Scripting a Radically-Distributed World

Recursively-nested Visual Dataflow Programming for Pervasive Computing

Peter Lucas, Jeff Senn, Magesh Balasubramana, Stuart Roth, Steve Spencer
MAYA Design, Inc.
Pittsburgh, PA United States

***Abstract*—The built world increasingly comprises objects with at least some information-processing capability, and these objects are rapidly acquiring the ability to intercommunicate. The problem of how to orchestrate huge numbers of independently designed devices into coherent environments is becoming acute. A pure dataflow programming paradigm is a natural choice for addressing this challenge. We present an information centric, recursive, hybrid visual programming environment that addresses many of the usability problems that have plagued previous attempts at large-scale visual dataflow programming environments.**

Keywords—visual programming; dataflow; information centric interfaces; pervasive computing; ubiquitous computing

I. INTRODUCTION

The long-predicted age of pervasive computing is nearly upon us. The fundamental new computational capabilities that have so dramatically transformed modern living have taken two distinct forms. The more obvious and mature of the two is a direct result of Moore's famous law. There have by now been so many of the predicted doublings of transistor density and increases in clock speed that the raw computational power now at our disposal practically defies comprehension. The net result of this bounty, although vast in its magnitude, has been essentially sustaining in nature. With the notable exception of the move from wired to mobile end-user devices, the basic character of personal computing has changed little. We may choose to call it "cloud computing" instead of "client-server computing", but the basic pattern of powerful general-purpose personal computers pushing and pulling data from even more powerful centralized services remains in place.

A second, less mature but ultimately more disruptive kind of pervasive computation is rapidly evolving. Simple economic considerations suffice to ensure that almost all newly designed products of any significant complexity will contain at least modest computational capabilities. This in itself is not particularly disruptive. However, if one adds to the mix the ability of these devices to intercommunicate, the nature of the coming revolution begins to come into focus. The technical and usability challenges implicated by this state of affairs are enormous. The tools, design patterns, and expertise that will be required to guide the unfolding of the pervasive computing scenario are fundamentally different from those that have matured in the age of "big computing". This scenario will be characterized by much more local (i.e., peer-to-peer) flows of data [1]; ad-hoc rather than carefully engineered device configurations; and a vastly heterogeneous mix of devices, both with respect to architecture, manufacturer, and level of computational power. A typical household already contains hundreds of CPUs. The number will soon be thousands, and it will someday be millions. The management of networks of this size is far from a routine activity, even for trained professionals. Their management by ordinary users and blue-collar technicians represents uncharted territory. An architecture adequate to meet this challenge must satisfy three prime criteria: scalability, tractability, and comprehensiveness [1].

II. DISTRIBUTED DATAFLOWS

A. Infotrons

In this demonstration we introduce the Information Device Architecture (IDA), an architecture designed from the ground up to address the challenges of guiding the emergence of a vastly distributed, pervasive computing environment. The core of the model is an abstraction known as the "*Infotron*". Infotrons are recursively defined, highly encapsulated computational nodes that are related to each other via a classical strict dataflow model. Infotrons are stateful devices, that share certain properties with the Actors formalism [2]. They accept *messages* on "input terminals" and emit messages on "output terminals". There is no other way to affect the state of an extant infotron. Infotrons are related to each other via *channels*, which are stateless, directed paths along which messages may travel. These three primitives (Infotrons, Messages, and Channels) form the formal basis for the model, with all higher-level constructs properly layered upon these constructs.

It is important to note that IDA is not merely (or even primarily) a programming environment. Rather, it is intended as a general architecture for the design and implementation of pervasive computing environments. Thus, for example, infotrons are an abstraction. The architecture makes no assumption about whether they are realized as hardware or software devices. Indeed, it is an explicit goal of the architecture to guarantee the fungibility of equivalent implementations of an abstract infotron, not only across software platforms (e.g., Python, C, Java), but also across the

[1] The alternative scenario in which, say, home light-switches are mediated via remote "cloud" services is rejected without comment.

software/hardware boundary. More specifically, it is guaranteed that a system of proper infotrons implemented as threads of execution within a single software platform can at a future date be distributed to an environment in which each infotron is given its own processor and memory, assuming only that message-passing channels can be established among the nodes.

B. *Recursive Decomposition*

Infotrons may be *atomic* or *composed*. Atomic infotrons are merely enumerations of terminals with some algorithmically defined specification of the relationship among the messages it may receive and those it may emit. In practice, these specifications are defined using traditional programming languages. In our implementation, we typically employ Stackless Python [3] for this purpose, although we have also demonstrated a Java-based implementation. In our prototypes, we usually implement hardware infotrons using the Interstacks® modular component architecture [4].

Composed infotrons are infotrons defined in terms of other infotrons (atomic or composed) plus terminals (which are themselves formally infotrons), formed into a directed graph via channel specifications. In this way, IDA is a recursively defined system which permits complex networks of infotrons to be factored into tractable units, forming the basis of a "marketplace" of components, with stable APIs but fungible implementations. This "marketplace of components" approach is offered as an economically viable complement to the open-source approach to systems development.

Composed infotrons are defined in such a way that they have no semantic effect on a realized dataflow. That is, dataflows containing composed infotrons can be "flattened" into equivalent graphs containing only atomic infotrons. Indeed, our reference implementation contains a "dataflow compiler" that performs such a flattening during the compilation process. Although this model is primarily hierarchal, it is important to note that infotrons do not form a strict tree structure. This is because terminals of infotrons can appear in the definition of more than one other infotron. In this way, IDA represents a "nearly decomposable system" [5], providing essential flexibility without unduly sacrificing tractability.

C. *Usability Considerations*

Previous applications of the dataflow model to the development of large-scale systems have met with mixed results. Perhaps the most notable such attempts are the Prograph environment [6] and the LabView instrumentation environment [7]. To the extent such systems represent "pure" dataflow systems, they typically face severe scalability problems, with the management of complex dataflows presenting serious challenges. To address such issues, Prograph evolved as a hybrid of dataflow and object-oriented programming techniques, thus adding significantly to the complexity of the system. LabView—a very successful and long-lived tool—contains a number of notable compromises to the "pure" dataflow model, most notably the ability to define global state and centralized queues. These compromises have proven successful in improving the usability of the system. Unfortunately, such compromises to encapsulation are in fundamental conflict with IDAs goal of guaranteed decomposability into independent systems.

IDA addresses the need for shared state by use of a single, explicit mechanism for externalizing data. This system is known as the Visage Information Architecture (VIA), and is based on a simple data abstraction known as a u-form and a novel database architecture called the VIA Repository [8]. Describing VIA is beyond the scope of this demonstration, except to say that it depends upon an explicit, introspective information architecture and replicated data objects as the sole mechanism for externalizing data.

III. BLUEPRINTS AND VISUALIZATION

The IDA architecture defines a standard format for the serialization of abstract descriptions of infotrons. This format, known as the Visage Standard Blueprint Format (VSBF) represents infotron descriptions in the form of trees of u-forms stored in a VIA Repository. This format was specifically designed to be amenable to visualization, manipulation, and sharing via the Visage information visualization environment [9]. The Visage environment provides an information-centric medium for constructing, sharing, and composing blueprints describing IDA dataflows. The demonstration will illustrate how Visage's aggressive use of direct manipulation combined with its nested containment model maps directly onto the IDA abstraction, providing the basis for a highly usable and scalable environment for managing dataflows of arbitrary complexity. We will demonstrate the intermixing of hardware and software implementations of infotrons within hybrid dataflows. The practicality of the model will be illustrated using the fact that the current version of the Visage system is entirely implemented using the IDA/VIA model.

REFERENCES

[1] P. Lucas. "The Trillion-Node Network." MAYA Technical Report, #MTR-00001, 1999.

[2] G. Agha. "Actors: A Model of Concurrent Computation in Distributed Systems." MIT Artificial Intelligence Laboratory Technical Report, #844, 1985.

[3] "Stackless Python." Internet: http://www.stackless.com/, [June 17, 2011].

[4] P. Lucas. "Interstacks: End-user 'scripting' for hardware," presented at the Association of Computing Machinery CHI 99 conference, Pittsburgh, PA, May, 1999.

[5] H. Simon. "The architecture of complexity." *Proceedings of the American Philosophical Society*, vol. 106, no. 6, pp. 467–482, Dec. 12, 1962.

[6] S. Matwin and T. Pietrzykowski. "Prograph: A Preliminary Report." *Computer Languages*, vol. 10, issue 2, pp. 91–126, 1985.

[7] J. Travis and J. Kring. *LabVIEW for Everyone: Graphical Programming Made Easy and Fun*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, August 2006.

[8] P. Lucas and J. Senn. "Toward the Universal Database: U-forms and the VIA Repository." MAYA Technical Report, #MAYA–02011, 2002.

[9] S. Roth, P. Lucas, J. Senn, C. Gomberg, M. Burks, P. Stroffolino, J. Kolojejchick, and C. Dunmire. "Visage: A User Interface Environment for Exploring Information," in *Proceedings of IEEE Symposium on Information Visualization '96*, 1996, pp. 3–12.