The DELPHI System: A System Level Tool for Integrated Design of Real-time Signal Processors

John S. Baras*, Fariborz Ebrahimi*, Bruce Israel*,
Anthony LaVigna**, and David MacEnany**

Electrical Engineering and Systems Research Center
University of Maryland, College Park MD 20742

1. Introduction

We are developing an expert system to facilitate the CAD of sophisticated and complex chips for real-time non-linear signal processing. The software is called DELPHI which is an acronym for DEsign Laboratory for Processing Hidden Information. The system combines an AI engine, symbolic algebra, and multiprocessor numerical schemes. Sophisticated reasoning, mathematical, and computational tools are provided in a form suitable for immediate use by systems engineers. One of the major advantages of DELPHI is its ability to interact symbolically with the user. The architecture of the DELPHI system is shown in Figure 1. The current architecture can be classified as a *shallow coupled system* since the knowledge-based system has little knowledge of numeric routines. The numeric routines, which are treated as 'black boxes', are managed by the system to solve the given problem and interpret the results. The system can be used as: (a) a tool for integrated system design, (b) a tool for integration of symbolic and numeric computing, and (c) an advanced teaching aid.

The *intelligent interface* block allows the user to enter a signal and observation model symbolically. The system is capable of discovering the nature of the given model such as diffusion, point, or Markov chain type model. The system can provide guidance to the user as needed. The *modeling* block can automatically build a linear dynamical system model for a Gaussian process using the Akaike-Rissanen-Hannan [1] theory. Currently, we are working on building general diffusion type [2], point process type, and hidden Markov chain type models. The architecture is designed in such a manner that provides the capability of calling a variety of statistics and time series libraries for performing statistical validation tests on the model under construction. The *likelihood ratio* block can perform similar computations for point process [3] observations, mixed diffusion point process type models, and Markov chain models. During this year we will couple the system to a silicon compiler for actual VLSI chip layout.

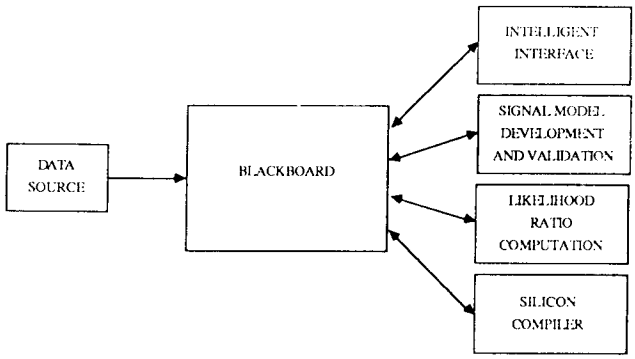

Figure 1. *DELPHI Architecture.*

2. An Example From Sequential Detection

Let us consider the binary sequential hypothesis testing problem. We are given a scalar-valued signal x_t which satisfies the stochastic differential equation

$$\frac{dx_t}{dt} = f(x_t) + g(x_t)\, n_t$$
$$x_0 = \nu$$

where n_t is a white noise signal. Unfortunately, we cannot observe x_t directly, instead we only observe y_t, a scalar-valued stochastic process. Under each hypothesis the observed data is the output of a stochastic differential equation, i.e.,

$$\text{Under } H_1: \quad \frac{dy_t}{dt} = h(x_t) + \bar{n}_t$$
$$\text{Under } H_0: \quad \frac{dy_t}{dt} = \bar{n}_t$$

* The work of this author was supported partially through NSF Grant NSFD CDR-85-00108 and partially through ONR Grant N00014-83-K-0731.

** The work of this author was supported by an ONR Fellowship.

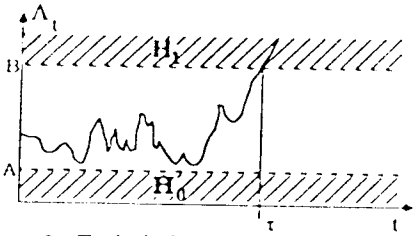

Figure 2. *Typical plot of the likelihood ratio*

where $\bar{n}_t$ is another white noise signal which is independent of n_t. Notice that if $f(\cdot)$ and $h(\cdot)$ are linear and $g(\cdot)$ is constant, then this becomes the standard problem solved by the Kalman filter.

Data is observed continuously starting at an initial time which is taken for convenience to be zero. At each time $t > 0$, the decision-maker can either stop and declare one of the hypotheses to be true or can continue collecting data. The decision-maker selects his decision based on the data collected up to time t, so as to minimize an appropriate cost function. To summarize, the decision maker makes two decisions, when to stop (represented by τ) and once stopped, which hypothesis is true (represented by δ).

More precisely, a decision policy involves the selection of a termination time τ, and of a binary valued decision δ. If $\delta = 1$, we shall accept hypothesis H_1; if $\delta = 0$ we shall accept hypothesis H_0. An *admissible decision policy* is any pair $u = (\tau, \delta)$ of RV's where τ and δ depend only on past observations. The collection of all admissible decision policies will be denoted by $\mathcal{U}$.

An admissible policy $u = (\tau, \delta)$ is a *threshold policy* or *of threshold type* if there exists constants A and B, with $0 < A \leq 1 \leq B < \infty$ and $A \neq B$, such that

$$\tau = \inf(t \geq 0 \mid \Lambda_t \notin (A, B))$$

$$\delta = \begin{cases} 1, & \Lambda_\tau \geq B \\ 0, & \Lambda_\tau \leq A \end{cases}$$

Here Λ_t is the likelihood ratio associated with this problem, namely

$$\Lambda_t = \exp\Big(\int_0^t \hat{h}_s \, dy_s - \frac{1}{2}\int_0^t \hat{h}_s^2 \, ds\Big),$$

and

$$\hat{h}_t = E_1(h(x) \mid \mathcal{Y}_t^y).$$

Pictorially, this means as Λ_t is computed for each time t it is compared to the interval (A, B). The process is stopped the first time Λ_t is larger than B or smaller than A (See Figure 2).

In [2] we showed that under either the fixed probability of error formulation or under Bayesian formulation. The optional policy is a threshold policy. The theory of [2] has been incorporated in DELPHI. Let us consider the Bayes case only.

We shall assume, two costs are incurred. The first cost is for observation and is accrued according to $k\int_0^t \hat{h}_s^2 ds$, where $k > 0$ and $\{\hat{h}_t,\ t \geq 0\}$ is defined above. The second cost is associated with the final decision δ and is given by

$$C(H, \delta) = \begin{cases} c_1, & \text{when } H = 1 \text{ and } \delta = 0; \\ c_2, & \text{when } H = 0 \text{ and } \delta = 1; \\ 0, & \text{otherwise,} \end{cases}$$

where $c_1 > 0$ and $c_2 > 0$.

We are interested in minimizing the average cost. If $u = (\tau, \delta)$ is any admissible policy, then the corresponding average expected cost is

$$J(u) = E\Big(k\int_0^\tau \hat{h}_s^2 ds + C(H, \delta)\Big).$$

The optimal policy is characterized by the thresholds $0 < A^* \leq 1 \leq B^* < \infty$ with $A^* \neq B^*$, are given by the relations

$$A^* = \begin{pmatrix} 1-\varphi \\ \varphi \end{pmatrix}\begin{pmatrix} a^* \\ 1-a^* \end{pmatrix}, \qquad B^* = \begin{pmatrix} 1-\varphi \\ \varphi \end{pmatrix}\begin{pmatrix} b^* \\ 1-b^* \end{pmatrix},$$

where a^* and b^* are the unique solutions of the transcendental equations

$$c_2 + c_1 = k(\Psi'(a^*) - \Psi'(b^*))$$

$$c_2(1-b^*) = c_1 a^* + (b^* - a^*)(c_1 - k\Psi'(a^*)) + k(\Psi(b^*) - \Psi(a^*)),$$

with

$$\Psi(x) = (1-2x)\log\frac{x}{1-x},$$

satisfying $0 < a^* < b^* < 1$.

From the theory of nonlinear filtering, it can be shown that the computation of $\hat{h}_t$ can be accomplished by solving a *linear* stochastic partial differential equation, known as the *Zakai equation*. A numerical approximation technique is used to solve this equation. This technique has been incorporated in the *likelihood ratio* block which currently has the following capabilities: (an M indicates a *Macsyma* computation, an F indicates a *Fortran* computation).

- M1: Input f, g, h in symbolic form. Generate the multi-dimensional Zakai equation.
- M2: Compute automatically discretizations of all stochastic differential equations. Automatically generate *Fortran* code.

F3: Solve numerically pathwise the resulting stochastic difference equations and store the y paths.

M4: Generate discretization schemes for the Zakai equation and automatically generate the associated *Fortran* code.

F5: Automatically integrate numerically the discretized Zakai equation.

F6: Display graphically the solution.

M7: Compute symbollically discretizations of the likelihood ratio and automatically generate appropriate FORTRAN simulation code.

F8: Evaluate numerically the likelihood ratio.

The following additions to the *likelihood ratio* block are planned: apriori bounds for performance of detectors and estimators, additional numerical schemes for the Zakai equation (eq., multi-grid algorithms), and automatic performance evaluation of sequential detectors.

Below is a sample run of the system. The problem is to determine whether the received signal is a lognormal signal plus white noise, H_1, or white noise alone, H_0. This problem can be represented mathematically as:

$$H_1: \quad \frac{dx}{dt} = -x + n_t$$
$$\frac{dy}{dt} = exp(x) + \bar{n}_t$$

$$H_0: \quad \frac{dy}{dt} = \bar{n}_t$$

In figure 3a, we have graphed the likelihood ratio when the input is lognormal. The likelihood ratio eventually is greater than B and the correct signal is detected. Likewise, in figure 3b, we have graphed the likelihood ratio when the input is only white noise. This time the likelihood ratio falls below A and white noise is detected.

3. Hardware and Software Environment

The DELPHI system has been implemented on a TI^{TM} Explorer LX System. It consists of a powerful AI development environment based on the TI^{TM} Lisp machine and the LX system which is a MC68020-based coprocessor board running $UNIX^{TM}$ System V. TI^{TM} Explorer system is the system of choice for this application since it integrates the symbolic computing capabilities of the AI engine with the number

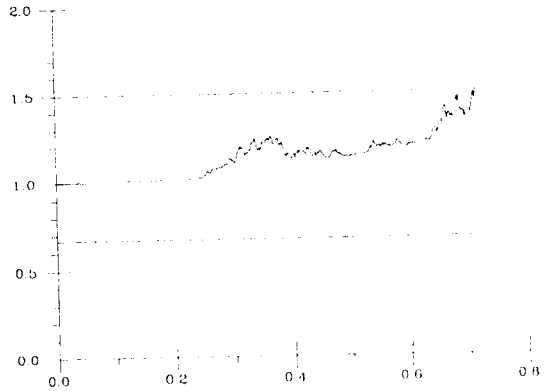

Figure 3a. *Likelihood ratio when input is lognormal*

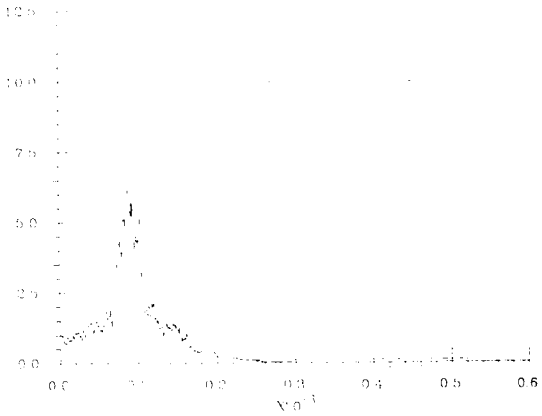

Figure 3b. *Likelihood ratio when input is white noise*

crunching capabilities of the LX system. The two processors are connected to the local NUBUS and communicate through remote procedure calls. The lisp processor has access to 8 MB memory as well as 2 MB of shared memory available on the LX system.

DELPHI is an object-oriented system written entirely in Lisp language with object-oriented facilities of Flavors. DELPHI accesses the numeric routines, written in either Fortran or C languages on the LX system, through remote procedure calls. DELPHI is composed of two types of objects, *data stream* objects and *function* objects. Data streams are sequences of numbers, representing data, along with associated attributes of the data object. Functions can process the data streams to either generate new data streams, to add new attributes to the data streams being processed, or a combination

of both. For example, a function to multiply a data stream by a constant is one that would generate a new data stream, while a function such as RANGE might only add MAX and MIN attributes to an existing data stream.

Data streams and functions, are instances of flavors that can be created by 'make-instance' facility. These flavors have various 'slots' that contain the information about that object and how to control its usage. Three slots are common to both function and data stream objects. The *name* slot specifies the name of the object; the *doc* slot contains documentation that describes the object which appears on the *who-line* when the mouse is placed over the object; The *menus* slot contains a list of the menu paths that the object is accessible through. This option allows for an object to be accessible through multiple menu paths. Each path is a sequence of atoms, where an atom is the name of the menu that would be chosen to go down that branch.

In addition to the above slots, a data stream has three other important slots. The *info* slot contains the actual data in the data stream which is a list of numbers. The *indirect* slot is used for situations where data is not obtained interactively through the *interactor window*. The data might be stored in a file on an alien host system on a network, or may even need to be received from a data acquisition device. In this situation, this slot would contain a lisp function that knows how to go out and get the data that the data stream represents. In this case, the *info* slot would be used to hold special information for that function (file name, host name, etc.) which *indirect* would be invoked with. The *qualities* slot represents the information known about this data stream. It represents a 'frame' stored as a list of pairs, each pair being an attribute and a value. For example, the information can represent the constraints on the data stream in order to determine if the given data stream is a valid input parameter for a specific function.

In order to process data streams effectively, a function object requires specifications for the inputs, the body of the function which specifies what will be done with inputs, and outputs which specifies what to do with the results. These characteristics of functions are specified by ***inputs***, ***body***, and ***outputs*** slots respectively. The ***inputs*** slot specifies required input parameters. This can be specified either as a list of parameter definitions or as an input body. If it is a list of the parameters, DELPHI will handle all interactions and choices necessary for the user to make. If more complex facilities are necessary, then this slot may contain an arbitrary lisp function that handles all inputs provided by the user. A macro, ***input-lambda***, is available for defining such functions. The ***body*** slot contains a lisp function which has as many arguments as were returned from the *inputs* definition. This function will be called with the arguments from *inputs*, and can return multiple values, each representing a different inference or computation. This function does not create or affect data stream structures in any way, it just performs calculations. The *outputs* slot specifies what to do with the results returned by the body. The macro *output-lambda* is used to define this, with two variable lists to be bound to the inputs and outputs respectively, and then the body of the *output-lambda* will be executed to determine what to do with the lists.

The DELPHI *user interface* module is composed of four windows, a *command menu*, a textual *display window*, a *graph window*, and a lisp *interactor window*. The *command menu* is used for examining data streams and functions, executing the various functions, creating new data streams, and changing various options controlling the behavior of the system. Data can be examined in either textual format, graphically, or both. If a data stream is displayed textually, all of its attributes will be displayed in the text *display window*, along with all the numeric values of the data. The user has the option of viewing the data graphically in the *graph window*. Another option allows the user to compare multiple data streams, by graphing each of the data streams at the same time in the *graph window*. If *Run Function* option is selected, a menu containing all available functions will pop up which may be executed. This menu is hierarchical, so a function may appear either directly or within sub-menus of this menu. Once the desired function is selected by the user, a new window will pop up which displays the required parameters needed for execution, along with the respective default values. The user has the choice of accepting the default values or changing them. Another available facility from the *command menu* is the *change options* button, which allows various options that customize the behaviour of DELPHI system to be set or changed.

4. Future Directions

We are planning to use a blackboard architecture [4] based on a *deep coupling* approach which utilizes extensive

knowledge of each numeric routine. Each routine's inputs and outputs, purpose, limitations, constraints, and side-effects may be explicitly represented which can be used by the knowledge-based system during the problem-solving phase. This approach has the advantage of providing much better maintainability and extendability of DELPHI. However, the cost is the overhead involved in its initial development phase.

5. References

[1] E. J. Hannon, W. T. Dunsmuir and M. Deistler, *"Estimation of Vector ARMAX Models"*, J. Multivanche Anal., 1981.

[2] J. S. Baras and A. LaVigna, *"Architectures for Real-Time Sequential Detection"*, this proceedings.

[3] D. C. MacEnany and J. S. Baras, *"Bayesian Sequential Detection for Point Processes"*, to appear in Proc. of 26th IEEE Conference on Decision and Control, Dec. 1987.

[4] H. Penny VII, *"Blackboard Systems: The Blackboard Model of Problem Solving and the Evaluation of Blackboard Architectures"*, The AI Magazine, Summer 1985, pp. 36-53.