Analysis of data flow models using the SARA graph model of behavior*

by W. RUGGIERO**, G. ESTRIN, R. FENCHEL, R. RAZOUK, D. SCHWABE and M. VERNON

University of California at Los Angeles
Los Angeles, California

A number of investigators have continued to discuss application of asynchronous techniques to improve the computational power of computing systems.2,15,6,9 In fact the need for asynchronous design techniques arose in the earliest machines26,6,19 which introduced parallel handling of bits in a number and overlapping of independent operations. The concept of distributed autonomous concurrent processors was essential to the visionary architecture proposed by Holland16 to support "operating programs floating in a sea of hardware." The importance of such concurrent and asynchronous systems has increased recently because of the availability of entire processing units on one or a few chips and the potential cost reduction of those units. Moreover, it is expected that work on VLSI technology will permit even more complex systems to be reduced to silicon if their design and market analysis have been validated sufficiently to justify the cost. The need for validation prior to costly physical implementation has increased the value of methods and tools which support modeling and analysis.

SARA (System ARchitects Apprentice) is such a supported methodology particularly applicable to multilevel modeling of concurrent systems. Dennis has proposed computation structures for concurrent system design which have aroused considerable interest. This work shows how SARA tools may be used to analyze data flow models utilizing Dennis' computation structure. This work refers to the latter as the Dennis Data Flow (DDF) Model.

In the first part of this paper we establish relationships between primitives of two models—the Dennis Data Flow Model and the SARA Graph Model of Behavior (GMB). The second part of the paper presents an example showing how SARA tools can be used to construct and simulate a data flow model. The discussion and the example should help to increase the reader's understanding of both models.

This opens the way for application of supported multilevel design methodologies, like that supported by SARA, to higher-level design of systems incorporating devices which implement data flow primitives.

RELATIONSHIP BETWEEN DDF AND GMB

Token flow models

The Graph Model of Behavior (GMB)11 and the Dennis Data Flow (DDF) Model8 can be considered token flow models which were invented to help synthesize asynchronous concurrent systems. A token flow model uses a directed graph composed of nodes and arcs to describe the static component of a system behavior. Tokens, which are initially placed on the arcs, flow through the graph, activating and deactivating nodes. In this way, selected dynamic behavior of a system can be modeled and observed. Such models are most suitable for describing the behavior of concurrent and asynchronous systems in which some events may occur concurrently, but those occurrences must be controlled to satisfy constraints, i.e. precedence and frequency.

The GMB was developed in a search for a natural, simple and powerful method for describing and analyzing the flow of data and control in systems. The GMB was derived from the bigraph model of computation (GMC)13 which formalized earlier work at UCLA.7,18 The GMB was influenced by the LOGOS work at Case Western Reserve.14

A simple data flow model was studied in the basic work by Karp and Miller.17 Dennis and Rodriguez23 developed program graphs which were later revised and analyzed as a form of parallel program schema. A chain of investigators in Dennis' information structure group has continued exploration of fundamental data flow primitives and architectures. In this paper we concentrate on the data flow model developed by Dennis. Other noteworthy data flow models have been reported in the work of Adams,1 Arvind - Gostelow,15 Bahrs2 and Kosinski.19

Dennis' data flow primitives5

A data flow language is a machine language for expressing computations in which an instruction executes when and only when all operands needed for that instruction become available. The instructions, at whatever level they might exist, are purely functions and produce no side effects. Dennis' data flow language seeks to define a scheme of repre-
sentation that exposes concurrency while maintaining a guarantee of determinacy. This language considers only programs that compute a set of output values from a given set of input values and that define the functional dependence of output values on input values.

An elementary data flow program can be represented as a bipartite directed graph where the two types of nodes are called links and actors. The arcs of a data flow program should be regarded as channels through which tokens flow carrying data values. A data flow token is a primitive concept. It is better described as a pair \((e, v)\); \(e\) is an enable flag and \(v\) is a data value. The enable flag signals the presence of the token which carries a data value to be used in the computation.

A DDF link cannot have more than one incident arc, whereas it may have more than one emanating arc. A distinguished input link is one that has no incident arcs; a distinguished output link has no emanating arcs. There are two kinds of link nodes—data link and control link. Figure 1 shows the kinds of actor nodes from which elementary data flow programs are constructed. The \(T\)-gate, \(F\)-gate and Merge actors are called control actors. The Or, And and Not actors are called boolean actors.

The execution of a data flow program is described by a sequence of snapshots; each snapshot shows the data flow program with tokens and associated values placed on some arcs. In the case of control arcs, the associated values are of the type \(\text{truth} = \{\text{true}, \text{false}\}\); for data arcs, the values are of the types \(\text{integer}, \text{real}, \text{or string}\). Execution of a data flow program advances from one snapshot to the next through the firing of some randomly selected link or actor that is enabled in the earlier snapshot. Except for the Merge actor, a node is enabled when all input arcs have a token and there is no token on any output arc. The Merge actor is enabled if there is no token on the output arc and either it has a token with value true on the input control arc and a token on the true arc input or it has a token with value false on the input control arc and a token on the false arc input. When a node fires it removes all enabling tokens from its input arcs and places tokens on its output arcs. An exception is made for the \(T\)-gate and \(F\)-gate. If the token value on the input control arc of a \(T\)-gate is False, the token on the input data arc is removed but no token will be placed on the output data arc. The same happens with an \(F\)-gate and a True value on the input control arc.

The value associated with an output token is a function of the values of the enabling input tokens. Firing an Operator applies the function denoted by the symbol written in the Operator to the set of values associated with the tokens on the input arcs and associates the resulting value with the

![Figure 1—DDF nodes.](From the collection of the Computer History Museum (www.computerhistory.org))

![Figure 2—Firing rules for link nodes.](From the collection of the Computer History Museum (www.computerhistory.org))

![Figure 3—Firing rules for operators and deciders.](From the collection of the Computer History Museum (www.computerhistory.org))
token placed on the output arc. Firing a Decider has a similar effect, but the symbol in the Decider denotes a predicate and a control value is associated with the output token. Figures 2, 3 and 4 show the effects of firing.

Relating the two models

In the most general terms, information processing systems can be considered as flow and transformation systems. Basically, there are two commodities which flow during a processing activity in those systems—one is the control over resources and data and the other is the data itself.

The Graph Model of Behavior (GMB) is a fundamental model incorporated in the SARA (System ARchitects Apprentice) methodology for multilevel design of concurrent systems. The GMB tools provide languages for modeling the behavior of a digital system in three related domains—control, data and interpretation.

The control domain is concerned only with the control flow aspects of the system. It utilizes a directed graph (called the control-graph) where nodes model steps in the computation and arcs model precedence. Tokens flowing through the control graph establish activation conditions for control nodes. A node is activated if a simple function (a combination of +, * and a weight) holds for the tokens on the incoming arcs. Activation amounts to removing the activating tokens and, upon completion of the activated process, adding tokens to the outgoing arcs according to a similar simple function.

The data domain describes flow of data and access capability constraints. A directed bipartite graph (called the data graph) is used to show how input data streams can flow through the system and where they are transformed (processed) in order to generate output data streams. The data graph describes the organization of data places (called data sets) and computation points (called processors). Data arcs describe allowable directed data paths between processors and data-sets.

The third domain, the interpretation, defines the format of data-sets, the format of data which flows along the data-arcs and the specific procedures to be carried out by activated processors. The current implementation of the GMB simulator uses an interpretation language which is an extension of PL/1 (called PLIP), but different interpretation languages could be introduced to enhance flexibility.

The three domains are associated as follows: Each control node is associated with at most one controlled processor in the data graph. Each controlled processor is associated with a unique (non-empty) set of control nodes in the control graph. The activation of a control node implies the non-preemptable activation of the associated controlled processor (if any). All concurrent activations of control nodes that are associated with the same controlled processor imply a sequence of activations of that controlled processor. The order of these activations is random. Segments of interpretation define the data structure associated with each data arc and the specific computation that is executed by each controlled or uncontrolled processor.

The GMB operates as follows: The "token machine" activates enabled nodes in the control graph. Control nodes activate controlled processors in the data graph while changes in the distinguished inputs activate uncontrolled processors in the data graph. The controlled and uncontrolled processors cause execution of code segments in the interpretation domain. The code segments cause data transfers (possibly with transformations) between data sets. Controlled processors may feed information back to the control domain to determine choice of output branching of the associated control nodes. The GMB can be used in a flexible way for the following purposes: a designer may choose to explicate more or less control flow; a designer may choose to vary the amount of data flow abstraction; a designer may choose to utilize different interpretation languages offering different powers of abstraction. Tables 1 and 2 tersely summarize properties of the current set of structural and behavioral modeling primitives. Table 3 summarizes properties of proposed extended primitives. The SL1, GMB and PLIP languages (which support multi-level modeling) and a simulator (which supports analysis) are implemented at both UCLA and MIT-MULTICS.

In order to illustrate and make more meaningful comparison between the two models, let us first express the data flow primitives in terms of GMB primitives. Subsequently, some GMB constructs are expressed in terms of data flow primitives.
TABLE I.—Modeling Primitives

<table>
<thead>
<tr>
<th>Structural Primitives</th>
<th>Type</th>
<th>Graphical</th>
<th>Machine Processable</th>
</tr>
</thead>
<tbody>
<tr>
<td>A NAMED MODULE REPRESENTS AN OBJECT WHOSE INTERNAL, FULLY-NESTED STRUCTURE IS RIDDEN FROM THE OUTSIDE. A MODULE'S ONLY POSSIBLE COMMUNICATION WITH THE OUTSIDE IS THROUGH A SOCKET. OTHERWISE, A MODULE'S NAME IS KNOWN ONLY TO THE STRUCTURE WITHIN WHICH IT IS NESTED. EXAMPLE: THE MODULE, CALLED &quot;NAME_C&quot;, IS COMPOSED OF MODULES &quot;A&quot; AND &quot;B&quot;.</td>
<td></td>
<td></td>
<td>NAME_C (A,B,C)</td>
</tr>
<tr>
<td>A NAMED SOCKET IS PART OF A MODULE BUT THE SOCKET'S NAME IS KNOWN BOTH INSIDE AND OUTSIDE OF ITS HOST MODULE. EXAMPLE: THE MODULE &quot;NAME C&quot; IS COMPOSED OF MODULES A WITH SOCKET SA, B WITH SOCKET SB AND C WITH SOCKET SC.</td>
<td></td>
<td></td>
<td>NAME_C (A&lt;@SA&gt;, B&lt;@SB&gt;, C&lt;@SC&gt;)</td>
</tr>
<tr>
<td>A NAMED INTERCONNECT IS A STRUCTURE WHICH CONNECTS TWO OR MORE SOCKETS. EXAMPLE: AN INTERCONNECTION CALLED &quot;LINK&quot; CONNECTS THE SOCKETS A&lt;@SA&gt;, B&lt;@SB&gt;, AND C&lt;@SC&gt;.</td>
<td></td>
<td></td>
<td>NAME_C (A&lt;@SA&gt;, B&lt;@SB&gt;, C&lt;@SC&gt;) LINK : A@SA-B@SB-C@SC)</td>
</tr>
</tbody>
</table>

Expressing data flow primitives in terms of GMB primitives

In general, any DDF can be abstracted into an uncontrolled processor where the interpretation of this processor...
### TABLE II. Behavioral Modeling Primitives

<table>
<thead>
<tr>
<th>Behavioral Primitives</th>
<th>Type</th>
<th>Graphical</th>
<th>Machine Processable</th>
</tr>
</thead>
<tbody>
<tr>
<td>A named control node represents a step in a process being modeled. A controlled data processor (see below) may be associated with a node to provide interpretation of the process. Example: A node N1 has a single entry arc S and a single exit arc X.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A named directed control arc represents non-volatile precedence relations between sets of nodes. If there is more than one source or destination node the arc is called coupled; otherwise it is called simple. An enabling token is placed on an arc either as a starting state or upon termination of any of its source nodes. When a node is initiated, its enabling tokens are removed. Example: A and X are simple control arcs. A is a complex control arc whose source set is nodes N1, N2, and whose destination set is nodes N2, N3, N4, and N5. It is an incoming coupled arc whose destination nodes are N1 to N3 in the order specified. If an initial token is placed on the arc, the token machine mechanism would non-deterministically enable N1 or N2 or N3 and the token would be absorbed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A logical relation among the input arcs to a node specifies the precedence conditions that must be satisfied by token states for the node to be initiated. Tokens from the initiating arcs are managed to satisfy the input relations are assigned by the token machine. Tokens are absorbed from one of an initiating arc set governed by an OR relation in a manner established in the token machine and from all members of an initiating arc set governed by AND relation. Example: If enabling tokens exist on either A1 or A2 and on either A3 or A4 then N1 can be initiated.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A logical relation among the output arcs specifies which arcs have tokens placed upon them when a control node is terminated. When an exclusive or output relation holds, a data processor interpretation must decide which arc receives a token. When an AND relation holds all output arcs receive tokens. Example: When N1 terminates, its associated controlled data processor will have decided whether tokens are to be placed on B1 and B2 or on B3 and B4.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A named controlled data processor represents a data transformation object which is activated when a designated control node is initiated. E.g., processor P1 is initiated whenever either N1 or N2 is initiated. When processor P1 terminates it causes tokens to be placed on output arcs of the control node which initiated it. An interpretation of the data transformation and other parameters such as time delay or resource requirements can be associated with the data processor. Example: Processor P1 has a random delay associated with it. Tokens are placed on the arcs in a manner determined by the control node which initiated it.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A named uncontrolled data processor represents a data transformer which processes, at its output, stated functions of its inputs independent of control node states. In the data graph an uncontrolled processor is identified by providing an explicit declaration - an interpretation of the data transformations and other parameters may be associated with it in an identical manner to the controlled processor. Example: The data set D1 is a six-decimal-digit complex floating point number.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A named data arc is statically named data arc, a data arc graph is a directed graph which contains a data arc set. A data processor may send or write access to a data set if the arrow points to or from the data processor respectively. Example: Processor P1 is initiated by control node N1. It reads data from datasets D2 and D3 and writes their sum into dataset D1.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**From the collection of the Computer History Museum (www.computerhistory.org)**
TABLE III.—Extended GMB Primitives

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>GRAPHICAL</th>
</tr>
</thead>
</table>
| A data link is a directed path between structured processors. A structured processor WRITES to a data link only immediately before its termination and READS from a data link only immediately after its activation. A data link may be used to build a connection between two structured processors by connecting a data link followed by a dataset followed by a data link. We refer to the composite as also being a data link between processors. | ![Data Link Between Processors](image1)  
P1: WRITE D;  
END P1;  
P2: READ D1;  
END P2 |
| A message link is a directed path between two processors that provides a fully interlocked mechanism to exchange messages. The processors are able to synchronize control and to exchange data. A message link is said to be active whenever the two processors are synchronized and ready to exchange data. A message link can be viewed as an input/output part for the involved processors. Only processors that can be active at the same time can be connected through message links.  
**single instantiation:** involves only one message link, two processors  
**complex instantiation:** involves several message links at same time; requires arbitration. See [RUGG78] for more details of this model. | ![Single Message Between Processors](image2)  
S: D:=a;  
F:=TRUE;  
while F do  
endwhile;  
b:=D;  
F:=FALSE;  
R: while F do  
endwhile; |

Figure 7—GMB representation of data and control links  
Figure 8—Explicit acknowledge tokens.

is the data flow program. The behavior described by this data flow interpretation can be immediately translated into the GMB Control Graph (CG) and Data Graph (DG).

One approach is to define extended primitives to express data flow behavior only in the DG using for example, un-
controlled processors and data links (see definitions in Tables 2, 3) as shown in Figure 5. The GMB data links have internal storage, are written into just before termination of a processor and are read from only immediately after initiation. Another approach is to describe the data flow behavior by separating the control and the data parts into a GMB CG and a GMB DG respectively. For the rest of this section we assume the latter approach because it gives more insight into the properties of the two models. For practical purposes, a data graph-oriented description of a DDF is usually preferable, (and is used in the demonstration example of Part B); however, the control-oriented description is more suitable for analysis purposes. In a data flow program the flow of control and the flow of data are shown together in the same graph.

As shown in Figure 6, the DDF enabling flag, \( e_r \), is, in GMB terms, an input token for a node \( n \) in the control graph. It signals the presence of a data value \( v \) in a data-set \( d \). A controlled processor \( p_n \) associated with a control node \( n \) then has read access to the data-set \( d \).

Let us consider, in a data flow program a link node \( l \) with one input arc and \( m \) output arcs (Figure 7). The equivalent
Figure 12—DDF representation of GMB fork construct.

Figure 13—DDF representation of GMB join construct.

Figure 14—DDF representation of GMB switch construct.

Figure 15—DDF representation of GMB union construct.
representation in GMB will have

- A data set \( d \) with one input data arc and \( m \) distinct output data arcs.
- A control node \( n \) with one input control arc and \( m \) output control arcs related by an \( "\ast" \) operator in the output logic expression of \( n \).
- If \( m \) is equal to "1," node \( n \) does not need to exist. It is reduced to just a control arc.
- The control node \( n \) does not have any controlled processor associated with it in the data graph.

Except for control actors, all elementary DDF primitives can be represented in GMB using only one control node and one controlled processor. The DDF control actors represent more complex computations.

The DDF interpreter requires an elaborate operation to decide which node to activate next. It needs some kind of acknowledge token which flows internally in the interpreter. In order to effect the same behavior in GMB, the acknowledge tokens may be shown explicitly in the control graph (see Figure 8), may be included in every GMB processor interpretation or may be placed as a burden on the token machine. The last approach would make the token machine equivalent to the DDF interpreter.

The use of explicit acknowledge tokens in a GMB makes the description much less readable. In the remainder of this section, we neglect the acknowledge tokens required in the GMB equivalent expression of the DDF primitives so as to focus attention on the relationship between the two models. Figure 9 shows how actors, except for control actors, are represented in GMB. Figures 10 and 11 show the representation of Gate and Merge control actors respectively.

The GMB representation of a Gate actor shows clearly (Figure 10) that this primitive has a control flow description which is not properly terminating. In GMB terms, the Gate actor is not considered "well behaved" and its presence may complicate the verification of a DDF model.

Finally, we should observe that there is no need to distinguish between data for different data types (boolean and numeric) in a GMB model.

Expressing some GMB constructs in terms of DDF primitives

We do not attempt the expression of non-deterministic GMB constructs in terms of DDF primitives because the set
of programs intended to be expressed by the DDF language includes only deterministic programs. But, we should point out here, that the lack of non-determinism is one of the main weaknesses of DDF language when applied in a real-time programming environment. Some important problems in real-time processing such as mutual exclusion, synchronization and resource management, are in essence non-deterministic problems. The complex control arc is the general form of expressing non-determinism in a GMB program.

Now let us see the equivalent expression, using DDF primitives, of some deterministic GMB constructs. Figures 12-18 show respectively the Fork, Join, Switch, Union, If_Then_Else, Parbegin_Parend and Do_While constructs in terms of DDF primitives.

The equivalent expression of the Union construct (Figure 15) (which is another form of non-determinism in GMB) presented a problem when described in DDF terms—namely the control input of the Merge actor was not defined. This does not seem to be a problem in deterministic programs, because this construct should be used as shown in the If_Then_Else and Do_While constructs (Figures 16 and 18).

Looking at Figure 18, where a sequential construct is presented, we note flexibility of the GMB program in including all the sequential processing inside of a controlled processor. To describe this purely sequential construct in DDF we need the full DDF interpreter mechanism, which was designed to handle concurrent operations. The capability to embody segments of sequential code in elementary primitives, without involving the elaborated mechanisms of the token machine, represents a great advantage for GMB programs. The programmer may specify the use of an extra processor only when the problem really requires it, i.e., when there are some concurrent operations to be performed.
DEMONSTRATION EXAMPLE

A deterministic program in GMB and DDF

In this section we present the solution of a simple computation expressed in GMB and in DDF languages. The problem to be used as an example is the calculation of a root of a function by Newton-Raphson approximation.

The following is a sample of a session with the SARA system. The SARA system is a set of design and modeling tools which are implemented on the MIT-Multics computer system. This demonstration is intended to acquaint all interested persons with the state of implementation of the SARA system. All the demonstrated tools are available to any user with access to the MIT-Multics system.

The example shown here is a GMB model simulating an input and output session for the calculation of a root of a function by Newton-Raphson approximation.

In the session, all user input is preceded by the SARA system prompt "$"; all other information is output by the SARA system (except where otherwise noted). Several comments are included to describe the session, they are surrounded by "*/" and "*/".

The SARA commands are generally divided into two categories:

1. System Commands (preceded by @)
   These are commands available at any level of input and within all tools. They allow the user to alter the SARA input/output environment and additionally to request assistance (help).

2. SARA Commands (preceded by $)
   All non-System commands are preceded by $. This is a standard observed by all SARA tools, as well as the tool Selector.

In addition to the above commands, a "shell" may be entered at any point of input (including at the end of an incomplete input line) to receive assistance in several of the SARA tools, including the Selector.

This is the GMB translator. The size factor is an integer.

Indication of the size of the model being created.

Since the source will be read in from a file, it is necessary to request that the input be echoed to the terminal.

Output for input echo classes

Input source is <data filename of GMB model>

Control_graph

Data_graph

controlled_processors e1,e2,e3,e4,e5


datasets e0,e1,e2,e3,e4

datasets r

GMB Translator V. 15m June 1977

SARA Selector september 20, 1978

New or modified news:

no news changes

&>saratree

The example shown here is a GIS model simulating an input and output session for the calculation of a root of a function by Newton-Raphson approximation.

The first step is the creation of the gmb-equivalent of "primitive" used in the GMB model resides in a file and will be read in/ by the "GMB" command */

Behavior

SARA_Behavior.GMB

GMBTranslator V. 15m June 1977

GMBTranslator V. 15m June 1977

GMBTranslator V. 15m June 1977

SARA.Behavior.GMBTranslator

gmplex size factor = 7

Figure 21—UCLA SARA (System Architect's Apprentice) demonstration.
the parallelism existent in the problem. Figure 19 presents
the GMB model and Figure 20 represents the DDF model.
Both models should be viewed as if they were part of a
SARA closed design universe, i.e., the model of the system
to be designed is enclosed by a module structure and can
interact with other modules only through specified sockets.
It is assumed that outputs from other modules initialize the
model and provide the four inputs at the top of the figure.
It is assumed that the outputs at the bottom are received by
other modules and possibly tested for error.
Among the SARA tools there is a GMB translator which can accept descriptions of the control graph and data graph and a PLIP translator which accepts interpretations. Both can accept descriptions of the control graph and data graph.

In order to demonstrate the power of the GMB, a GMB simulation and a PLIP translator which accepts interpretations. Both can accept descriptions of the control graph and data graph.

Figure 21 (continued)
control graph. Templates for PLIP interpretations were used to make each uncontrolled processor behave like the corresponding DDF primitive. The resulting GMB description representing the DDF model was then translated and simulated by SARA at MIT-MULTICS via the ARPANET. The dialogue between user and SARA was captured and is incorporated in Figure 21 to complete the body of this section. In the interest of space, some details have been removed and comments have been added to notify the reader.

CONCLUSIONS

This paper has attempted to describe two token models which have been used to explore architectures taking advantage of concurrency. The primitives of the Dennis Data Flow Model and the SARA Graph Model of Behavior have been described and further understanding of their relationships was obtained by expressing one set of primitives in terms of the other. It was then demonstrated that data flow architectures could be explored making use of existing SARA tools. In particular, some investigators have found that developing a DDF model is very much like low-level design of a hardware system. The multi-level modeling supported in SARA gives some hope of expressing functions at a high level and refining them systematically.

REFERENCES