7 1111

Weaves as an Interconnection Fabric for ASIMs and Nanosatellites

Michael M. Gorlick
The Aerospace Corporation
P.O. Box 92957
Los Angeles, California 90009
gorlick@aero.org

Abstract

Many of the micromachines under consideration require computer support, indeed, one of the appeals of this technology is the ability to intermix mechanical, optical, analog, and digital devices on the same substrate. The amount of computer power is rarely an issue, the sticking point is the complexity of the software required to make effective use of these devices.

Micromachines are the nanotechnologist's equivalent of "golden screws," in other words, they will be piece parts in larger assemblages. For example, a nanosatellite may be composed of stacked silicon wafers where each wafer contains hundreds to thousands of micromachines, digital controllers, general-purpose computers, memories, and high-speed bus interconnects. Comparatively few of these devices will be custom designed, most will be stock parts selected from libraries and catalogs. The novelty will lie in the interconnections, for example, a digital accelerometer may be a component part in an adaptive suspension, a monitoring element embedded in the wrapper of a package, or a portion of the smart skin of a launch vehicle. In each case this device must inter-operate with other devices and probes for the purposes of command, control, and communication.

We propose a software technology called *weaves* that will permit large collections of micromachines and their attendant computers to freely intercommunicate while preserving modularity, transparency, and flexibility. Weaves are composed of networks of communicating software components. The network, and the components comprising it, may be changed even while the software, and the devices it controls, is executing. This unusual degree of software plasticity permits micromachines to dynamically adapt the software to changing conditions and allows system engineers to rapidly and inexpensively develop special-purpose software by assembling stock software components in custom configurations.

1 Introduction

Without extensive software support nanomachines, microdevices, and application-specific integreated microinstruments (ASIMs) are just so much fancy dirty glass. Indeed from the perspective of a computer scientist many of the devices being proposed can be regarded as multicomputers with "unusual" peripherals. Another perspective, one which emphasizes their information content, regards these constructions as collections of sensors, actuators, and transmuters, whose purpose is to obtain, produce, and transform information. Even small assemblages may require significant amounts of software. For example, a modern rechargable electric razor contains about 2 kilobytes of software, a digital thermostat about 12 kilobytes, and an automotive emissions control system contains in excess of 300 kilobytes of software. Satellites based on nanotechnology will contain a wide assortment of interconnected digitally mediated subsystems including attitude control, power, navigation, communication, and sensors whose combined software elements may easily exceed tens of megabytes. The economic assembly of such systems will require:

- software that can be assembled component-wise from stock piece-parts;

- software that can be flexibly reorganized to cope with the introduction of novel components or new combinations of common subassemblies; and

- software that can be dynamically reconfigured to compensate for hardware failures or changes in the mission of the satellite.

One medium that addresses these issues is weaves, a component-based approach to software composition and interconnection. Weaves are networks of components in which streams of arbitrary objects flow from one component to another. They occupy a computational niche midway between fine-grain dataflow and large-grain stream processing (as exemplified by Unix pipes and filters). A detailed discussion of the computational and communication semantics of weaves (including the features that distinguish it from data-flow languages like Khoros[2], Show-and-Tell [6], and Prograph [1]) can be found in [5]. We have implemented a visual software composition and integration environment for constructing systems as weaves. The weave visual editor, *Jacquard*, provides users with mechanisms for rapidly assembling weaves from components, executing and observing weaves, and combining and modifying weaves dynamically — all using nothing but point, click, drag, and drop. A more detailed view of the environment for constructing weaves can be found in [3].

Weaves are well suited for systems characterized by processing on continuous or intermittent streams of data, and they have been applied to such tasks as satellite telemetry processing, tracking, and the rapid prototyping of satellite ground stations. Figure 1 shows a portion of a stereo tracker implemented as a weave. Weaves are comprised of sockets (which are either unpopulated or populated), tool fragments, and jumpers. Unpopulated sockets are placeholders for tool fragments that consume objects as inputs and produce objects as outputs. Jumpers between sockets provide transport services for moving objects from one place to another and thus define the topology of the network. Users assemble weaves by interconnecting sockets and populating each socket with a tool fragment (which may itself be a weave). Type information about the objects flowing through a connection is specified by labeling the jumper.

Weaves adhere to the three rules of *blind communication*:

- no tool fragment in the network is aware of the sources of its input objects or the destinations of its output objects, consequently, all tool fragments are independent of their position in the topology of the network;

- no tool fragment is aware of the semantics of the transport services that are used to deliver its input objects or transmit its output objects, consequently, new transport services can be freely substituted for old.

- no tool fragment is aware of the loss of a connection and to the extent that the computation can continue it will, consequently, weaves can by dynamically edited and rewired without risking the integrity of the weave.

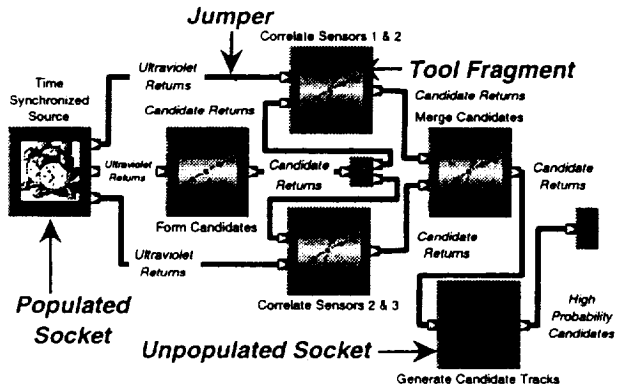

Figure 1: A portion of a weave-based stereo tracker.

Weaves were specifically designed to tackle the problem of constructing large systems by composing components and interconnections — the visual equivalent of a module-interconnection language. A weave that contains one or more unpopulated sockets can be thought of as a framework for an entire family of implementations that are customized by populating empty sockets with components. Figure 2 illustrates such a framework for a sensor and its controller. The sensor is connected to the controller via a feedback loop through which the controller issues commands and control messages to the sensor and receives sensor-specific measurands and status information.

This weave can be used as a tool fragment in some higher level construction since it contains an input pad that passes commands and controls to the controller socket in from the outside and an output pad that transmits measurands and status from the controller socket to the outside. Figure 3 illustrates a particular

instantiation of the framework, an integrated vibration sensor, that contains the tool fragments to command and control a vibration microsensor and deliver its measurements to some higher level device.

In the following sections we illustrate how weaves can be used to integrate large numbers of diverse microinstruments. Two different approaches are shown. In Section 2 we examine a weave framework designed to support a small number of tightly coupled microinstruments bound together as a multiparameter sensor. In Section 3 we outline a weave framework based on message buses that is suitable for large numbers of loosely coupled subsystems such as that found on a nanosatellite. Finally in Section 4 we frame some of the research questions that must be resolved to make this approach viable.

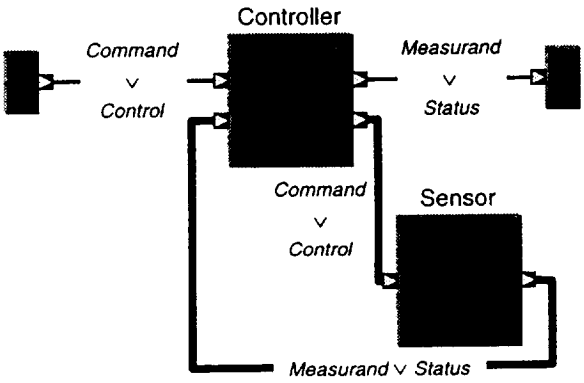

Figure 2: A generic framework for a sensor and its controller.

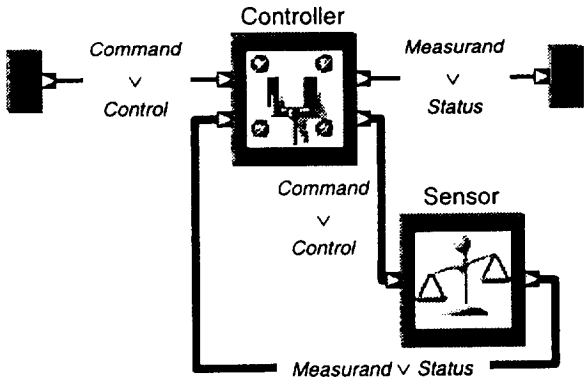

Figure 3: A instantiation of the generic sensor framework as a vibration sensor.

2 Weaves for Multiparameter Sensors

A multiparameter sensor module combines several sensors on a single substrate. A simplified layout of one such hypothetical module is shown in Figure 4. This module provides location, acceleration, sound level, and the detection of one or more chemical species and contains an wireless link for the transmission of telemetry and the receipt of commands. It would be powered by a thin film battery layered on the underside of the substrate and would be so small (certainly no larger than a pack of cigarettes) that it could pasted almost anywhere such telemetry might be required. Possible applications include environmental monitoring, industrial process control, and launch vehicle data acquisition. Specialized versions of such modules could be incorporated into the skins and structures of aircraft or trucks, or strategically placed on bridges or within buildings.

Multiparameter sensors offer three advantages over a comparable collection of independent sensors:

- all of the sensor readings are location correlated, that is, all measurands are being collected at the same location (within a few tens of millimeters);
- the individual instruments can be tightly coupled, for example, their sampling rates can be phased or synchronized;
- the activity or sampling frequency of one instrument can be made dependent on the measurands of another, for example, a multiparameter sensor in a rocket motor compartment could increase the sampling frequency of its temperature sensor when the vibration level crosses a preprogrammed threshold.

It makes sound development and economic sense that it should be as easy to construct control software for a multiparameter module as it is to construct the multiparameter module itself, that is, by combining and interconnecting stock piece parts into a cohesive whole. Logically the integration of an N parameter sensor should be the flat composition of N individual sensors, that is, for the multiparameter device that we are considering all sensors are peers equitably sharing a common substrate and resources such as power bus and specialized services such as analog/digital conversion. In the generic sensor framework sketched in the introduction each individual sensor is comprised of a

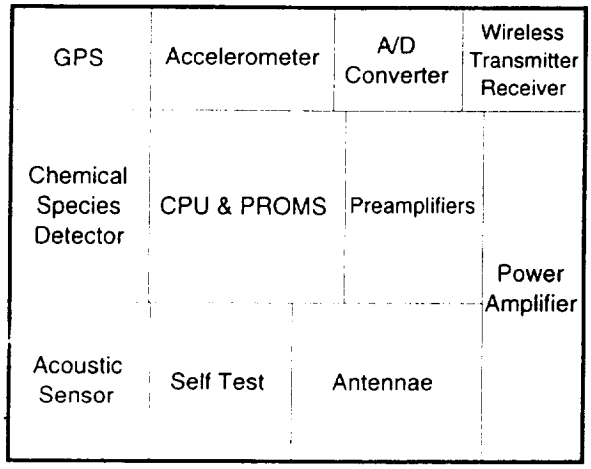

Figure 4: A hypothetical multiparameter sensor module.

controller and the sensor proper. This base organization suggests that a multiparameter sensor be organized hierarchically where each sensor device has its own local controller but reports to, and is commanded by, a higher order controller which is responsible for coordinating the activities of the individual sensors and mediating resource contention.

A sample weave framework for such an organization is shown in Figure 5. The framework is designed to accomodate four independent devices; the changes required for a different number of devices are obvious. The framework supports three different principal classes of tool fragments. Moving from right to left in Figure 5 the first class is represented by the "device" sockets which are intended for tool fragments that are the "embodiment" of the individual sensor devices. These tool fragments are weaves in their own right whose general form is suggested by Figures 2 and 3. Since weaves are indifferent to the composition or form of the tool fragments that populate the sockets of the framework these sensors can be arbitrarily complex.

The second class is represented by the unpopulated "router" socket in the middle of the network. Routers are responsible for the distribution of command and control messages from the higher level controller in the multiparameter sensor to the individual sensor devices. The router helps to insulate the controller from the multiplicities of the sensors, and to a certain extent, from some of the characteristics of the sensors themselves. The router can perform protocol conversions or command translations to supply a uniform unvarying interface to the controller.

The third and final class is represented on the far left by the "controller" which is responsible for accepting higher-level command and control and transforming that into individual sensor commands. Using a feedback loop analogous to that which appears in the individual sensors it also accepts the N-way merged output of the N sensors. Like the lower-order devices that it controls, it has input and output pads thereby permitting the entire multiparameter sensor itself to be embedded in some higher-order device.

A particular instantiation of this framework is illustrated in Figure 6 where a wireless transceiver, an accelerometer, a chemical species sniffer, and an acoustic sensor are combined into a single integrated multiparameter sensor. To the extent that the individual sensors observe a common command and control protocol the high-level controller and its matching router can be generic elements. Note that the same intermixing of custom and stock software components can be applied to both the controller and router which, like any other tool fragment, may be weaves in their own right. The combination of weave frameworks and the hierarchical composition of weaves permit one to construct the software for highly integrated, tightly coupled multidevices in a straightforward and elegant manner.

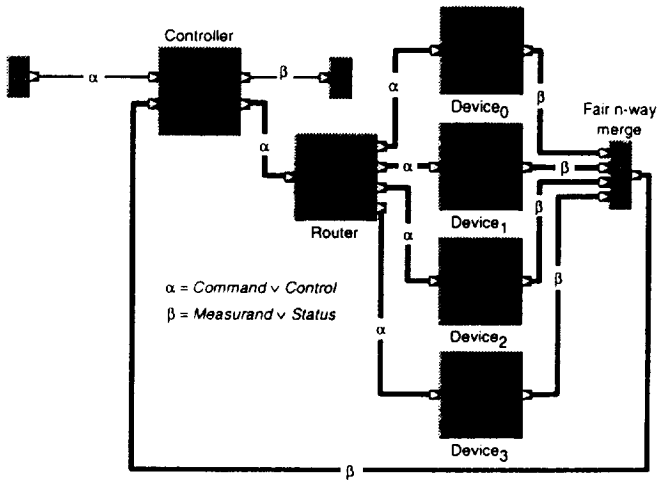

Figure 5: A weave framework for a generic multiparameter sensor.

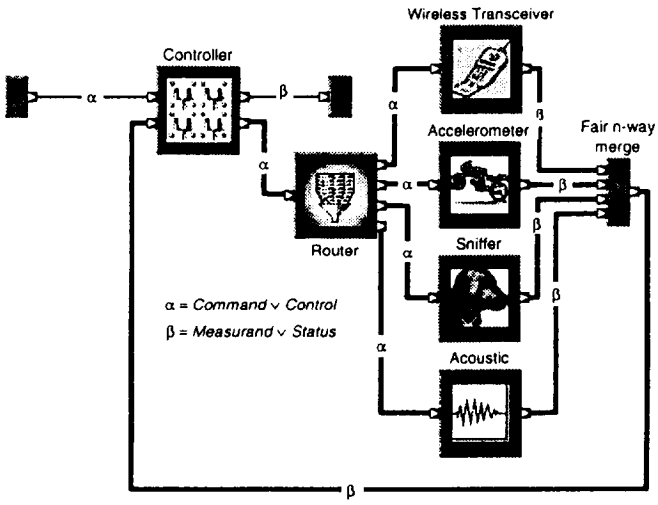

Figure 6: A sample weave for a specific multiparameter sensor.

3 Weaves for Nanosatellites

Unlike a tightly integrated multiparameter sensor a nanosatellite will be composed of a number of loosely integrated subsystems. Those subsystems in turn will encompass a broad degree of integration and coupling ranging from lightly coupled systems that communicate infrequently or irregularly to highly coupled, synchronized systems that require substantial bandwidth. The multiparameter sensor described in Section 2 illustrates some of the techniques required for tight coupling among components. The interconnections among weave components in Figure 6 are all point-to-point which is suitable for components that intercommunicate frequently. However new components can not be added to the weave without rewiring. While weaves fully support rewiring on-the-fly during execution dynamic editing may force the system into an unsafe state due to temporal or behavioral constraints. Furthermore complex systems contain cooperating subsystems that exhibit a variety of communication behaviors ranging from infrequent, low bandwidth communication to regular, frequent, high volume message traffic.

A highly integrated device such as a nanosatellite will be composed from independent subsystems that are mechanically, electrically, or optically integrated with one another. For example, one might construct a nanosatellite by stacking and bonding individual wafers where each wafer is a stock subsystem (guidance, batteries, power management, ...). Ideally the nanosatellite software should be able to recognize the stacking arrangement and arrange its communication paths accordingly. This capability would give system designers the freedom to insert custom subsystems without changing the software base that supports the stock subsystems.

This capability will prove increasingly important as swarms of nanosatellites cooperate to accomplish a task. Hundreds of nanosatellites in close physical proximity could organize themselves as a giant phased array thereby allowing space system architects to assemble on-orbit powerful communications "megasatellites" from individual satellites the size of a tea saucer. It would be advantageous if the software structures of the individual nanosatellites generalized smoothly to the software structures required for the control and management of swarms.

Hierarchical message buses [4] are flexible communication architectures that hold the promise of scaling smoothly from a single nanosatellite to large groups of independent, coordinated nanosatellites. We discuss below a bus-like communication structure that significantly reduces the amount of coupling and therefore may be more appropriate for a collection of semi-independent, cooperating subsystems.

Figure 7 illustrates a framework for a single message bus. Imagine a weave assembled on a sheet of paper where the sheet itself is a broadcast medium for the transmission of objects. Taps into the medium allow sockets to receive messages broadcast on the sheet or to themselves send broadcast messages over the sheet. This arrangement is typified by the socket *Alpha* whose inputs arrive from a tap (an *on-sheet receiver*) and whose outputs are in turn transmitted (via an *on-sheet transmitter*) to any other socket that has a comparable tap into the sheet. As a consequence of blind communication it is impossible for a tool fragment seated in the *Alpha* socket to determine if its inputs are arriving courtesy of a point-to-point connection or are obtained from a bus tap. Similarly, the same tool fragment can not discover that its outputs are being placed on the sheet bus for broadcast.

Sheets (message buses) can be cross-wired using *off-sheet* transmitters and receivers. Each sheet has a unique name. An off-sheet receiver is shown in Figure 7 that is receiving broadcast messages from a sheet named *Source*. All of the messages broadcast on sheet *Source* are being fed as inputs into the socket *Beta*. Likewise all objects output by any tool fragment populating socket *Gamma* will be broadcast on the sheet named *Sink* by the off-sheet transmitter attached to the output pad of *Gamma*.

Finally sheets, like any other weave, are permit-

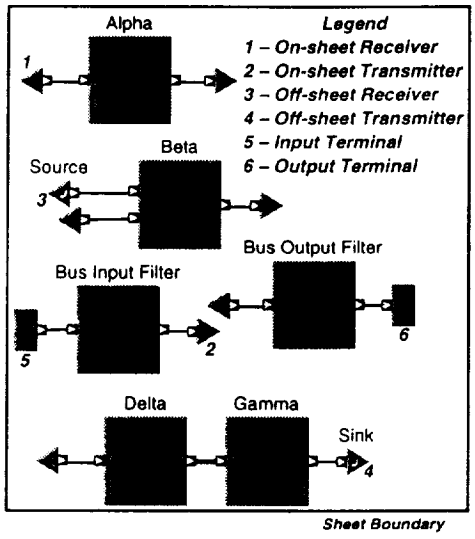

Figure 7: A generic weave bus.

ted input and output terminals. Consequently a sheet can be encapsulated as a tool fragment and may appear as a component in some higher-order weave. This permits message buses to composed hierarchically and dramatically reduces the scope and volume of the object traffic on any one sheet (bus). The combination of hierarchical composition and inter-sheet crosswiring via off-sheet receivers and transmitters allows system architects to construct complex multicast architectures that scale as the number of sheets (subsystems) increases.

To illustrate some of the possibilities we briefly sketch a high-level nanosatellite software architecture as shown in Figure 8. At the top level the spacecraft is organized as a single sheet (message bus) with a subweave responsible for each individual major subsystem. The individual subsystems are each constructed using a combination of point-to-point and bus topologies. A sample framework for the power subsystem is given in Figure 9. It bears a strong resemblance to the framework for the integrated multiparameter sensor shown in Figure 6 with a few important differences. In the power subsystem all of the outputs of the controller are fed into a filter that generates two granularities of monitoring and status information. The monitoring and status information that is fed to the output terminal is of coarser grain than that fed to the on-sheet transmitter. The message traffic appearing on the output terminal is a summary of the activities of the power subsystem that is suitable for processing by a higher-level controller or monitor. The message traffic appearing on the power sheet itself is a finer-grain, more detailed view of those same activities.

The flexibility of weaves and the bus-based architecture outlined above make it possible to add monitoring elements to the spacecraft architecture while the craft is on-orbit without extensive weave "rewiring." Figure 10 illustrates this approach. A specialized monitor has been added to the top-level spacecraft software architecture. The inputs of the monitor are derived from two sources, the spacecraft sheet and the power subsystem sheet using an on-sheet and an off-sheet receiver respectively. Note that this modification is completely transparent to the power subsystem and because no rewiring was required can be performed without endangering the integrity of the craft as a whole. When the monitor is no longer required it can be safely removed and the software restored to its original state. Similar techniques could be applied for fault tolerance, hot sparing, on-orbit testing, or reconfiguration.

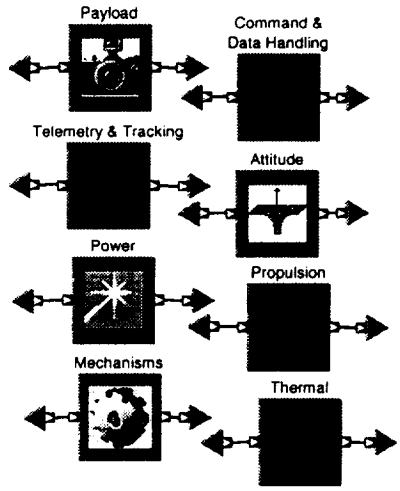

Figure 8: A generic spacecraft architecture.

4 Future Research

One outstanding problem is porting the weave runtime infrastructure to a generic micromachine hardware environment. This must be a cooperative venture between micromachinists and computer scientists. The history of processor architectures is illuminating in this respect. For many years processor architectures were designed by electrical engineers and

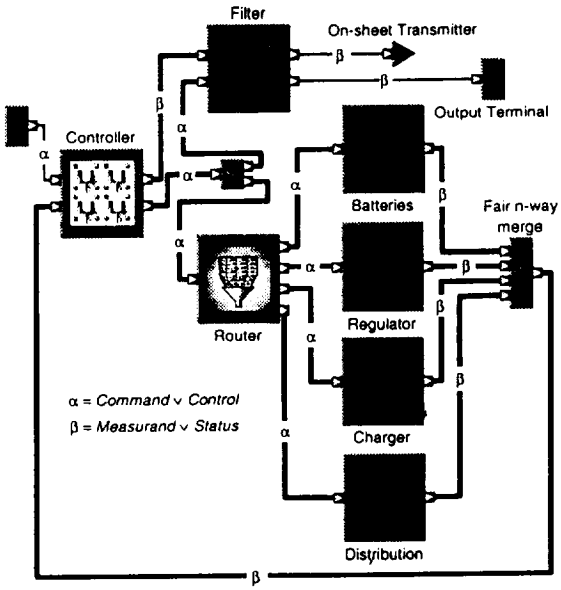

Figure 9: A power system framework.

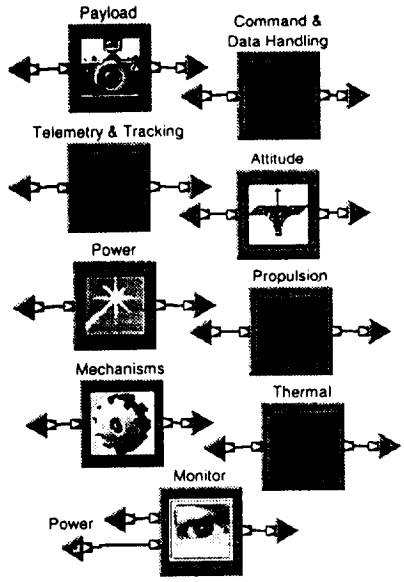

Figure 10: A spacecraft architecture with additional monitoring.

device architects as if software didn't exist and processors were useful in their own right wholly independent of software. We now understand that view to be misguided and the rise in popularity of RISC architectures is the outcome of a joint venture to arrive at architectures that were designed from the outset to support complex software.

The weave execution infrastructure is non-trivial and makes numerous demands on the underlying operating system. It also has strong implications for the bus structure that is used as the communication path among microdevices. The best of all possible worlds would be to design micromachines and their digital controllers with weaves in mind from the very beginning including hooks for atomic weave components (that is weave device drivers) where the bottom of the weave ecology is connected to the particulars of the custom devices provided by the micro-environment.

If one accepts the argument that the missing piece in the weaves-to-micromachines picture is the software/hardware glue, then we must build generic weave components that can talk at the hardware level to a micromachine device, and at the same time, encourage micromachine hardware designers to settle on a generic hardware interface structure that is, at worst, not hostile to weaves. Once this is in place, it should be relatively easy to demonstrate the viability and power of weaves for the software structures of assemblages of micromachines. One attractive, low risk possibility is to simulate a micromachine assemblage using chip-level components and construct a prototype weave infrastructure for such a simulation.

References

[1] P. T. Cox, F. R. Giles, and T. Pietrzykowski. Prograph: a step forward liberating programming from textual conditioning. In *Proceedings of the 1989 IEEE Workshop on Visual Languages*. IEEE Computer Society Press, 1989.

[2] R. A. Earnshaw and N. Wiseman. *An Introductory Guide To Scientific Visualization*, chapter 8. Springer-Verlag, 1992.

[3] Michael Gorlick and Alex Quilici. Visual programming in the large versus visual programming in the small. In *Proceedings of the 1994 IEEE Symposium on Visual Languages*, St. Louis, Missouri, October 1994. IEEE Computer Society Press.

[4] Michael M. Gorlick. Cricket: A domain-based message bus for tool integration. In *Proceedings of the 2nd Irvine Software Symposium*, Department of Information and Computer Science, University of California, Irvine, Irvine, CA 92717, March 1992. Irvine Research Unit in Software.

[5] Michael M. Gorlick and Rami R. Razouk. Using weaves for software construction and analysis. In *Proceedings of the 13th International Conference on Software Engineering*, pages 23–34, Austin, Texas, May 1991. IEEE Computer Society Press.

[6] M. A. Najork and E. Golin. Enhancing Show-and-Tell with a polymorphic type system and higher-order functions. In *Proceedings of the 1990 IEEE Workshop on Visual Languages*, pages 215–220. IEEE Computer Society Press, 1990.