10-2001

SOLAR: Towards a Flexible and Scalable Data-Fusion Infrastructure for Ubiquitous Computing

Guanling Chen
Dartmouth College

David Kotz
Dartmouth College

SOLAR: Towards a Flexible and Scalable Data-Fusion Infrastructure for Ubiquitous Computing

Guanling Chen and David Kotz
Dartmouth College
Hanover, NH, USA 03755

{glchen, dfk}@cs.dartmouth.edu

Abstract

As we embed more computers into our daily environment, ubiquitous computing promises to make them less noticeable and to avoid information overload. We see, however, few ubiquitous applications that are able to adapt to the dynamics of user, physical, and computational context. The challenge is to allow applications flexible access to these sources, and yet scale to thousands of devices and sensors. In this paper we introduce our proposed infrastructure, SOLAR. In SOLAR, information *sources* produce *events*. Applications may *subscribe* to interesting sources directly, or they may instantiate and subscribe to a tree of *operators* that filter, transform, merge and aggregate events. Applications use a subscription language to describe the tree, based on event streams registered in a context-sensitive naming hierarchy. SOLAR is flexible: modular operators can be composed to produce new event streams. SOLAR is scalable: it distributes operators across hosts called *Planets*, and it re-uses common subgraphs in the operator network.

1 Introduction

As wireless computing devices become more common, their users are exposed to a tremendous amount of information that comes from an array of diverse sources, such as location sensors, weather or traffic sensors, network monitors, information appliances, the status of computational or human services, and so forth. One important goal of ubiquitous computing, however, is to help the user overcome information overload and concentrate on the current task [7]. To do so, ubiquitous applications must be aware of the situation in which they are running. They must obtain and analyze the data about their context, and adjust their behavior without unnecessarily distracting the user. This task is challenging because applications face heterogenous data types, different communication protocols, unreliable wireless links with limited bandwidth, low data quality due to error-prone sensors affected by environmental noises, and many other issues.

Asking each application to work directly from information sources overwhelms programmers with details and complexity. It leads to ad-hoc solutions that will not scale with N^2 pipes between the sources and applications. The programmers need a toolkit to simplify their task and the toolkit needs an infrastructure to deliver context data, an infrastructure that can scale by sharing data streams.

Both Gryphon [1] and Siena [2] are content-based publish/subscribe systems and provide large-scale event-routing services. While they can provide some event-fusion functions, it is unclear how these functions are deployed and whether it is possible to share them across applications. Since the ubiquitous adaptive applications typically gain the high-level "awareness" by filtering, transforming, merging, and aggregating low-level sensor data, what we really need is a fusion-centric (instead of routing-centric) data dissemination infrastructure that provides sharing of both data streams and fusion functionalities. Building such a data-fusion infrastructure for a ubiquitous mobile environment has to face many technical challenges, which are summarized by Cohen et al. [4].

We propose a system infrastructure, SOLAR, to meet the challenges. Our goal is to allow applications to define their own operations, to describe flexible compositions of operations, and to support many such applications with scalable performance. SOLAR provides flexibility by allowing applications to define and interconnect *operator* objects. SOLAR provides scalability by distributing these operators

This research has been supported by DARPA contract F30602-98-2-0107, by DoD MURI contract F49620-97-1-03821, by Microsoft Research, and by the Cisco Systems University Research Program.

across hosts in the network and by sharing identical data streams across users and applications.

This paper provides a brief overview of the Solar infrastructure. We necessarily skip many of the details here because of the space limitation. For more information see our Technical Report [3].

2 Operator graph

Solar treats sensors of ubiquitous data as *information sources*, whether they sense physical properties such as location, or computational properties such as network bandwidth. Information sources produce their data as *events*, implemented in Solar as objects. The sequence of events produced is an *event stream*, which is inherently unidirectional. An event *publisher* produces an event stream, and an event *subscriber* consumes an event stream.

An *operator* is an object that subscribes to and processes one or more input event streams, and publishes another event stream. Since the inputs and output of an operator are all event streams, the operators can be connected recursively to form a directed acyclic graph, an event-flow graph that we call the *operator graph.*

Our operator graph consists of three kinds of nodes: sources, operators, and applications. The *sources* have no subscriptions. They are wrappers for information sensors. *Operators* are deterministic functions of their input events. They only publish an event when they receive an input event. *Applications* are sinks of the graph. They subscribe to one or more event streams and react to incoming events (and possibly other stimuli, such as interactions with the user).

There are four common categories of operators. A *filter* outputs a subset of its input events. A *transformer* inputs events of one type and outputs events of the same or another type. (The types may be the same if the transformer only changes some attribute values.) The *merger* simply outputs every event it receives. While mergers are not strictly necessary, since any of the merger's subscribers could directly subscribe to the same inputs, a merger aids re-use of event streams. An *aggregator* outputs an arbitrary type event stream based on the events in one or more input event streams.

Example. We show a simple example operator graph in Figure 1. Suppose in a typical office environment with location system (such as Active Badge) installed, each room has a location sensor that periodically reports the badge number it detects in that

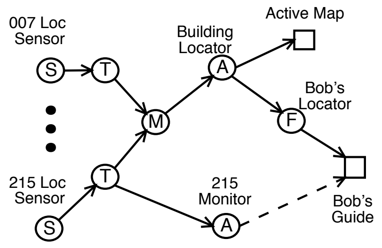

Figure 1: An example operator graph.

room. These events are transformed to appropriate names of the objects associated with the badge before merged into a single event stream. The aggregator labeled as "Building Locator" takes this stream and produces only location-change events about the objects. Similarly, the "215 Monitor" reports whenever an object leaves or enters room 215. The Active Map running on a public wall-mounted screen can then display the current location of tracked object by subscribing to the "Building Locator" aggregator. The aggregator's output can also be filtered to report only Bob's movement. The personal Guide application on Bob's PDA displays the descriptions about the room and the list of current resident objects and associated links, following which Bob can find more information such as the calendar of a colleague and the manual of a scanner. The dashed arrow represents a context-sensitive subscription (assuming Bob is currently in room 215), which will be discussed in Section 4.

In this example we show how the operator graph achieves flexibility by allowing dynamic composition of operators and achieves scalability by re-using event streams. These applications adapt to high-level context that is derived from the same sources, simply reusing the operators without having to process raw data from sensors on their own. The redundant and irrelevant events have been filtered out before reaching these applications, making it possible to support a larger number of applications without saturating network edge links.

In summary, once the data-fusion functionalities (represented by operators) are isolated from the applications, Solar can deploy them across the network and promote re-use of these operators by various applications. This approach achieves several advantages: 1) It provides a fine-grain application-partitioning method for flexible adaptation to various capabilities of the mobile devices and available network connectivity; 2) Solar encourages the use and sharing of filter operators, and the deployment of those filters close to the information sources, which can dramatically reduce bandwidth used on slow network links; 3) Re-use of an operator instance in

Solar is to re-use all the event streams and operators in the subgraph rooted by that operator, allowing much easier composition than manually specifying a list of components the data should go through from source to application (such as the "after list" in Active Names [6]); and 4) the operator graph can dynamically grow as an application customizes the information flow by adding its own fusion operators and re-using many others, avoiding the limitation (the need of a large predefined service repository) imposed by automatic path construction [5].

3 Context-sensitive naming

Although we can build operator graphs from the connections described by a subscribing application, and Solar has a small language for that purpose, it is frequently useful to name event publishers so that their streams are easily usable by many subscribers. By naming a publisher, applications can subscribe to its event stream without needing to describe that stream from first principles (sources). It is common to construct named mergers, for example, to make pre-defined combinations of event streams available to many applications.

A flat name space does not scale. Furthermore, given the dynamic nature of an application's context in ubiquitous computing environments, Solar's name bindings change with the changing context. So Solar's name space is organized as a tree of labelled nodes, with the addition of cross edges (see an example in [3]). The *name* for a node is the sequence of labels encountered on any path from the root to the node, separated by slashes. There are several types of nodes. Some of the leaves are *alias nodes* that are representatives of another node. By analogy to Unix, we call these cross edges "soft links". Some leaves may be *service nodes*, which represent available services. Other leaves are the event publishers we discussed in the operator graph.

Internal nodes are *directories* that refer to several child nodes. *Static* directory nodes contain a list of children, a list updated only by explicit requests. *Dynamic* directory nodes generate a list of children dynamically, from internal state that depends on contextual information obtained from subscriptions. Typically the children of dynamic directories are nodes already located elsewhere in the name space.

Soft links are dynamically generated by a directory or alias node when needed. These directory and alias nodes must, therefore, be operators with appropriate subscriptions and sufficient state to be able to generate the appropriate list of children when asked. For example, the node [/places/2F/215/people] lists Bob as child and [/people/profs/Bob/location] refers to room 215. These links are automatically updated as the user moves across geographical spaces. Since both operators derive the location information from the same source [/places/2F/215/location-sensor], these two views will remain consistent.

Nodes in the name space are also publishers of changes to name bindings. More precisely, directory nodes are publishers that announce additions or deletions of their children by publishing events. (Static directory nodes are sources and dynamic directory nodes are operators.) Alias nodes are publishers that announce changes in their bindings. Interested applications can subscribe to these sources to detect changes in the name space. So if Alice wants to track the location of her advisor Bob, her application subscribes to the operator named as [/people/profs/Bob/location].

Although Solar's concept of "operator graph" is independent of its naming scheme, we believe the context sensitive namespace is ideal for adaptive ubiquitous applications. The namespace re-uses the operator abstraction and connects the physical location with entities inside, allowing applications to use one persistent name to access location-dependent information sources while Solar automatically selects and delivers appropriate event streams as user moves around. This approach removes much of the complexity of location tracking from applications.

4 System architecture

We discuss Solar's architecture in this section. At the center of any Solar system is a *Star*, which keeps a reference to the root of naming tree, maintains the operator graph, and services requests for new subscriptions. When the Star receives a new subscription-tree description, it parses the description, checks the name space, and matches the subscription tree against its internal data structure representing the operator graph. When it decides to deploy an operator, it instantiates the operator's object on one of many *Planets*. Each Planet is an execution platform for Solar sources and operators. (In our implementation, operators are Java objects and Planets are Java virtual machines.) Applications run outside the Solar system, on any platform. They use a small Solar library that allows them to send requests to the Star, and to manage their subscriptions, over standard network protocols.

Planets play a key role in the subscriptions of resident operators. When deploying new subscriptions, the Star tells the Planets to record a subscription from one of its operators to an operator in another Planet. In our implementation there is at most one network (TCP/IP) connection between any two Planets, regardless of the number of operators on or subscriptions between the two Planets.

Another critical advantage of our approach is that the Planet supports subscription requests that involve context-sensitive names (CSNs). These subscription requests are mapped to subscriptions, which need to be changed when the CSN binding changes. Consider an operator X that records the name of every person Bob meets. The operator requests subscription to the CSN [/people/profs/Bob/location/people], currently bound to an operator P. X's Planet subscribes to the name [/people/profs/Bob/location]. The Planet receives the current binding and subscribes X to P. When Bob moves, suppose the binding changes to operator Q. X's Planet contacts P's Planet to remove X from P's data structure, and contacts Q's Planet to add X to the Q's data structure. All the work is done by planets and the namespace operators; P, Q, and X are never involved.

These and other aspects of the Solar architecture are key to its scalability and flexibility. Furthermore, although the operators must execute inside a Solar Planet, applications may run on any platform. Thus, legacy or COTS applications may easily be integrated with a ubiquitous-computing environment built on Solar technology.

5 Future work

Our focus on Solar so far has been to develop a useful model for flexible, scalable access of contextual information. There are three critical directions in which we must refine our design: scalability, reliability, and security. We plan to

- distribute the responsibility of the Star to avoid a central point of failure and to avoid any possible performance bottleneck in systems where new subscriptions occur frequently,
- extend the name space to allow federation of many name spaces,
- address reliability in the face of failure of a Planet or a network connection,
- develop a mechanism for flow control between publishers and subscribers,
- consider security and privacy (Planets need to execute untrusted operator code securely, Planets must limit the resource usage, and Solar must limit subscription to event streams according to an access policy), and
- experiment with the use of Solar in several context-sensitive mobile applications to determine the value of the abstraction and the performance of the system.

References

[1] G. Banavar, M. Kaplan, K. Shaw, R. E. Strom, D. C. Sturman, and W. Tao. Information flow based event distribution middleware. In *Proceedings of the Middleware Workshop at the 19th IEEE International Conference on Distributed Computing Systems*, Austin, Texas, May 1999. IEEE Computer Society Press.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an Internet-scale event notification service. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, Portland OR, USA, July 2000.

[3] G. Chen and D. Kotz. Supporting adaptive ubiquitous applications with the SOLAR system. Technical Report TR2001-397, Dept. of Computer Science, Dartmouth College, May 2001.

[4] N. H. Cohen, A. Purakayastha, J. Turek, L. Wong, and D. Yeh. Challenges in flexible aggregation of pervasive data. Technical Report RC21942, IBM Research Division, Thomas J. Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, January 2001.

[5] E. Kiciman and A. Fox. Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In *Proceedings of Second International Symposium on Handheld and Ubiquitous Computing*, pages 211–226, Bristol, UK, September 2000. Springer Verlag.

[6] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active Names: Flexible location and transport of wide-area resources. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, October 1999. USENIX.

[7] M. Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991.