of the system and provides the user with possible locations to tap into when there is a problem with the system.

3.1 Construction Operators

In our proposed framework, a sensor system is composed of several ILSS modules connected together in a certain structure. We define operations for composing ILSS modules, and then define the semantics of these operations in terms of the performance parameters. Some of these operations are (see Figure 5):

- $Serial(ILSS1, ILSS2)$: two logical modules are connected in series. Here $COV3 = COV2$.
- $Select(ILSS1, ILSS2)$: $COV3$ is equal to either $COV1$ or $COV2$.
- $Combine(ILSS1, ILSS2)$: $COV3$ is the concatenation of $COV1$ and $COV2$.

For these simple constructs, the semantics is defined as a set of functions that propagate the required performance measures. Several techniques can be used for propagation. Best case analysis, worst case analysis, average, etc. Selecting among these depends on the application, hence it should be user defined. As an example, the time of the resulting logical system using worst case analysis can be calculated as follows:

- $time(Serial(ILSS1, ILSS2)) = time(ILSS1) + time(ILSS2)$
- $time(Select(ILSS1, ILSS2) = max(time(ILSS1), time(ILSS2))$
- $time(Combine(ILSS1, ILSS2) = max(time(ILSS1), time(ILSS2))$

Hence, the semantic functions of the composite system are defined in terms of the semantic functions of the subcomponents, Similarly, functions that define the propagation of other performance measures can be defined in the same way.

For error propagation, we use a simple approach which does not require carrying a lot of information through the system. This approach is based on the uncertainty propagation described in (Faugeras, 1993; Holman & W. J. Gajda, 1978). Assume that we have a certain module with n inputs $X = (x_1, x_2, \ldots, x_n)$ and m outputs $Y = (y_1, y_2, \ldots, y_m)$ such that $Y = f(X)$, and assume that the error variance associated with the input vector is $\Lambda_X = (\Lambda_{x_1}, \Lambda_{x_2}, \ldots, \Lambda_{x_n})$ (see Figure 6),

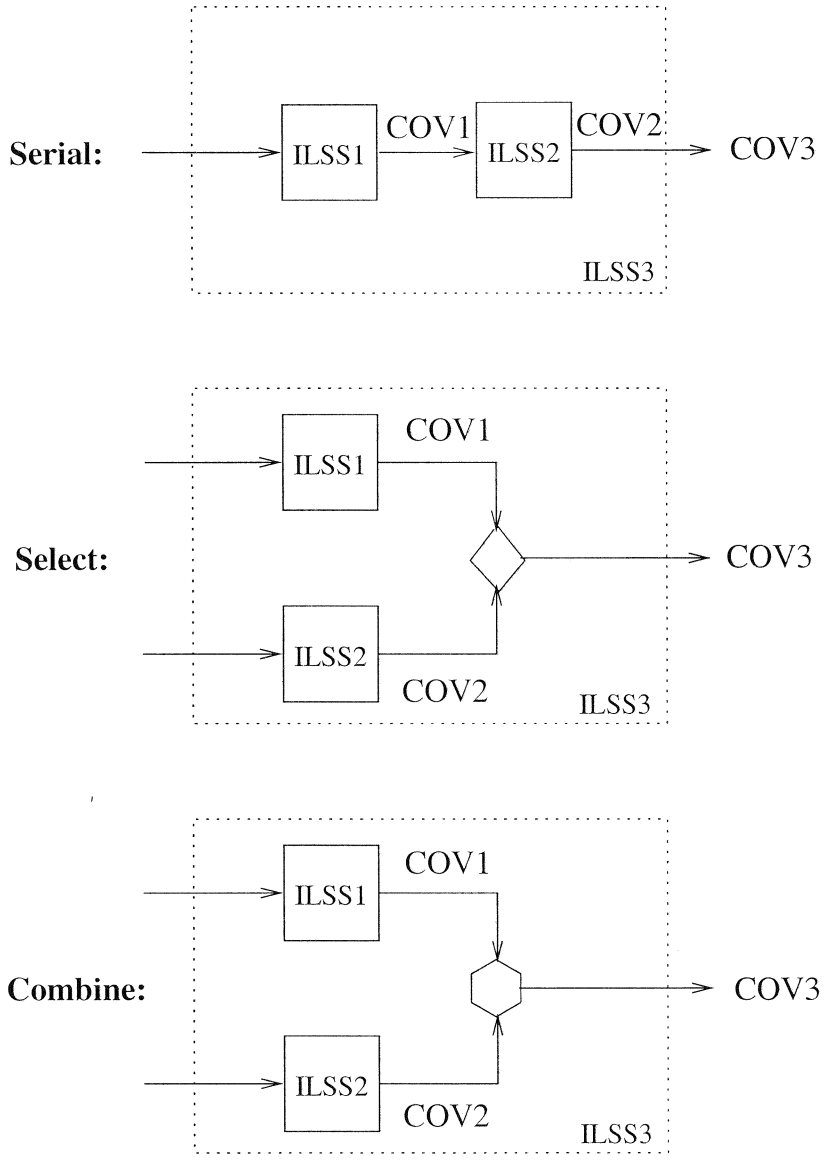

Figure 5: Some operations used for propagating the performance measures.

then the error variance for the output vector is calculated using the equation:

$$\Lambda_Y = \left(\frac{\partial Y}{\partial X}\right) \Lambda_X \left(\frac{\partial Y}{\partial X}\right)^T$$

where $\frac{\partial Y}{\partial X}$ is the partial derivative of Y with respect to X evaluated at the measured value of the input vector X. If all the elements in X are independent variables, then this equation can be written as:

$$\Lambda_{y_i} = \sum_{j=1}^{n} \left(\frac{\partial y_i}{\partial x_j}\right)^2 \Lambda_{xj} \ , i = 1, 2, \ldots, m$$

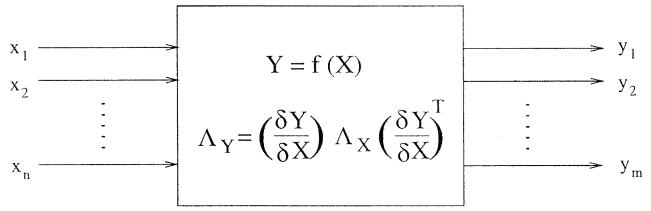

Figure 6: A simple approach for error propagation.

Our overall goal is to provide a tightly coupled mechanism to map high-level performance measures onto an appropriate set of monitors, tests and taps so as to provide the required information.

4 Implementation

The ultimate goal of this project is to utilize the proposed theoretical framework in a usable modeling and prototyping environment with tools for analysis, debugging, and monitoring sensor systems with emphasis on robot control applications. Thus, we are developing an ILSS library within a visual programming system called CWave targeted toward the development of control systems for measurement devices and hardware simulations. CWave is developed by the Component Software