Interstacks
End-User “Scripting” for Hardware

Peter Lucas

Founder, Principal, Board Chair
MAYA Design, Inc.

Published August 1999

ABSTRACT

More and more consumer and commercial products contain at least one microprocessor. While efforts to develop “device bus“ standards to integrate the automation of these devices have increased the potential for large-scale interoperability, this potential will remain largely unfulfilled for some time. Interstacks is a modular hardware system that empowers even non-technical users to integrate bits of specialized hardware in order to automate and control the flow of information among electronic products. It reinterprets the notions of component architecture and end-user scripting in the domain of hardware devices.

Keywords

Modular, End-user Programming, Home Automation, Process Control, Scripting

A MAYA® paper

MAYA Design, Inc.
2730 Sidney Street
Pittsburgh, PA 15203

T: 412-488-2900
F: 412-488-2940
maya@maya.com
www.maya.com

INTRODUCTION

An increasingly high percentage of consumer and commercial products contain at least some computing capabilities. Although the use of embedded processors is motivated primarily by lower costs and improved functionality, a side effect of this trend is that many devices in the environment have at least some ability to be controlled externally (e.g., infrared remote controllers for televisions), thus making automation and integration across devices possible.

Accelerating this trend has been the development of a large number of “device bus” standards aimed at integrated automation among separately manufactured devices. There are literally dozens of such proposals, involving such diverse media as infrared, RF, power line signaling, fiber optics, and dedicated copper connections.

The result has been a great increase in the potential for cross-device integration. The large number of competing communications options, however, has led to a tower of babble that threatens to leave the potential for large-scale interoperability unfulfilled for a long time to come.

In the software industry, a similar situation led to the introduction of so-called “component architectures” as a way to support interoperability among technologies in the face of heterogeneous implementation strategies. Standard software interfaces and message-passing schemes allow developers with little understanding of the internals of each module to assemble diverse components into end applications. Moreover, component architectures go hand-in- hand with end-user scripting environments, such as HyperCard and Visual Basic, which are the “glue” for connecting components together in the field by so-called “power users” who have the conceptual knowledge, but not the detailed technical skills to do low-level programming.

INTERSTACKS

The Interstacks project began with the question, “How could the notions of component architectures and enduser scripting be reinterpreted in the domain of hardware devices?” Just as HyperCard lets power users string together components without having to worry about type coercion and hash tables, how could we empower such users to string together bits of specialized hardware without having to worry about fanouts and despiking capacitors?

Interstacks is composed of a series of modules that can be plugged together in any combination and number. Plugged directly into each other, rather than into a common backplane, the modules form a bus that can pass data and multimedia information among the modules in arbitrary combinations. Such a collection of interconnected modules is known as a “stack.”

Interstacks extends the notion of traditional control and instrument buses (e.g., the IEEE 488.2 GPIB standard [1]) by adding extensible multimedia support, and by associating the uniform bus protocol with a standardized self-stacking package that is both inexpensive to manufacture and easy to use. Modules can be used in many different configurations, ranging from only one or two sitting on a desktop or attached to a controlling device, up to a large number mounted on a wall or an equipment rack. The logical relationships among modules are determined via a separate direct-manipulation visual programming application that exposes the inputs and outputs of each module to the user. This interface supports module interconnection via the dragging of virtual “wires” from module to module.

Figure 1: Interstacks Modules

Modules intercommunicate via a simple serial token bus optimized for low-bandwidth control applications and very low cost. There are also provisions for optional specialized buses such as multichannel audio and video, Universal Serial Bus connections, etc.

From a functional perspective, modules can be divided into several categories:

Master Module. This module is the “stack manager,” coordinating the intercommunication of all the other modules in a stack and optionally providing a communications channel to an external control computer.

I/O Modules. These modules implement the physical and logical layers necessary to support communications with external

©2009 MAYA Design, Inc.

devices. Examples might include an RS-232 module, an X-10 lighting-control module, an IR remote control module, and a telephone interface.

Control Modules. These are “programmable” modules that send data to other modules, receive data from them, and perform local computations. Examples might include a simple programmable controller, or a module implementing the Java Virtual Machine.

Memory Modules. These can be used as auxiliary storage for other modules in a stack, or can come preloaded with useful information such as a list of telephone numbers or a dictionary. In the Interstacks architecture, only memory modules are permitted to persistently store state.

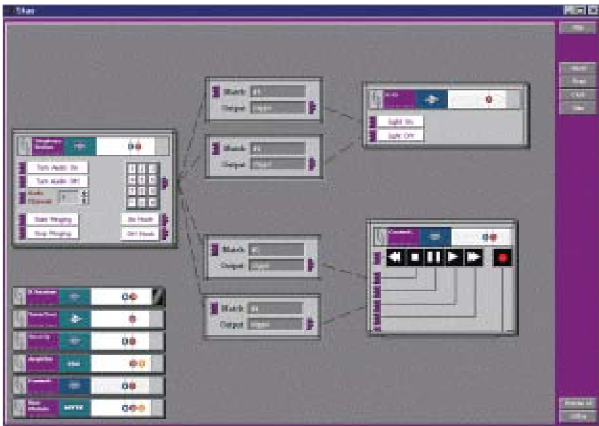
Human-Interface Modules. These accept messages containing information to be presented to users and/or generate messages in response to user actions. Examples might include a keyboard module, a touch screen module, and a VGA flat-panel display module.

SCRIPTING INTERFACE

The “scripting” interface to Interstacks is implemented as a PC application that implements a highly intuitive direct manipulation environment that combines a visual programming scheme with a simple, declarative scripting language. This visual programming approach is in the tradition of such visual scripting approaches as LabVIEW [3] and Java Studio [2]. We can plug together a set of Interstacks modules, including a base module connected to a PC, and use this development environment to “script” complex sets of behaviors among different technologies (e.g., home security, home theater, and telephony).

Figure 2: Scripting Interface for Interstacks

As soon as we apply power, the PC will present a visual representation of the stack in the form of a set of “proxy modules”—directly manipulable screen representations of each module, including its logical interface (i.e., the kinds of messages that the module can send or receive).



We can establish message paths from module to module simply by dragging virtual “wires” among them. For example, we can “wire” an X-10 lighting module to an IR remote control module that will permit us to control the light with the TV remote control. Arithmetic and logical operations, future event scheduling, and other semantically complex processes can be implemented by using simple “script” modules, which are logically identical to proxy modules, but are implemented in software and have no physical module behind them.

ACKNOWLEDGEMENTS

Portions of this work were funded by DARPA contract #F30602-97-C-0262.

The ideas presented here have emerged from long collaboration with my colleagues at MAYA, and with our clients. I must particularly thank Jeff Senn, whose contributions to these ideas cannot be fully separated from my own, and Susan Salis, for whose editorial assistance my readers should feel grateful.

REFERENCES

1. IEEE 488.2-1992. IEEE Standard Digital Interface for Programmable Instrumentation. Institute of Electrical and Electronic Engineers, 1992.
2. Java Studio. Sun Microsystems Inc. http://www.sun.com/studio/.
3. Johnson, Gary W. LabVIEW Graphical Programming. McGraw-Hill, New York, 1997.

©2009 MAYA Design, Inc.