BUILDING INTERPROCESS COMMUNICATION MODELS USING STILE

Michael P. Stovsky and Bruce W. Weide

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
(ARPANET/CSNET: stovsky@ohio-state.edu, weide@ohio-state.edu)

Abstract

STILE provides a graphical environment for describing logical relationships among components of systems. The syntax of the graphs produced using STILE is separate from the semantics which are supplied by a post-processor. A major advantage of this approach is the ability to address different models of computation, especially different concurrency models, using the same graphical environment. This paper describes how to create primary building block parts for analog computing and data flow computing from parts defined in a different model of computing. Composing parts in this fashion eliminates the need to build post-processors for these additional models of computation, thereby overcoming one of the major problems introduced by separating graphical syntax from semantics.

1. Introduction

STILE (STructure Interconnection Language and Environment) is a general-purpose computer-based graphical design and development system for describing logical relationships among components of systems. Software systems created using STILE can be compiled into runable code by a post-processor. STILE allows software engineers to construct systems bottom-up by defining component parts and combining them. Similarly, systems may be constructed top-down, outside-in, or inside-out. Because parts are easily composed from other parts, hierarchical systems engineering is supported and reusability is facilitated.

The STILE environment follows the classical engineering design metaphor: each part has a catalog page containing specifications describing *what* the part does and a blueprint detailing *how* the part is constructed. A text editor is used to create and modify catalog page specifications. A syntax-sensitive graphical editor is used to construct blueprints describing part implementation details. The graphical editor does not allow the construction of syntactically invalid blueprints. However, the editor knows nothing about the meaning of blueprint contents. Semantic interpretation of blueprints is performed by a post-processor whose input is the structural description maintained by STILE. This separation of blueprint syntax from semantics allows STILE to be used to describe systems for a variety of computational models. STILE can be customized to address new models of computation either by constructing new post-processors or by combining parts from currently used models of computation so the resulting parts behave like those in the new one.

Previous papers on STILE concentrated on the philosophy, engineering design metaphor, syntax of STILE graphs, and the "STILE Basic interpretation" — an abstract concurrency model that separates data and control[1,2,3]. This paper describes how to assemble parts that behave like parts from analog computing and data flow computing. These models of computation are built from parts in the STILE Basic interpretation and, therefore, do not require their own post-processors.

Section 2 introduces the notion of an "interpretation," the context for supplying meaning to STILE graphs. Sections 3 and 4 concentrate on two useful interpretations corresponding to analog and data flow computation, respectively. Section 5 briefly discusses future work.

2. Interpretations

Each blueprint has an external and an internal section. The external section describes the part's connection points for communicating with the outside world. The internal section contains the part's implementation details. Each section has an associated *interpretation* that determines which semantic rules should be applied by the post-processor. In addition, parts with a particular external interpretation can only be instantiated in blueprints with that internal interpretation. For example, if the part being constructed has an external interpretation of Analog and an internal interpretation of STILE Basic, only parts with an external interpretation of STILE Basic can be instantiated in the internal section of the new part's blueprint. Similarly, the new part can only be instantiated in blueprints whose internal interpretation is Analog. As this example suggests, the two interpretations need not be the same. Hence, changing interpretations at the external/internal boundary serves as an abstraction mechanism. (Figure 10, explained in section 3.4.3, uses this capability.)

The primary building-block parts for a new model of computation and, consequently, a new interpretation of STILE graphs, can be created by an expert programmer from existing parts as described in this paper. Once these parts have been created, they are available to the community of programmers as "primitive" parts in the new interpretation. In other words, people using these parts to create graphs in the new model of computation are not aware the parts are compositions; they are not allowed to examine the blueprints created by the expert, as enforced by system administration policy. They may only see the catalog pages (specifications) for these parts. Aside from the obvious information hiding this provides, programmers are (unknowingly) using abstraction since they are describing graphs in the new model of computation by assembling parts composed in a different model of computation.

By exploiting STILE's part composition, abstraction, and information hiding capabilities, trees of the models of computation supported by each post-processor can be constructed. Engineers designing and constructing systems are able to select models of computation that precisely match the problem domain at each level of system decomposition. Because each part within STILE has an external interpretation and an internal interpretation which may differ, the interpretation applied to parts within a system may change as different levels of the system hierarchy are encountered.

As each new interpretation is added to STILE, its position in the tree of interpretations is specified. Figure 1 is the interpretation tree for the post-processor supporting the models of computation described in this paper. Arrows emanating from a node describe which interpretations are candidates for the internal portion of a blueprint whose external interpretation is the node value. For example, a graph whose external interpretation is Analog may have either Analog or STILE Basic as its internal interpretation.

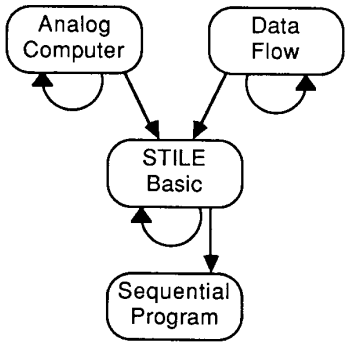


Figure 1: Interpretation Tree

3. Analog Interpretation

Digital computers use clocks to synchronize the activities necessary to carry out computations. The functional units of a digital computer are activated by clock pulses. Real numbers are approximated using finite-precision floating point numbers. On the other hand, in an analog computer the functional units (devices) operate continuously. There is no concept of a clock "tick" or device synchronization. At any time, the output of a device is a function of its input(s) and time. Furthermore, all numeric quantities in analog computers are proportional to voltage levels. These levels are free to vary within the circuits' electrical limits; they are not limited to discrete levels.

Ideally, the wires used to connect the units in an analog computer provide instantaneous transmission of voltage changes. In addition, an output wire may be connected to several inputs, thereby exhibiting fan-out. Since the output of an analog device is often fed back into the device's input, analog outputs may be dependent on the computation's history; i.e., analog computers have "memory".

This section explains how software based on the analog computing model can be built from more primitive parts using STILE.

3.1. Composition and Reusability

From a software perspective, analog computing has interesting properties related to reusability. Analog computers have been used to solve complex problems by combining existing circuits to form more powerful ones[4]. For example, as shown in Figure 2, multiplier, inverter, and operational amplifier circuits can be combined to form a square root circuit. (An inverter negates its input. The output of an operational amplifier is a very large constant times the sum of its inputs.) This square root circuit can be considered a new part which may be used independently or as a building block in more powerful circuits.

The ability to construct such hierarchies of parts within an interpretation is captured by the *composition property*[5]. This property can be summarized as follows. A compound agent should behave abstractly like a primitive agent; it should communicate and be composed just like primitive agents; and the composition operations should be independent of the internal structure of the agents to which they are applied. Finally, the meaning or function of a compound agent should be uniformly definable in terms of its constituents. Analog computer components satisfy this property, which makes them interesting from the standpoint of studying reusability.

Historically, analog computers have served as the basis for real-time controllers. Figure 3 is a simple model of a feedback control system. In such a system, the setpoint is the desired value of the controlled variable. For example, the temperature setting on a thermostat is a setpoint. A control engineer synthesizes a control algorithm to

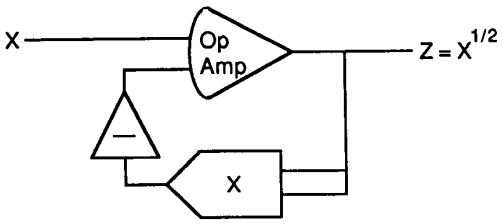


Figure 2: Square Root Circuit Built from Simpler Parts

influence the system to insure the value of the controlled variable tracks the setpoint. In order to synthesize a stable control algorithm, the engineer must account for the dynamics of the controlled system. Inability to correctly account for system dynamics can result in an unstable system which may exhibit wild fluctuations of the controlled variable's value.

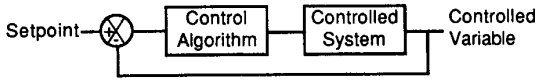


Figure 3: Feedback Control System

STILE's three part types can be mapped to analog computer components in a straightforward fashion. Specifically, STILE boxes denote functional units, links denote wires, and ports denote electrical terminals. This informal semantic assignment is called the *Analog* interpretation. We have found this interpretation to be useful for designing real-time control software[4,6,7]. Constructing a STILE interpretation for analog computing provides a vehicle for exploring software reusability as well as an environment to solve real-time software problems.

The Analog interpretation provides a framework with which a control engineer can build software simulations of analog systems. However, processing delays involved in this software are analogous to bandwidth limitations of the analog components, have a similar impact on stability, and must be analyzed using feedback control theory. STILE supplies the syntactic and semantic constructs needed to design these systems, but does not insure their stability. Stability, an independent property, must be verified by the control engineer.

3.2. A Sample Analog System

Figure 4 is a blueprint (implementation) for the software control of a Heating Control System (HCS). (Each caption in this paper specifies the external and internal interpretations as follows: (external interpretation/internal interpretation). The internal interpretation is not revealed in a figure if it is not revealed to the user of the system, as in catalog pages.) The Dial box reads the desired room temperature (the setpoint). Temp is a thermometer returning the actual temperature (the controlled variable). The Subtract box computes the difference between these two quantities. Furnace turns on the heating element and fan if the room temperature is too far below the setpoint. Once the temperature is high enough, Furnace turns the heating element and fan off.

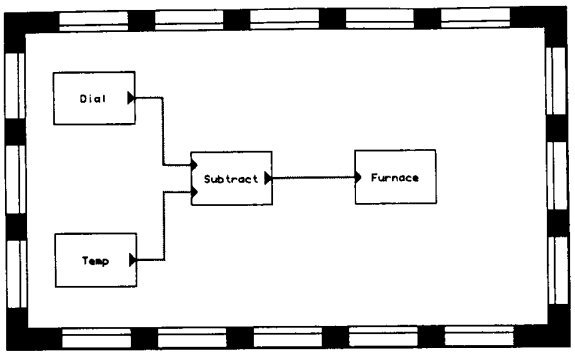


Figure 4: Heating Control System (Analog/Analog)

Figure 5 is the catalog page (specifications) for the Furnace box and Figure 6 is its blueprint. The Range box exhibits hysteresis so the heating element and fan do not oscillate on and off. Constants supplies the Range box with a sensitivity value. If the difference between the desired and actual temperatures is greater than this value, the heater and fan are turned on. Similarly, when the room temperature exceeds the setpoint plus this sensitivity value, the heater and fan are turned off.

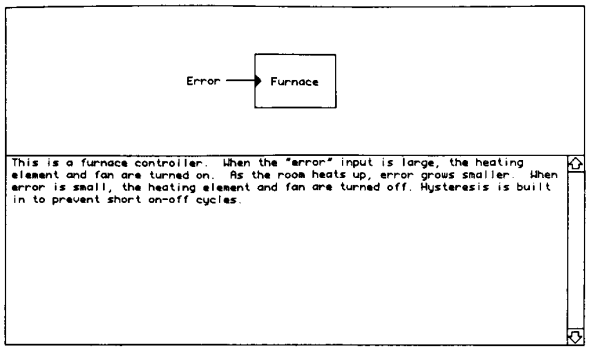


Figure 5: Furnace Catalog Page (Analog)

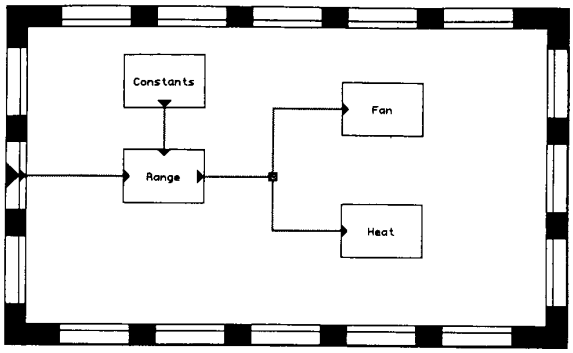


Figure 6: Furnace Blueprint (Analog/Analog)

3.3. The STILE Basic Interpretation

Since the objective is to show how Analog interpretation semantics can be "created" from STILE Basic interpretation parts, this section briefly summarizes the STILE Basic interpretation. A previous paper contains a detailed explanation of the STILE Basic interpretation[3].

In the STILE Basic interpretation, the internal section of a box may contain text which describes sequential computations in a typical sequential programming language, or it may contain a STILE Basic interpretation graph, thereby allowing hierarchical design. The interpretation of a box containing a graph is as though its internal section were substituted for its icon wherever the box is used. Each box in a completely expanded (flat, or single-level) graph represents a single sequential process. Each process operates concurrently and communicates with the other processes via its ports and according to its link bindings.

Ports are of two major types: data ports and control ports. Data ports are used to communicate data with other boxes. An input data port is like a register that can be read by the box. It keeps its current value until it is overwritten by another box writing to an output data port connected to the input port by a link. Control is communicated through input and output control ports. An input control port is like a single flip-flop. Each box has a monitor that keeps track of the states of its input control ports. An input control port is set when a box "pokes" an output control port connected to it by a link. The monitor continually checks to see if any control input ports are set. If so, it selects one of the set input ports arbitrarily, resets the port, runs a corresponding sequential code segment to completion, and resumes monitoring the input control ports. Internal communication among separate code segments of a box is by shared data. All communication between separate boxes is by communication protocols and data transmission via ports and links. Links in the STILE Basic interpretation are simply extension cords which bind together the ports of boxes.

3.4. Implementing the Analog Interpretation in STILE

In the STILE Basic interpretation, data and control are transmitted as discrete entities and boxes compute only when their control inputs are poked. On the other hand, wires in analog computers continuously carry meaningful values to functional units which continuously compute with them. Despite these differences, STILE Basic interpretation parts can be arranged to simulate analog computer parts, as described below.

3.4.1. Analog Ports

Figure 7 shows a STILE port in the Analog interpretation, called "Pin." A Pin represents an electrical terminal to which wires can be connected. The Pin is composed of a data port and a control port. When a box computes a value, it writes the value to the data port of the Pin and pokes the Pin's control port, thus signalling a fresh value has been computed.

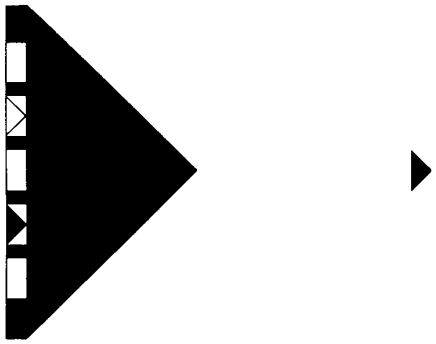


Figure 7: Pin Blueprint (Analog/STILE Basic) and Icon

3.4.2. Analog Links

Figure 8 is the blueprint of a STILE link in the Analog interpretation, called "Wire." The Wire part consists of a Pin at each end with the data and control ports connected to their duals. Wire forwards the data and control signals from the source to the destination, exactly what is needed. The values produced by a box may be transmitted to several other boxes, but "wired ORs" are not allowed. Consequently, a Wire may exhibit fan-out but not fan-in. This syntax rule is described in Wire's catalog page by allowing multiple output connections, but not multiple input connections, as shown in Figure 9.

3.4.3. Analog Functional Units

Figure 10 is the blueprint of a typical STILE box in the Analog interpretation, called "Subtract." It is constructed from a STILE Basic Subtract box. When either of the Analog Subtract box's input

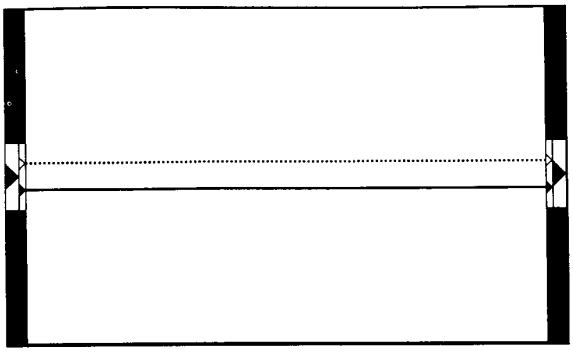


Figure 8: Wire Blueprint (Analog/STILE Basic)

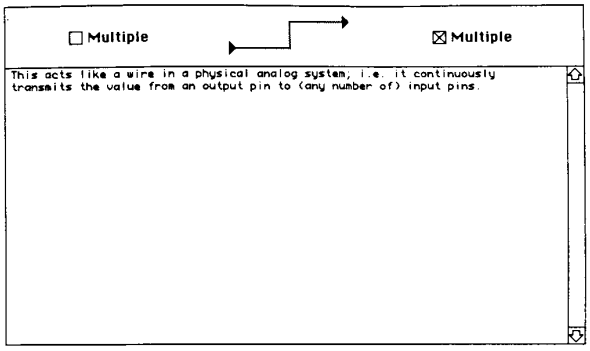
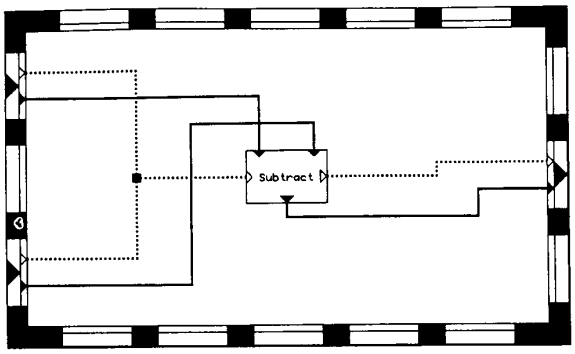


Figure 9: Wire Catalog Page (Analog)

Pins has a new value (i.e., the control input port is poked) the subtrahend is subtracted from the minuend and the difference is available on the output Pin (i.e., the difference is written to the output data port, and the output control port is poked). Note the box performing the subtraction is a STILE Basic Subtract box which has not been altered in any way. It has been instantiated in Analog Subtract's blueprint and connected to the Pins on the box's periphery using STILE Basic control and data links. Once created, the Analog Subtract box can be instantiated wherever it is needed without further modification, as in Figure 4. Other Analog functional units are similarly constructed from their STILE Basic cousins.

For some Analog boxes, such as Dial in Figure 4, there are no input Pins. Hence, there is no external triggering mechanism to serve the function of the control port of an input Pin. This kind of Analog box is triggered internally. For example, Dial might be constructed from a STILE Basic box that samples an A/D converter whenever its input control port is poked, and whose output control port is fed back into its input control port. Figure 11 is the blueprint for this implementation of Dial. Alternatively, the stimulus for sampling may come from a STILE Basic "Clock" box that periodically pokes its output control port[7].



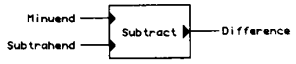


Figure 10: Subtract Blueprint (Analog/STILE Basic) and Icon

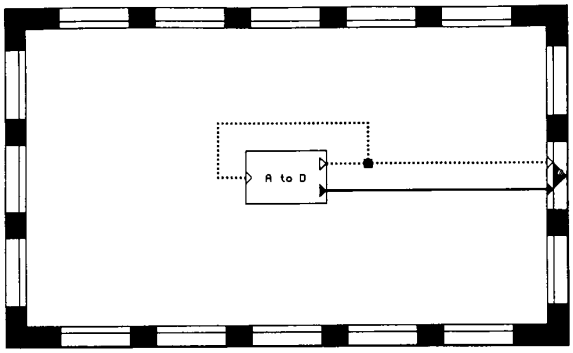


Figure 11: Dial Blueprint (Analog/STILE Basic)

3.5. An Improved Implementation

Figure 12 is the blueprint for an improved Wire. The STILE Basic "Change" box propagates control and data signals it receives only when the new data value differs from the value last written to the output data port. The blueprint for the Change box is shown in Figure 13. Substituting the improved Wire link for the original Wire has no effect on the functionality of an Analog system, but reduces the computation load in a nearly-quiescent system by preventing a box from recomputing its outputs when no inputs have changed. In a real-time control application this may be an important factor in responsiveness and in keeping hardware costs to a minimum[1].

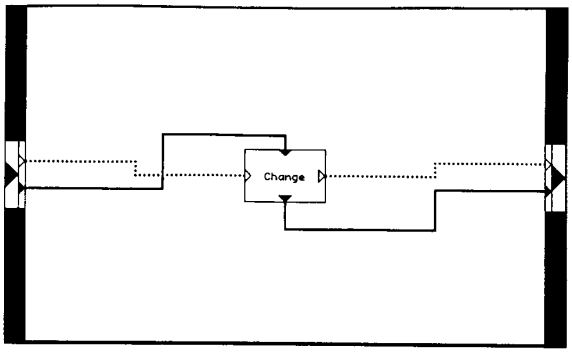


Figure 12: Improved Wire Blueprint (Analog/STILE Basic)

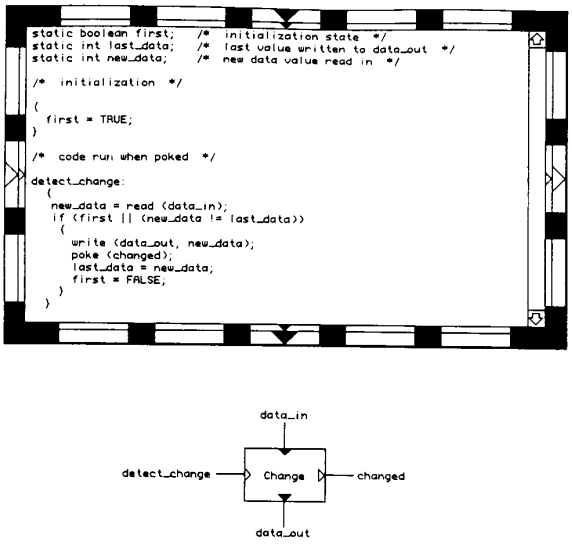


Figure 13: Change Blueprint (STILE Basic /STILE Basic) and Icon

4. Data Flow Interpretation

Data flow graphs are useful computational models[8] as well as an analysis and design tool[9,10]. This section explains how to construct the parts necessary to assemble data flow graphs, using STILE Basic parts, based on Dennis' data flow model of computation.

In this model, data tokens flow along arcs connecting functional units. A unit may perform its function as soon as all of its input data tokens have arrived. Once the function is computed, output token(s) are created, and the unit awaits a new set of input tokens. At no time can an arc have more than one token on it. Hence, a unit cannot output a new token until its previous output has been consumed (by a downstream unit) *and* all inputs are available.

An example data flow graph is shown in Figure 14. It computes the area and volume of a sphere, given its radius r:

$$\text{Area} = 4\,\pi\,r^2 \qquad \text{Volume} = 4\,\pi\,r^3/3$$

Squares represent data inputs and outputs which provide or remove the indicated values. Circles represent functional units which perform the indicated operations. In this example, all functional units happen to multiply their inputs together.

Figure 15 shows an equivalent STILE graph, using the *Data Flow* interpretation. STILE boxes represent functional units, links represent the arcs along which data tokens flow, and ports represent the input and output connections on the functional units.

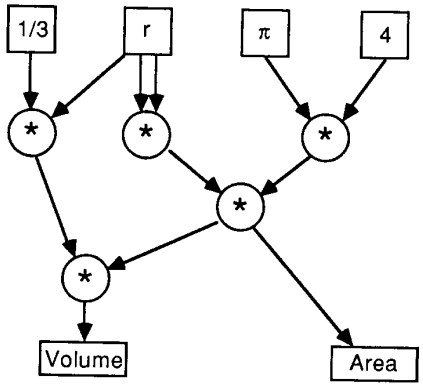


Figure 14: Data Flow Solution for Volume and Area of Sphere

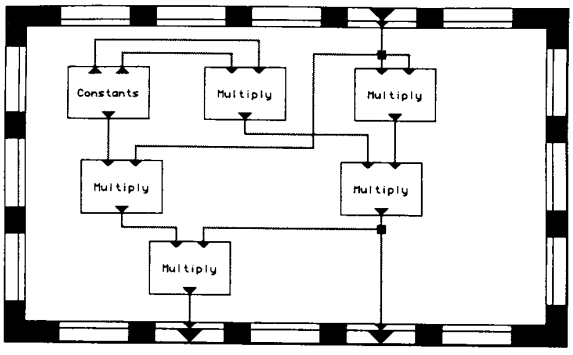


Figure 15: Sphere Blueprint (Data Flow/Data Flow)

4.1. Implementing the Data Flow Interpretation in STILE

It is possible to construct data flow elements from STILE Basic interpretation parts to capture the semantics of data flow graphs in much the same way as these STILE Basic parts were combined to build the Analog interpretation in Section 3. The parts created include data flow ports, data flow links, and data flow boxes.

4.1.1. Data Flow Ports

Figure 16 is a data flow port. In order to be faithful to the data flow paradigm, it is necessary to insure there can never be more than one token on a link at a time. This is accomplished by "handshaking" involving the control ports. When a data value (the value carried by a data token) is written to a data port, the corresponding control port is poked. This informs the recipient a token is available. When this box consumes the data token, it pokes the return control port, informing the token generator it may generate a new token.

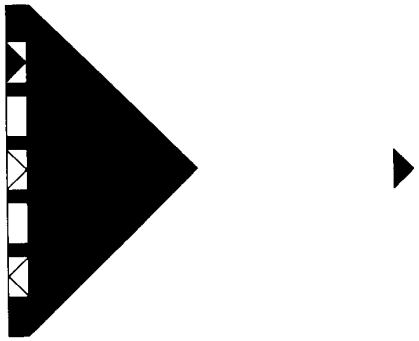


Figure 16: Port Blueprint (Data Flow/STILE Basic) and Icon

4.1.2. Data Flow Links

Externally, a data flow link appears to be an extension cord which can connect data flow ports. Further, the link allows fan-out to occur so multiple token recipients may be connected to a single token generator. The protocol described above works only if there is no fan-out from any data token generators, however. When there is fan-out, all copies of the token must be consumed before a new token can be generated. Even in the simple area and volume evaluation example, several token generators exhibit fan-out. Consequently, the data flow port has to be augmented to model a data flow computation correctly. This modification occurs within data flow links. Figure 17 is the blueprint for such a link.

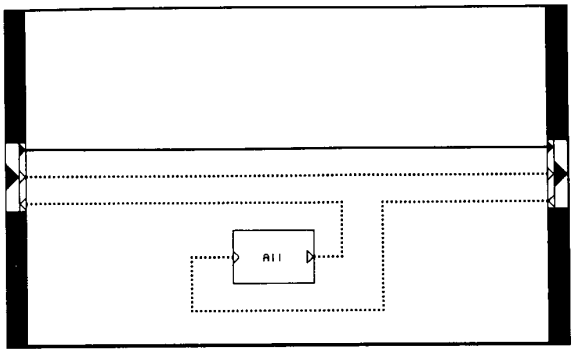


Figure 17: Link Blueprint (Data Flow/STILE Basic)

Fan out capability is due to the presence of the STILE Basic "All" box. When all the control ports connected to an All box's input control port have been poked, the All box pokes its output control port. Hence, all downstream data boxes must consume their data tokens before the data source can generate another token.

The All box is crucial to correct operation of data flow links. The input control port of All is a vector port. The size of the vector depends upon the degree of fan-in (i.e., the degree of fan-out from the data flow link). The STILE Basic interpretation processor expands the single port into as many distinct ports as there are connections to it in the graph. These distinct ports can be accessed individually by the code inside the All box.

By using an input control port which is a vector port, only one versatile All box need be constructed. Figure 18 shows the blueprint of such an All box.

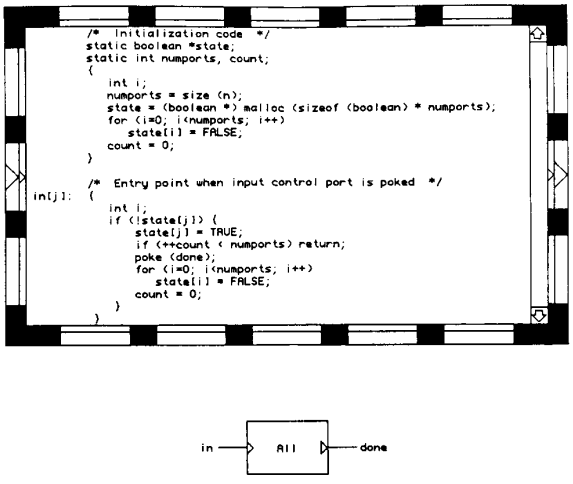


Figure 18: All Blueprint (STILE Basic/Sequential Program) and Icon

The "in" port is a vector port. A boolean state vector is maintained to remember which input control ports have been poked. The first chunk of code is initialization code, run only when the system starts up. The "size" function returns the number of output control ports connected to the All box's input control port (i.e. the length of the vector). This information is used to allocate storage for the state vector. The state vector is then cleared since no ports have been poked. Once all of the constituents of the vector port have been poked, the All box pokes its output control port and resets the state vector.

Notice vector ports are not created by graphical means, but by cues in the text of parts whose internal interpretation is Sequential Program. Processing of the STILE graph structure in conjunction with this text provides the vector port feature. Recall data flow programmers cannot see the blueprint for a data flow link since they believe such links are primitive and need not be concerned with such details of the STILE Basic interpretation.

4.1.3. Data Flow Functional Units

In Dennis' model of data flow, a functional unit cannot proceed with its computation until its output tokens have been consumed and all input tokens are available. Boxes must be designed so this rule is not violated. Figure 19 is the blueprint of the Data Flow "Multiply" box. An All box is used to insure the computation cannot proceed until all input tokens have been generated and previous output tokens have been consumed. Once the appropriate tokens have been generated and consumed, the All box pokes its output control port which activates the STILE Basic Multiply box. Upon completion of its computation, Multiply pokes its output control port which informs the downstream units a token has arrived. The same poke is used to inform the upstream units their tokens have been consumed.

4.1.4. System Initialization

Upon system start up, each box needs to request tokens from upstream units. This is accomplished by initializing the data flow link's return control port to "poked" when the data flow link part is created. As a result, engineers instantiating links need not be concerned with this aspect of start up.

5. Future Work

We continue to look for new areas of application for STILE. One goal of this work is to learn more about reusability, composition, and hierarchical systems design and apply this knowledge to software. In addition, we are trying to find and define the limits of the STILE approach to systems design and development. Areas currently being investigated include creating interpretations for image processing, module interconnection, processor assignment, and general software design. In the future we plan to use STILE to create parametrically configured parts for discrete event simulation, computer graphics, and computer music. By using the STILE environment we hope

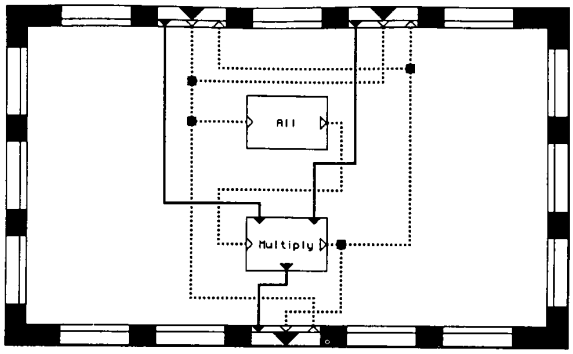


Figure 19: Multiply Blueprint
(Data Flow/STILE Basic)

to create software platforms of reusable parts from which to build systems for a variety of problem domains.

6. Summary

STILE is a general-purpose design and development environment. By separating graph syntax from semantics, STILE provides a rich environment which can be applied to many problem domains. New models of computation can be introduced to the system by either building new post-processors to interpret the graphs differently or by assembling existing parts so the result behaves like a part in a different model of computation. Engineers creating systems using parts from these new models of computation need not be aware the parts are composed from other parts. Consequently, information hiding and abstraction are strongly supported. In this paper we have illustrated how to build basis parts for Analog computing and Data Flow computing from STILE Basic parts. The fundamental parts for other models of computation can be similarly constructed.

Acknowledgements

We would like to thank Mark Brown, Douglas Harms, Peter Maurath, Robert McGhee, William Ogden, Karsten Schwan, Sanjiv Taneja, Gregor Taulbee, and Stuart Zweben for their ideas and valuable contributions to this work.

Generous support for this project from The Ohio State University and the Department of Computer and Information Science are gratefully acknowledged.

References

[1] Brown, M.E., and Weide, B.W., "Automating Process-to-Processor Mapping Under Real-Time Constraints," in *Proc. 1984 Real-Time Systems Symposium*, IEEE, Austin,TX, December, 1984, pp. 145 - 150.

[2] Taneja, S., and Weide, B.W., "Graphical Description and Run-Time Environments for Real-Time Software," in *Proc. ACM Computer Science Conference*, ACM, Cincinnati, February 1986, pp. 205 - 211.

[3] Stovsky, M.P., and Weide, B.W., "STILE: A Graphical Design and Development Environment (Extended Abstract)," in *Digest of Papers, Spring COMPCON 87*, IEEE, San Francisco, February, 1987, pp. 247 - 251. Full report available from Department of Computer and Information Science, The Ohio State University, Columbus, Ohio, OSU-CISRC-86TR1BWW, October 1986.

[4] Jackson, A.S., *Analog Computation*, McGraw-Hill Book Company, 1960.

[5] MacQueen, D.B., "Models for Distributed Computing," Rapport de Recherche No. 351, April 1979, Institut de Recherche d'Infromatique et d'Automatique (IRIA), Le Chesnay, France.

[6] Ogata, K., *Modern Control Engineering*, Prentice-Hall, Inc., 1970.

[7] Brown, M.E., and Weide, B.W., "Preliminary Design of a Highly Parallel Architecture for Real-Time Applications," in *Proc. 18th Anual Allerton Conf. Communications, Control, and Computing*, October 1980, pp. 534 - 543.

[8] Dennis, J.B., "First Version of a Data Flow Procedure Language," *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, 1974, pp. 362 - 376.

[9] Yourdon, E., and Constantine, L.L., *Structured Design*, Yourdon Press, 1978.

[10] DeMarco, T., *Structured Analysis and System Specification*, Prentice-Hall, 1979.