Title

Open implementation approach to Internet-scale context awareness

Author

Boyer, Robert T.

Publication Date

2005

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Open Implementation Approach to Internet-Scale Context Awareness

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in

Computer Science

by

Robert T. Boyer

Committee in charge:

Professor William G. Griswold, Chair
Professor Ingolf Krueger
Professor Mohan Trivedi
Professor Andre Van Der Hoak
Professor Geoffrey M. Voelker

2005

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

The dissertation of Robert T. Boyer is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2005

Table of Contents

List of Figures

List of Tables

Acknowledgements

"It takes a village to raise a child." [African proverb]

An unusual double entendre, but appropriate. As I toil at this task, I thank the village raising my son, for whom we pray every night, and I thank the village that continues to raise me.

I pay my respects to my friends who have died too young: Bob Simms, Cynthia Mott, Marcus Gerstler, Steve Mulkern, and Richard Staeffler.

I honor all the men and women of our armed services who defend our freedoms and remember all the victims of the twin-towers disaster of 9/11.

Above all, I especially thank Anne-marie & Ryan for enduring this.

Bob

This dissertation reprints material as it appears in *Fulcrum – An Open Implementation Approach to Internet-Scale Context-Aware Publish / Subscribe*, Boyer, Robert T., Griswold, William G., Proceedings 38th Hawaii International Conference on System Sciences, 2005. The dissertation author was the primary investigator and author of this paper.

Vita

Robert Thomas Boyer was born in Canoga Park, California and came to San Diego at the age of two, where he has resided since. He began working full time in 1982 to pay his way through college. In 1986, he received a Bachelor of Arts in Computer Science from the University of California at San Diego.

After an internship with a startup company in 1984, the early phase of his industry career began in 1985 when he started performing static software analysis on real-time mission critical software systems and hardware / software interfaces. These included the digital flight control system for the F-16, the Phalanx Close-In Weapon System, the flight control system for the JAS Gripen-39, the Titan Centaur digital control unit, and the Titan Centaur inertial navigation unit. For these analyses he developed the tools, performed the analyses, and led the teams.

The second phase of his career began in 1992 and was marked by a transition into C^4ISR (command, control, communications, computers, intelligence, surveillance, and reconnaissance). The focus has been real-time information distribution and management. This work included development of automated target recognition algorithms from 3-D laser radar data, air courses of action assessment models, multi-source multi-environment correlation algorithms, and command and control communications (and data base replication) for the tactical data links as part of the global command and control system and as ported into the B2. These tasks have ranged from individual research and development, to hiring and leading teams of twenty people, to leading integration designs across four companies.

Prior to the dot com boom, he also consulted for, or was a principle of, several startups on the side. Tasks included developing tools and processes for document imaging services, development of the Tae Kwon Do Tutor, and development of the High School Yearbook on CD-ROM.

While continuing full-time employment, in 1999 he returned to the University of California at San Diego to refresh his formal education. In 2001 he earned his Masters Degree in Computer Science with an emphasis in Artificial Intelligence. As part of a class project, he and his teammates created the original Active Campus project. User location was determined by triangulation, based on 802.11b signal strengths, and colleagues and points of interest were dynamically plotted on a campus map displayed on a PDA. After slowly divesting himself of leadership responsibilities he was able taper off employment hours in favor of education and research hours until, in 2004, he was finally able to devote his full attention to completing his Ph.D.

Publications

- R. T. Boyer and W. G. Griswold, ``Fulcrum - An Open-Implementation Approach to Internet-Scale Context-Aware Publish / Subscribe'', *Software Technology Track, Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, January 2005.
  Best Paper Award

- W. G. Griswold, P. Shanahan, S. W. Brown, R. Boyer, M. Ratto, R. B. Shapiro, and T. M. Truong, ``ActiveCampus - Experiments in Community-Oriented Ubiquitous Computing'', *IEEE Computer*, Vol. 37, No. 10., pp. 73-81, October 2004.

- W. G. Griswold, R. Boyer, S. W. Brown, and T. M. Truong, ``A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure'', *2003 International Conference on Software Engineering (ICSE 2003)*, May 2003.

- W. G. Griswold, R. Boyer, S. W. Brown, T. M. Truong, E. Bhasker, G. R. Jay, and R. B. Shapiro, ``Using Mobile Technology to Create Opportunitistic Interactions on a University Campus'', *UbiComp 2002 Workshop on Supporting Spontaneous Interaction in Ubiquitous Computing Settings*, Technical Report CS2002-0724, Computer Science and Engineering, UC San Diego, September 2002.

- W. G. Griswold, R. Boyer, S. W. Brown, T. M. Truong, E. Bhasker, G. R. Jay, and R. B. Shapiro, ``ActiveCampus - Sustaining Educational Communities through Mobile Technology'', Technical Report CS2002-0714, Computer Science and Engineering, UC San Diego, July 2002.

Presentations

- R. T. Boyer and W. G. Griswold, ``Fulcrum - An Open-Implementation Approach to Internet-Scale Context-Aware Publish / Subscribe'', *Software Technology Track, Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, January 2005.
  Best Paper Award

Poster Sessions

- R. T. Boyer and W. G. Griswold, ``Fulcrum - An Open-Implementation Approach to Internet-Scale Context-Aware Publish / Subscribe'', *Foundations of Software Engineering*, October 31-November 5, 2004.

ABSTRACT OF THE DISSERTATION

Open Implementation Approach to Internet-Scale Context Awareness

by

Robert T. Boyer

Doctor of Philosophy in Computer Science

University of California, San Diego, 2005

Professor William G. Griswold, Chair

Proliferation of hardware and software sensors and our desire to determine relationships between the near-real-time data from multiple publishers motivates our introduction of Internet-scale context-awareness (ISCA). Content-based publish / subscribe (CBPS) seems the most natural substrate for ISCA because it provides the right separation of concerns, efficient event distribution, extensibility, and scalability.

However, our evolving information environment is different from that for which CBPS was designed. Attempting to use the black-box style transparency afforded by CBPS precludes efficiently detecting data relationships for publication as context-aware events and leads to information glut and device saturation. We overcome these problems

by recognizing that any component-based system is an ecology for which we can achieve global efficiencies by providing top-down and bottom-up context and collaboration.

We extend CBPS with an open implementation approach to enable subscribers to inject domain-specific knowledge into the network in the form of first-class publish / subscribe agents. Agents are distributed algorithms that observe and transform data, dynamically manage bounding region filters, and exchange data only on an as-needed basis to eliminate useless event traffic at the sensor-edges of the network. Filtering at the network edge reduces bottlenecks in the network core to increase the scalability of the system. Content-based routing mechanisms are leveraged to allow the user to control where code is deployed, to develop complex relationship hierarchies, and to construct one-to-one conversations by leveraging existing network knowledge without flooding the network with either advertisements or subscriptions. We are programming the network. We add dynamic contextual message filtering and distributed memoization to minimize re-computation at downstream nodes.

Combining open implementation, distributed processing, content-addressability, and distributed memoization satisfies the required increases in expressiveness, efficiency, and scalability necessary to achieve our Internet-scale context-awareness vision. Our algorithm detecting the proximity of mobile buddies reduced event traffic from $O(|\text{events}|)$ to an expectation of about $\ln(|\text{movement}|)$ event-hops. Complex traffic-route monitoring used eight times fewer events than basic CBPS and reduced aggregation enhanced CBPS load imbalances by distributing relationship computations over the event-entry edge-brokers. Our algorithms are scalable with increased reporting rates because they measure data movement.

1.0 Introduction

1.1 Motivation

The continuing co-evolution of hardware and software abilities, layered on social tendencies and trends, is leading us on a trajectory toward an environment that rivals science fiction writing of forty years ago. We are evolving toward an environment in which we can have smart homes [**ierp-2004**, House_n-2004, GreenHouse-2004, AHRI-2004, X10KB-2004, X10-2004], offices [WJH-97, BP-2000], smart construction sites [Botts-2003b], cities [Cooltown-2004, PlaceLab-2004] and beyond [**SER-2001**, Fulford-2002, Botts-2003a]; it is a place where vast databases are immediately available and searchable on just a few keywords [AOL-2004, Google-2004, Yahoo-2004, CiteSeer-2004, ULib-2004]; and, of course, where everyone is mobile and has their personal communicator (i.e., cell-phone++) [Rosenthal-2004, Scuka-2004, Tam-2004, Palm-2004].

Although prescient in many ways, that science fiction is a product of its era and failed to imagine our tremendous capacity to produce exponentially more information. Recent documentation claims the amount of information that an average business must process doubles every nine months [NH-2004] and information on the Internet doubles every twelve months [Berger-2004]. "[I]nformation, especially on the Net, is not only abundant, but overflowing. We are drowning in the stuff, and yet more and more comes at us daily. That is why terms like 'information glut' have become commonplace, after all" [Goldhaber-04]. Despite our complaints of too much information, we continuously generate new sources of raw data and add new capabilities to our computing environment. The convergence of trends is yielding an environment – an Internet experience – that is different in kind. In short, it will be a *ubiquitous, real-time, highly-mobile, always-on, event-based, content-focused, individualized, context-aware* computing environment, which we are coming to call the Pervasive Internet [Harbor-2002a].[1]

One research area attempting to deal with these growth trends is context-aware computing, which takes data from a highly dynamic environment and synthesizes it into information and knowledge for decision-making. It is based on the idea of reducing computational or cognitive load by getting the right information to the right person, device, or application at the right time. Previous context-awareness research focused on local information – either device awareness (e.g., available battery power) or as applied against a static backdrop (e.g., my location with respect to the stores in a mall). We take context-awareness to the next level to consider dynamic data in relationship to other dynamic data at an Internet-scale. We call this *Internet-scale context-awareness* (ISCA).

As one simple example, consider a criminal gang that has been arrested, tried, convicted, and paroled. A condition of their parole might be that the gang's members are not allowed to associate with one another (e.g., come within 100 yards of each other) for the period of one year. Then wearing a GPS-enabled ankle bracelet with wireless communications abilities, their positions are reported at one second intervals and are constantly monitored for parole violations [BBC-2004] by a parole officer in the field

[1] The Pervasive Internet terminology comes from "the fusion of pervasive computing, Internet connectivity, and new enterprise-level data-management applications and Web-based smart services" [Harbor-2003c] and was first seen in [Harbor-2002a]. I am using the term in a slightly more expansive sense.

who is using a lightweight edge-device. To conserve power, communications need to be minimized.

Similarly, teenagers with cell-phones are interested in buddy proximity. In fact, the cell-phone is likely to become the notification device of choice.[2] Other diverse usage scenarios can include monitoring energy supply costs in relation to home energy usage, financial decision making by commodities brokers, (remote) health monitoring across a variety of devices, and even traffic routing. As identified in various context-awareness surveys, specific context of interest depends on individual users and their current tasks and objectives [CK-2000, DAS-2001, Tarasewich-2003]. To effectively support context-aware computing in this environment requires efficient, extensible, and scalable data distribution and processing.

A second research area dealing with these trends, content-based publish / subscribe (CBPS), holds the potential for such a solution. A CBPS system is a middleware infrastructure that transparently connects subscribers to publishers based on the data content instead of on some administered channel or topic name. Publishers declare availability of event-types as a conjunction of attribute-constraints and then publish notification *events*[3] conforming to those specifications. Subscribers declare interest in event-types as a conjunction of attribute-constraints. The middleware provides efficient mechanisms to route matching events from publishers to subscribers. Hence, publishers and subscribers do not need to know about each other, thus enabling new publishers,

[2] With capability convergence, the cell phone is also likely to become a primary personal sensor.

[3] CBPS literature uses *events* and *notifications* interchangeably. However, the general term is *message*, as in a message passing system. In attempting to maintain parity with CBPS research, we need to use event and notification. Consequenty, we use these three terms interchangeably, based on the context of use.

subscribers, event-types, and subscriptions to be freely added to the milieu.

Not only does the middleware provide separation, but also efficiency; it filters out new events against subscriptions at the publisher's event-broker, achieves economies of scale by exploiting overlapping subscriptions, and employs multicast-like routing of events to subscribers. Efficient filtering at the publisher's event-broker is achieved by content-based pattern matching against a publisher's event in a series of independent filters (e.g., {(event.x < 10) & (event.y > 30)}). Sequences of events can be similarly pattern-matched [Carzaniga-1998, CRW-2000].

CBPS seems the most natural substrate for ISCA because it provides the right separation of concerns, efficient event distribution, extensibility, and scalability. Yet, its design is best suited to publishers for whom there are many interested subscribers (e.g., sports scores, news events, or stock tickers). The emerging information environment will be substantially different from that for which CBPS was designed. It is characterized by:

1. more producers (e.g., nearly every application and appliance will provide data),
2. more consumers (e.g. the producers will probably also be consumers),
3. continuous streams of data (e.g., apps and devices will constantly provide status),
4. higher sensor data rates (e.g., location data at 1 Hz) and
5. individualized usage patterns combining **dynamic** data from **multiple** information providers (e.g., proximity relationships are tailored by and for a single person).

Ultimately, we can anticipate that contextual data will derive from "billions of users connected to millions of services using trillions of devices." [CPTWY-2000] Despite the high frequency of raw event data, context-aware events will have a low frequency of occurrence. For example, a proximity relationship between two people may be comprised of location events from each user at a one second update rate (2 * 3600 reports per hour) even though the desired proximity may not occur for several days.

1.2 Problem

Context-aware relationships, like the proximity relationship, require attributes from different events, from different publishers, to be combined into a single expression for evaluation. However, CBPS does not provide the expressiveness required to evaluate attributes across event-boundaries. The black-box style of transparency afforded by CBPS currently requires subscribers to subscribe to the raw location-events and to compute the distance themselves. Hence the efficiencies of evaluating subscriptions at the publisher's event-broker are lost. By precluding efficient detection and publication of context-aware conditions, CBPS will lead to information glut and (low-powered) edge-device saturation; it will not scale. Sensor networks and streaming databases face similar problems when trying to determine relationships between dynamic data originating from multiple publishers.

Recent work enables the aggregation of attributes from multiple data streams with more complex processing and filtering performed within the network [CK-Solar-2002a, JS-2003]. Common aggregations and transformations can also be shared [CK-Solar-2002a]. It is possible, then, to evaluate the proximity relationship at the first common event-broker node (***1st CN***) that connects the publishers with the subscriber. When publishers are distant from the 1st CN, each intervening event-broker node must process and forward all events, which is costly. It also burdens the network core with the more complex processing. Moreover, the extra costs get shared with all other applications using the network. It is not scalable.

To achieve the desired scalability and the best possible performance requires evaluating context-aware relationships at the publisher's event-broker. To do so efficiently requires knowledge of the modeled relationship as well as how it will be used. Only the subscriber has this domain knowledge, and currently has no way of expressing it to the middleware.

1.3 Hypothesis

The increased expressiveness, efficiency, and scalability requirements for Internet-scale context-awareness can be achieved by extending CBPS without compromising its desirable separation of concerns. First, subscriptions may be imbued with the full power of a programming language in the form of first-class publish / subscribe clients, agents, such that complex filtering may be achieved at the event-entry edges of the network. These agents may be designed as distributed algorithms to enable independent filtering at the event-entry edges of the network despite the need for multi-publisher, attribute-to-attribute comparisons. Second, content-based routing techniques may be leveraged to allow the user to control deployment of these agents, to efficiently set up communications pathways between them, to develop complex relationship hierarchies, and to increase the throughput of the middleware.

1.4 Conceptual Approach

Expressiveness is directly addressed in two ways. First, subscriptions are allowed to apply the richness of a programming language. Thus, dynamic, off-axis (e.g., $X/Y < c$) and more complex filtering may be directly expressed. Second, content-based routing techniques are used to control deployment of these agent filters. This gives users control over the deployment (without needing to know anything about the structure of the network) and provides the ability to express complex, distributed filters.

Efficiency is addressed in three key ways. First, distributed filters are constructed to enable more events to be filtered on entry into the network and hence fewer events need to pass through the system. Second, content-based routing techniques are used to deploy other subscriptions into the network to efficiently set up communications pathways without causing a flood of advertisements or subscriptions. Third, distributed memoization is introduced to allow edge-brokers to hide multi-attribute subscriptions inside single-attribute subscriptions, hence improving the throughput of the internal nodes.

Scalability is addressed in two key ways. First, moving complex filters from the subscriber into the network allows more lightweight edge-devices (e.g., cell-phones) to participate in context-aware subscriptions. Second, moving complex filters to the event-entry edge-brokers removes the bottleneck of computing the complex filters in the network core and distributes the processing among a far greater number of processors. Third, by providing distributed memoization, internal processing costs are reduced, which allows the network to grow.

In sum, we recognize that CBPS, like any component-based system, is an ecology for which we can achieve global efficiencies by providing top-down control, bottom-up context and bi-directional collaboration. Thus, from the top-down, we wish to allow subscribers to inject their contextual knowledge into the network; from the bottom-up, we wish to expose more contextual information about the network and messages that pass through it.

1.5 Implementation Approach

We introduce Fulcrum[4], as an extension to CBPS, to create an efficient substrate for Internet-scale context-aware publish / subscribe while preserving the anonymous, asynchronous, and loosely coupled nature of CBPS.

Fulcrum extends CBPS with top-down information control using an open implementation approach [Kiczales-96, JS-2003]. In the open implementation approach, a module's interface is designed to allow a client to assist in the selection of the module's implementation strategy. The module's auxiliary interface may allow a client to describe its usage patterns (e.g., high insert rate, few deletes), to specify an implementation (e.g., hash table), or even to provide its own implementation – adhering to well-defined interface specifications. This allows the subscriber to tailor the middleware's implementation strategy to better suit its needs, while retaining the advantages of closed implementation modules (i.e., the traditional black box).

Our research allows subscribers to inject domain-specific knowledge (implementation strategies) into the network in the form of first-class publish / subscribe Java applets (agents) as attachments to their context-aware subscriptions. Thus, a subscriber can subscribe directly to a *derived relationship*, such as the distance between two entities, for which there is no publisher, *per se*. Leveraging CBPS content-based routing capabilities, we introduce *deployment slips* to allow the user to control the location where the code is deployed. We are, in fact, programming the network.

These *implementation strategy* agents serve two purposes. First, they serve as

[4] "Give me a lever long enough, and a fulcrum on which to place it and I shall move the world." Or "Give me a lever long enough and a place to stand and I shall move the world." Archimedes, 230 BC

proxies to the sensor to observe, transform, and filter data in ways that a basic query / subscription cannot. Second, they serve as distributed algorithms that collaborate in the filtering process to determine the satisfaction of a given relationship property. This is particularly necessary for high frequency raw events that combine into low frequency context-aware events. Note that the same property can be implemented by different strategies as appropriate to the context of use. Similarly, the same implementation strategy can be reused among properties with similar semantics (e.g., the notion of distance is not merely physical, but could be economic). Leveraging the content-based routing capabilities, we introduce *routing slips* to efficiently set up communications pathways between the agents in a distributed algorithm, thus preserving CBPS's desirable separation of concerns property, while exercising considerable control over the placement of computations in the network.

The agent algorithms are based on the "law of continuity" – that for an entity (e.g., the relationship in question) to change from one state to another, all the intermediate states must be visited. Consequently, we can think in terms of the distance between two alternatives along a number line – only the units change across domains (physical, financial, temperature, pressure, etc.). The distance is inherently continuous. Thus, we know that the distance must first shrink by half, by half again, and so on before a desired proximity can be reached. Thus, bounding regions can be created to allow each sensor proxy to act independently most of the time and to exchange data only on an as-needed basis. Consequently, these agents are able to eliminate useless event traffic at the sensor-edges of the network.

Efficient distributed algorithms and context-awareness in general require information about the network. Increasing bottom-up information availability is achieved in two ways. First a dynamic message context is attached to every message to provide system-level contextual information about the message's passage through the system. This then allows us to add contextual filtering on top of the existing content-based filtering. (E.g., limit distribution to 3 hops.) Second, each information component of a message is given a cache identifier (i.e., a hash key), to be reused across event-brokers to enable the distributed memoization scheme. Once a match for an N-attribute subscription has been determined at an edge-broker, the cache identifier is passed on as a single-attribute notification to subsequent event-brokers to avoid re-computation at downstream nodes. We achieve this by leveraging the content-based routing mechanism with the introduction of an *express forwarding slip*.

1.6 Evaluation

We evaluate the effectiveness of our solution in comparison to basic CBPS along three dimensions: expressiveness, efficiency, and scalability.

Does the middleware increase expressiveness? Can the user express the relationship of interest even when it requires attribute-to-attribute comparison, across multiple events that originate from multiple sources? Basic CBPS limits subscriptions to conjunctions of attribute-constraints, where the constraint is limited to a pre-defined constant, and attribute-to-attribute comparisons are not possible. Fulcrum retains that capability and allows the user to inject and control the deployment of domain-specific knowledge into the network using open implementation techniques. The user is given the full expressiveness of a programming language.

Is the middleware efficient? Basic CBPS achieves local efficiencies by limiting expressiveness and global efficiencies by eliminating common subscriptions. Fulcrum retains these basic capabilities, but it is designed for an environment where very few common subscriptions exist. Therefore, additional efficiencies are achieved in two ways. First, allowing the user to provide domain-specific distributed algorithms reduces unnecessary event traffic. For the example of monitoring for proximity parole violations, Fulcrum enables a reduction in the event traffic from O(|events|) to an expectation on the order of $\lg_2$ |movement|. The same algorithm applied to detecting the proximity of mobile buddies using then current positioning technologies, with an average reporting interval of 17 seconds, reduced event traffic from O(|events|) to an expectation around 4ln(|events|). For vehicular traffic-route monitoring with a reporting interval of one minute, normalized by the number of nodes performing relationship computations, Fulcrum achieves an average event reduction over basic CBPS by 8:1 and has a performance cross-over with respect to the 1st CN as the event relationship complexity increases. When considering the effects of load balancing, Fulcrum achieves a performance improvement over basic CBPS of 12:1 and over the 1st CN approach of 94:1. In both cases, increasing the reporting rate will dramatically affect the processing requirements of basic CBPS and 1st CN approaches, but will have little effect on Fulcrum. Second, internal-brokers leverage computations performed at the edges, such that once an event matching has occurred, the internal-brokers are not forced to recompute the matching from scratch.

Is the middleware scalable? Basic CBPS is effectively limited to environments where economies of scale may be leveraged through shared subscriptions. We increase the scalability using three techniques to remove bottlenecks from the core of the system. First, we eliminate unnecessary event traffic through independent filtering at the event-entry edges of the network, thus reducing the overall burden on the system. Second, by moving complex computations from the interior to the sensor-edges of the network, the workload is better distributed; there are far more edge-brokers than all internal-brokers combined. Third, by allowing internal-nodes to leverage the computations of edge-nodes through our distributed memoization technique, we reduce the burden on the core system.

1.7 Overview

Chapter 2 describes the need for Internet-scale context awareness and the opportunity created by the emergence of the pervasive Internet. It provides a collection of motivating scenarios, estimates of the number of producers and consumers for a variety of sensor data, and analyzes the recurring themes found among the scenarios. It then discusses ubiquitous and context-aware computing as the two technology ideas that form the basis for our approach.

Chapter 3 reviews closely related work in context-aware computing and content-based publish / subscribe research with respect to our evolving ISCA needs. In this section we recognize the benefits of earlier technologies and consider their drawbacks with respect to ISCA requirements. This review identifies why and where we must create new technologies.

Chapter 4 describes our open implementation-based solution. We leverage the strengths of prior CBPS research and introduce new concepts to provide the necessary expressiveness and information availability. This chapter first provides an overview of a desired solution, then presents our solution within the context of a simple proximity relationship example, and finally adds depth to concept details not fully explored in the example. This chapter primarily addresses efficiency and expressiveness issues.

Chapter 5 presents the distributed algorithms used to achieve efficient detection of context-awareness relationships. Simple algorithms, based on the “law of continuity”, are presented for pair-wise proximity and group proximity. The scalability of the approach is demonstrated in the development of traffic routing algorithms. Lessons learned through developing several algorithms demonstrate a common idiom and have led to the creation of several common “widgets” for building algorithms that enables reuse; these are presented. This chapter primarily addresses expressiveness and scalability.

Chapter 6 discusses the architecture and implementation details. This chapter reviews the motivations and design decisions for performance and extensibility of the architecture; it discusses the general architecture; and it details the major components of the system, including: the pub / sub clients, the implemented communications infrastructure, event-brokers, and the database and fast-forwarding algorithms. The details include message specifications, application programmer interface details. Snippets of source code are provided where appropriate. Additionally, sample source code is provided for a publisher, subscriber, and an implementation strategy agent.

Chapter 7 evaluates the performance effectiveness of the system with analysis and experiential results of both trivial and complex contextual-relationships as well as for the distributed memoization mechanisms. We primarily use event-hops to compare the behavior with respect to basic CBPS and 1st CN approaches. We then evaluate the load balancing characteristics of distributing the relationship computations to the edges of the network. First, results from (trivial) pair-wise proximity testing, using a range-ring implementation strategy, are provided. Second, results from (complex) traffic monitoring, using a mediator with multiple agents, are provided. Finally, performance measurements for the distributed memoization mechanisms are provided.

Chapter 8 provides a general discussion of tradeoffs, considers outstanding issues, and sketches future research directions. The tradeoffs consider the cost of deploying subscriptions, the potential to implement our innovations using other infrastructures, and why systems might choose to host remote code. The outstanding issues examine ways users might unintentionally or maliciously harm the infrastructure (e.g., through denial-of-service-like behaviors). The future research directions touch on productization and usability, explore additional efficiency measures, and consider issues associated with agent behaviors.

Chapter 9 concludes and details the contributions of this dissertation.

2.0 Problem and Opportunity

Our ability to efficiently act on opportunities or needs in the world is limited less by the information available than by our ability to observe it, reason about it, and act on it in a timely fashion.

As only one simple example, imagine a criminal gang has been apprehended, tried, convicted, and paroled; where a condition of the parole is not to associate with one another for the period of one year – as defined by being co-located within 100 yards. Their parole officer in the field then needs to learn when and where gang members are in violation. Today, she would be sorely challenged to get the information and would be overwhelmed in keeping track of every movement of every parolee in order to determine parole violations. The information overload becomes a more significant issue when we desire to track additional kinds of proximities, monitor different kinds of data relationships, or use lightweight edge-devices.

To efficiently use our time and reduce cognitive load, what we would really like is the promise of context-aware computing, to be notified only when something of actual interest, to us personally, occurs, and not be bothered otherwise. Even better, we might want a third party, whether person or application, to act on our behalf, taking us out of the loop entirely. The condition on which to trigger such an event notification would be expressed as the relationship between the attributes of dynamic data, originating as individual events from potentially multiple publishers.

The technology for the above scenario is already here and more is coming.

Location-based services are becoming ubiquitous; sensors are getting smaller, wirelessly networked, and are less power hungry. As a consequence, they are becoming more pervasive. Software sensors are becoming popular and emit a continuous stream of messages to record every significant action of an application. The Internet will permit us to network these sensors, so that we can receive the necessary event streams as part of an open architecture that we can configure to our desires.

As we evolve into a pervasive Internet environment[5] with "*billions of users connected to millions of services using trillions of devices*" [CPTWY-2000], we are increasing the amount of data available, doubling it about every 9 months [NH-2004], mostly by increasing the efficiency with which we create information. An important question is how we can harness this data, yet avoid information overload. "One important goal of ubiquitous computing, however, is to help the user overcome information overload and concentrate on the current task." [Weiser-91]. We must take the personalized information requirements of a billion users and provide context-aware computing techniques to reduce the amount of information of which each user must be aware, yet we must do so in a way that does not overwhelm the network itself, especially at the less-capable, user-facing, edges. Complex relationships will need to be inferred in and distributed throughout the network. For example, to avoid a flood of location reports nominally required to determine the proximity between entities, the network will need to compute distances between entities, and only forward the relevant ones.

This chapter is organized as follows. We start by detailing several examples that

[5] The pervasive Internet terminology comes from "the fusion of pervasive computing, Internet connectivity, and new enterprise-level data-management applications and Web-based smart services" [Harbor-2003c] and was first seen in [Harbor-2002a].

demonstrate the need for a diverse and personalized set of data relationships. These include the various data relationships necessary for decision making as found in environments like location-based proximity relationships, energy use / cost management, commodities brokering and financial management, health monitoring, and traffic routing.

Next we examine how the ideals of context-aware computing in a ubiquitous environment can reduce the information burden on the end-user by reducing the information load to that which is relevant to the user's current needs.

Then a brief sketch of one promising technical approach is provided. It uses a content-based publish / subscribe (CBPS) infrastructure to provide the necessary reach and the ability to easily plug-in information providers and consumers.

Finally, the core technical challenges are discussed, such as the complexity of the information relationships, the need for a general purpose solution and unconstrained information access, the personalization of contextual relevance, and the tradeoffs between expressiveness and the ability to scale to Internet proportions – and the exponential growth of information, etc.

2.1 Motivating Scenarios

The pervasive Internet environment with its continuously expanding information flow provides more data for us to work with, not only improving the decisions we make, but also increasing the speed with which we are able to make such decisions.

> *The Internet's most profound potential lies in its ability to connect trillions upon trillions of fast, smart sensors, devices, and ordinary products into a global "digital nervous system" that has been dreamed about by visionaries since at least the 1940s.* [Harbor-2003c]

Perhaps the most significant new feature of this evolving terrain is our ability to observe and collate more diverse information from multiple information producers in real-time. As can be seen in the following examples, we want to use this available data to make informed decisions in a timely fashion. Example scenarios are described for five classes of problems: location proximity, energy use / cost management, financial decision making, health monitoring, and traffic routing. More examples, like those that follow, can be discovered every day. As supporting capabilities are developed, more will be envisioned and demanded.

2.1.1 Location Proximity

Location proximity, specifically that of criminal-gang members violating parole, will be used as a running example throughout this dissertation. This example is useful due to its simplicity, the recurrent reporting of one's location, the high profile of location-based services, the third party observer characteristic of our example, our ability to readily understand 2- or 3-dimensional, geographically-oriented data, and our ability to easily recognize the generality and scalability issues of such a problem.

The basic problem is to know when two people are ***near*** to one another. The idea of *near* is context-dependent and hence is user defined to correspond with the circumstances. In the following scenarios, we also show that *near* need not be symmetric between two people and that the location proximity request might be defined by a third party observer and not be either of the participants.

Scenario 1. Criminal-Gang Parole Violation Detection. Consider first the case of our gang members. They are caught, tried, convicted and ultimately released on probation with the condition that, for the period of one-year, none of the gang members are allowed to associate with one another. For this group, the court decrees that association is defined as any two members being co-located within 100 yards. For another group, association might be defined as three or more members within fifty-yards.

To monitor their activity, each is outfitted with an ankle bracelet with a GPS device [BBC-2004] that constantly reports their position. The parole officer then needs to monitor and analyze all the position reports to determine any parole violations. The continuous flood of position data would place a significant strain on the parole officer who has responsibility for not one gang but perhaps hundreds of people. The natural response to reduce overloading is to down-sample the data to some fixed rate, say to one report a minute. However, this creates an opportunity to be exploited. Attempts to hide or secure such decisions are seldom successful as the system capabilities are constantly exposed to public view.[6]

Scenario 2. Restraining Order Violation Detection. In the case of a restraining order, one person might be prohibited from coming within 100 yards of another or their domicile. Again, similar position reporting technology might be used and again, someone or something would need to monitor all the data reports in order to determine violation and risk of property damage or bodily harm. In certain high-risk cases, a police officer monitor might first wish to know when a 1000 yard threshold has been breached, followed by 500 and 250 yard limits. The rate of change over these thresholds as well as the amount of time inside a single band – indicating a potential stalker – is important.

These two scenarios describe situations where one might wish to centralize the monitoring service, say at police headquarters. However, global travel is easy, and

[6] Criminals caught by the San Diego Harbor Police manage to communicate with others how they were detected; consequently criminal tactics to evade detection constantly evolve – as reported by Lt. Ken Franke.

scalability in terms of distance begins to infringe on such solutions. The following location-based scenarios also consider scalability in terms of the quantity of information providers and consumers. Later scenarios include scalability in terms of the number of different types of information that are interesting.

Scenario 3. High-Risk Sexual Predator Monitoring. Convicted rapists and pedophiles are significant concerns to society. Under California's Megan's Law [caag-2004] these persons must register with their local law enforcement agencies and their personal information, including the address of their residence is made public.[7] This information, although fairly static, does affect where people choose to live. What is desired is more dynamic information. A woman, going about her business alone, might feel (and be) more secure knowing there was no known threat within a quarter of a mile. A parent today is concerned allowing children to play outside without constant vigilance. The ability to be informed that a higher level of risk exists would be valuable. Again, the same or similar location reporting technologies can be applied.

Scenario 4. Serendipitous Meetings. It is common that more can be accomplished during a two-minute face-to-face discussion than sometimes occurs after hours of e-mail or telephone exchanges. Sometimes there are things to share that would not be conveyed well over other mediums or for which we wish no firm record. In such cases, the "impromptu hallway meeting" serves our needs well. We could improve the likelihood of such meetings if we could monitor the location of our desired collaborators or somehow be notified when they are within, say, ten yards, thus alerting us to their

[7] The registration requirement has existed for more than 50 years. Megan's law is referenced because it is well known. However, it actually imposes the requirement to inform the public and it also expands the amount of information available to the public.

presence and providing a reminder of why we wanted to talk.

Scenario 5. Demand Meetings. Meetings that are expected, yet due to environmental constraints might be difficult to coordinate effectively. Noisy and crowded places, such as concerts, subways, county fairs, mall food-courts, etc., can be difficult places to coordinate meetings, especially when the area is unknown and the participants' timely arrival is uncertain. First responders to accidents, fires, floods, earthquakes, etc., are often in situations where they need to know when a higher-level authority has arrived on-scene in order to transition command and provide a status briefing. Yet, in the activity required, no time or consideration can be spent looking-out-for or waiting-on such a person. Similarly, when the authority arrives, chasing down the person who knows the whole situation can be challenging. It would be useful then to be able to know when such a change-of-command, debriefing meeting is possible. In the case of a large fire, *near* might be defined as a range of 250 yards.

As an example of an asymmetric relationship, we can imagine the same two people have a relationship both in Scenario 4 and in Scenario 5, except the person who is interested in the meeting are different and the significance of the meeting is substantially different as well. In either case, each wants to be notified that the potential for a meeting has occurred, thus creating an asymmetric situation where the range for *near* differs.

Scenario 6. Meeting Monitoring. Large companies are frequently composed of inbred fiefdoms such that high level management goals might be to increase cross-fertilization among groups. By monitoring proximity among individuals and knowing the roles they play, an evaluation of how frequently face-to-face meetings occur across groups is possible.

Scenario 7. Group Detection. We can extend the one-to-one meeting detection capabilities to groups in many ways. It could be the detection of a mob of protestors or a group of friends; it could be the detection of a necessary quorum or a requirement that a busload of school children on a field-trip all be present or accounted for.

Scenario 8. Ad hoc Wireless Routing Management / Support. The scenarios described so far consider the "coming together" of entities. The dual to this is the "going away from" of entities. Consider an ad hoc relay network that is put together as part of a first responder network or as part of a military operation. The individuals carrying devices need to be mobile, to have a certain freedom of movement, to achieve their goals and to complete their tasks. However, they also have a constant need to be in range to be connected – either to report findings or receive alerts or commands. They also have limited power supplies; thus they wish to keep transmissions to a minimum. Therefore, they want infrastructure support to efficiently recognize the ability to be connected and event notifications to warn when the user is at risk of moving out of communications range – before they inadvertently do.

Whenever the discussion of high availability of information occurs, there also arises a frequent concern over privacy. We can address some of the obvious concerns over privacy and big-brother monitoring here by recognizing that in some cases, the subjects may not have a choice (e.g., criminals) while in other cases there may be the voluntary release of information (e.g., children to be chaperoned). An interesting aspect of the efficiency equation is that – in certain limited instances – people are willing to give up a certain amount of privacy.

> *A long line waits at the Minneapolis-St. Paul International Airport on Monday, where frequent business fliers wait their turns to sign up for a 90-day project that allows them to trade personal privacy for travel efficiency. The project is the first in the country and includes 'biometric' identification, which involves eye scans, fingerprints, and criminal background checks. Participants will be able to pass through a special lane at the security checkpoint beginning in July.* [Mone-2004]

This demonstrates a trend that efficiency enters the equation with privacy. This allows us to place privacy concerns as a back-burner issue; it is not the 1st order driver; if people like the service or efficiency enough, they will accept it as is. In short, there is a cost / benefit tradeoff that people are making. This is analogous to car rental companies placing GPS devices in their cars, charging less if people sign the waiver to be constantly monitored and substantially more if people want to avoid it. Identifying the point at which the perceived tradeoffs result in people unwilling to relinquish privacy is beyond the scope of this research.

Additionally, as will be shown later in the dissertation, we have developed techniques that effectively inhibit capturing all information, focusing instead only on the key proximity relationships of interest.

From a technology perspective, we can understand location-based proximity relationships in six dimensions. First, we have the number of ***kinds*** of information that might be published – for location this amounts to one type, composed of latitude, longitude, and possibly altitude/elevation/depth – and should be time-stamped as well. Additionally, the convergence of sensors might provide the additional data of heading, course, and speed [Philips-2004]. Second is the number of different instances (of the given kind) originating from the same reporting unit (e.g., sensor). For a personal location sensor there would be a one-to-one mapping with the number of kinds, or even just a single composite information report. However, a provider of stock quotes might have one kind but hundreds of instances of stocks to report. Third, we estimate the expected quantity (or rate) of reports – for typical location sensors this is likely to be at 1/second but may be as low as 1/minute. Fourth, we estimate the number of information providers – for location sensors, we can estimate over a billion, if for no other reason than an expected convergence of location based technologies with cell phones. Fifth, we estimate the number of different instances of data a consumer is likely to be interested in – for typical personal location sensors there are likely to be in the high single-digits of interested consumers.[8] Sixth, we estimate the quantity of potential consumers for the information – we estimate that most information providers will also be consumers. These dimensions can be viewed at a glance in Table 2-1.

Table 2-1 Personal Location Device as Information Provider

Qty:	Single	Few	Medium	High
Kinds	1-4 (1)			
Instances	1-4 (1)			
Quantity			(1/sec) - 1/min	
Providers				10^9
Consumer@		1-20+ (9)		
Consumers				10^9

Where ranges are given, a number in parenthesis is provided to estimate an average. Despite the four different kinds of information, they will typically all be wrapped into a single position report with each component being treated as an attribute of the report. In the case of consumers, the "@" annotation indicates how many location events from different providers might be interesting to the common consumer, while the "+" indicates

[8] The 2002 UCSD ActiveCampus data shows an average of 6.73 designated buddies per person, but only 3.37 were mutually agreed upon (i.e., bi-directional). We expect a higher average if everyone, not merely on-campus friends, could be identified and if the system is used for a longer period of time to make the entry cost versus usage tradeoff favorable.

an expectation that a few consumers will be interested in many more location events than average (e.g., a cowboy with a herd of cattle or a teacher with a busload of children).

This table (and those that follow) provides insight into the enormity of the data that we are facing and that individuals will be interested in different subsets. To support such demands will require efficient techniques in addition to significant processing power.

2.1.2 Energy Use / Cost Management

Southern California suffers from an energy crisis that in summers past has required rolling black-outs and unintentional brown-outs. Current challenges with involving customer assistance in reducing consumption include the coarse-grained event reporting and the inability to disseminate current loads. Consequently, there is no dynamic mechanism to support self-regulating supply-and-demand based markets. With newer technologies, such as broadband power lines (BPL) [Coursey-2003,Forrest-2003], and the ability to network all home appliances, the energy consumption of individual consumer electronics as well as overall household consumption could be measured and reported in real-time. Similarly, the energy providers could report the overall usage, available capacity, and projected cost and status values to the consumer base. The energy providers could then apply different pricing based on current usage rates with respect to available supply in real-time. The combination of market pressures and looming threat of black-outs would encourage cooperation / collaboration with the client base. Using UpnP [UPnP-2005], X-10 [X10-2004], Home-Plug [HomePlug-2005], or ZigBee [ZigBee-2005] enabled devices; clients could remotely control a variety of devices in their house to further reduce energy consumption in times of great need as well as on a daily basis. Such actions might include reducing the cooling settings in the refrigerator, freezer, or air conditioner. Failure to participate would simply result in higher energy bills and in the worst case, suffer the whims of blackouts. Using the same data categorization as for proximity, we estimate utility company information in Table 2-2 and home gateway information in Table 2-3.

Table 2-2 Utility Company as Information Provider Example

Qty:	Single	Few	Medium	High
Kinds		3		
Instances		3		
Volume		1 / 10 sec		
Providers			10^4	
Consumers				10^4-10^7

Each utility company is an information provider with a large following. There may only be a few kinds of data event types (e.g., usage, capacity, cost) to provide with a likely one-to-one mapping between the instances and kinds. The update rate might be one of the three possible events every 10 seconds. Across the United States, there are thousands of utility companies and the number of information subscribers could easily be in the 10's of millions.

Table 2-3 Home Gateway as Information Provider Example

Qty:	Single	Few	Medium	High
Kinds			40-280	
Instances			40-280	
Volume			1 / sec	
Providers				10^9
Consumers		1-5+ (2)		

A home gateway that collects information from all the devices in the house and rebroadcasts it onto the network may be responsible for 280 [Pinto-2002] devices and five user defined aggregations (for five household members) and have a one-to-one mapping between instances and kinds of events. The update rate might be to send one event each second. The only consumers would be the normal occupants. Precautions would need to be taken to ensure the data content is not completely open inviting burglars while on vacation.

2.1.3 Financial Decision Making

Scenario 1. Commodities Brokers. As an example, brokers currently monitor as many as nine data relationships between different real-time data streams when determining whether or not to buy or sell a given commodity.[9] Such data includes not merely current commodity bid and ask prices, but also currency rates, and even weather reports as they indicate floods / droughts and other crop affecting circumstances.

> *The system uses relationships with elements of the US economy, rather than price movement, to determine entry and exit. These relationships include: the current status of the yield curve as measured by five-year and ten-year yields; the correlation between the instrument and semi-precious metals as represented by silver futures; the correlation between the instrument and industrial-use metals, as represented by cash nickel prices; the correlation between the instrument and the stock market, as represented by the S&P 500 futures prices; and reliable seasonal tendencies of the instruments.* [ts-2004]

The ability to monitor this information and come to accurate decisions faster is valuable. To some extent, stove-pipe[10] capabilities exist. However, finer granularity in reporting weather status can give earlier indications of commodity shortages. The introduction of new sensor data will affect the relationships monitored. More data and hence data relationships, in a more timely manner, with finer granularity of content, will

[9] As reported by a commodities broker, John Michael Schwaebe, 2004

[10] A “stove-pipe” or “silo” reference is commonly used to indicate information or capabilities in isolation.

be valuable.

Scenario 2. Portfolio Monitoring and Management. Fund managers or day traders especially, but also individuals managing their personal portfolios, might consider such data as the value of the stock they have purchased on margin as a percentage of their overall financial portfolio, which might consist of information about other stocks owned, dividends expected, savings, real-estate holdings, current interest rates, and so on. Although many of these are reasonably static (i.e., very low update rates), some are highly dynamic. In Table 2-4 we explore expected usage of stock exchange information.

Table 2-4 Stock Exchanges as Information Providers

Qty:	Single	Few	Medium	High
Kinds		3		
Instances				100’s
Volume				25 / sec
Providers		Low 10’s		
Consumers				10^7

The Stock Exchange (e.g., NYSE) might publish three different categories of data, stocks, bonds, and mutual funds, yet it reports on hundreds of different entities. It publishes the current rates for everything once a minute, yielding, say 25 events per second. And there are millions of consumers of who want some piece of the data (very few want it all).

2.1.4 Health Monitoring

Medical facilities are often hampered by interoperability problems, thus limiting choices. However, individual monitors only tell a part of the story about a person’s health. Sometimes it is the combined information from several monitors that can give early warning. When brought into the context of the availability of the patient’s physician, as described by presence, ability to be interrupted, and physical proximity, we find a series of data relationships from multiple data streams that become important.

As technology continues to shrink, constant monitoring of our personal health becomes possible. This might include monitoring bloodstream conditions to skin conductance for everything from diabetes to diet, exercise, or stress monitoring.

> *The MP3RUN is a combined speed / distance monitor and music source, jointly developed by Philips and Nike. It provides runners with stereo music while on the move, as well as audible feedback of performance related parameters such as distance covered, running speed, and elapsed time. This information is determined by a sensor on the running shoe, wirelessly transmitted using Bluetooth ® to the MP3RUN unit on the athlete's arm, and then relayed by a pre-recorded human voice through the headphones*. [Philips-2004]

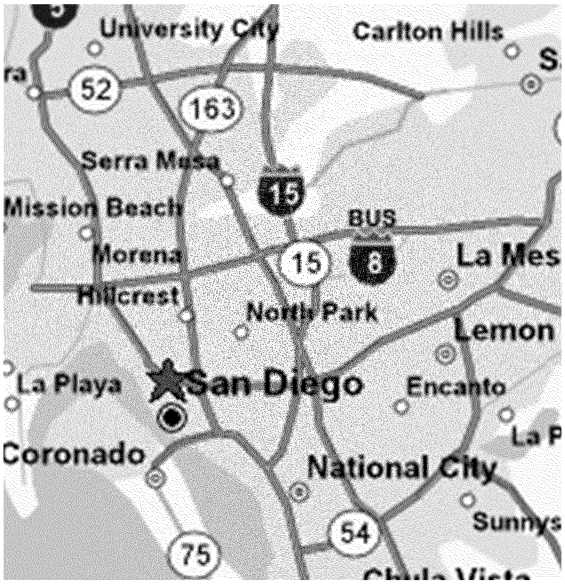
These sensors are clearly not at what we think of as an Internet scale yet they share similar characteristics. But, when their information is shared across the Internet we can provide constant medical monitoring and early detection of high-risk conditions while still allowing people to go about their lives.

Here, the cost to transmit information will be relatively high even for a personal area network. This information might then be used in conjunction with driving conditions to head off road rage. Finding ways to share only the critical information is important.

2.1.5 Traffic Routing

The CalTrans 2002 State Highway Congestion Monitoring Program Annual Report states that for California, the total daily delay on monitored freeways is 1,024,223 vehicle-hours, at a cost of $11,941,462 and 512 tons of emissions [CalTrans-2003]. This

$3 billion annual cost[11] for California and $68 billion nationwide[12] is cause in part by lack of visibility into information about alternative, less congested routes.



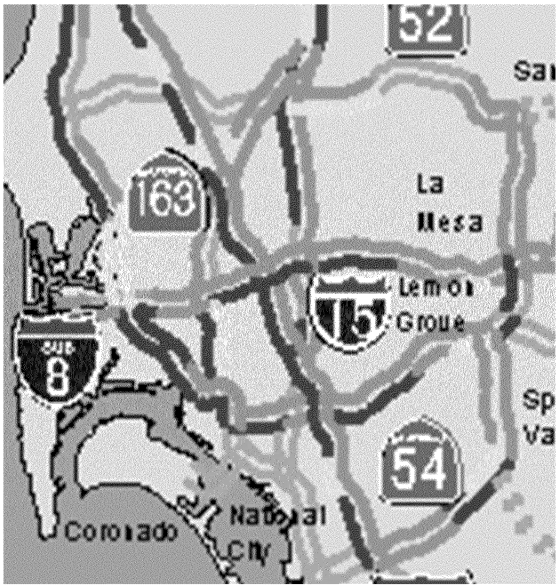


Figure 2-1 Mapquest View of San Diego Freeways

Figure 2-2 San Diego Traffic Management Center Speed Annotated Traffic Map. 1-35 mph (red), 36-50 mph (yellow), 51+ mph (green), future activation (gray), future construction (gray dashed)

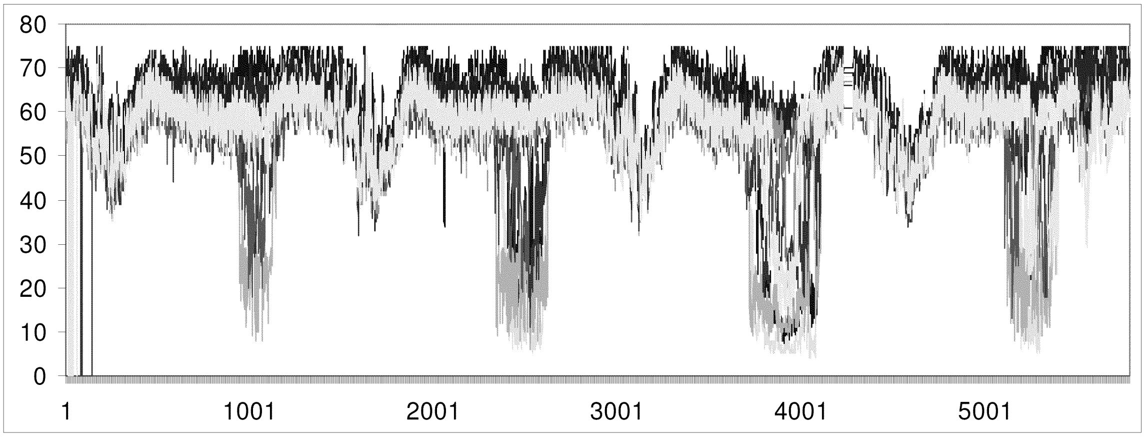


Figure 2-3 CalTrans Loop-Detector Sensor Readings. I5 North (Carmel Valley Rd to Cannon Rd). Speeds (mph) (y-axis) are reported at one minute intervals (x-axis) over four days. Major dips coincide with evening rush-hour traffic. The third day represents a 6-hour window (due to rain).

Listening to traffic reports over the radio or viewing colored-coded "live" Internet

[11] Only weekdays are counted.

[12] As reported by CBS, February 14, 2005

maps does not help much in determining whether an alternative route should be used, because the information is not provided in the contextual framework of the individual - thus leaving the user to habits and rule-of-thumb judgments. We want the essence of context-aware computing – to get the right information to the right person at the right time. The goal then is to perform the computations efficiently and to only burden the user with useful 'change' information. We need do this for millions of users throughout the day.

Scenario 1. Finding the best path, say from work to home, varies by hour, day of week, holiday, season, construction, accidents, etc. The goal is to get home and avoid traffic delays, whether caused by normal congestion or any unusual slowing – say from road debris or an accident. Our ability to dynamically determine the most efficient, least stressful route can take on new dimensions with available sensor data. We can combine traffic sensor data from CalTrans-monitored road segments, geographic information, and personal task information to determine the most likely routes home based on key freeway and surface street intersections and a historical record of my prior traveled routes. We then need to monitor these route times to determine when an alternate path is preferable. To continue the information cycle, the current estimate to arrive home is a by-product that then would be published to the network and consumed by agents that control our home environment.

By incorporating other sensor data including personal emotional state [ACMTN-2004] and health monitoring, the meaning of *best* may shift during the ride home. Initially, it might be the shortest path home (both temporally and physically), which later evolves to the temporally shortest path, to one which includes short errands, to one urging the driver to go take a break, have dinner and get back on the road later.

Scenario 2. With present technology, vehicles could be outfitted with the appropriate capabilities to collaborate with the traffic light system and other vehicles. Lights could also share their expected time sequence, either a set of fixed rates or estimates based on traffic variations. The vehicles could bring into play the multitude of onboard sensor data plus information about the driver, his calendar, and his urgency. Without human attention, visibility, and cognitive limits, the information incorporated could extend out to the next N traffic lights (e.g., 6 hops). The result could be a much smarter, more efficient, and safer traffic system.

Table 2-5 Traffic Sensors as Information Providers

Qty:	Single	Few	Medium	High
Kinds	1			
Instances	1			
Volume			1 / min	
Providers				10^4
Consumers				10^4-10^7

In California, loop detectors are used in Los Angeles, San Bernardino, San Diego, and Irvine in order to report speeds and determine traffic congestion. [CalTrans-2003] The website for San Diego aggregates the sensor data across all lanes and then makes the speeds directly available for 338 road segments[13] with a refresh rate of once per minute [Dist11-2004]. California alone has nearly 2000 monitored segments and we can estimate ten times that many exist nationwide. We have tens of millions of drivers that would be interested in comparing the travel times across their optional routes.

[13] These are directional segments. Often sensors do not cover the same segment in opposite directions. Los Angeles, San Bernadino, and Irvine speed reports are buried behind a Java-applet-based user interfaces.

2.1.6 Scenario Analysis

In each of the example scenarios there are four recurring themes. These form the basis for problem formulation and the analysis criteria by which we will evaluate existing capabilities and our solution.

2.1.6.1 Highly Personalized

The data relationships of interest are highly personalized even if the data elements are reused. Take our criminal gang member (CGM). The parole officer is concerned with parole violations – CGM's location relative to other criminal gang members; the lone woman is concerned with her safety – CGM's location relative to hers; the criminal is interested in hanging-out with his buddies – CGM's location with respect to the locations of his new friends. Although the raw data of CGM's location is reused at least three times, each data relationship is different and is unlikely to be reused.

For my commute, I am interested in one of two paths home; the differences between the two paths are composed of a total of fifty road segments. This sensor data will be shared with thousands of other interested persons at the same time. However, few if any of these will be considering the same route trade-offs at the same time. There might however be useful aggregations of sensor information across stretches of highway, say between major freeway intersections.

A bachelor managing his home energy usage would be the only one who cares about individual appliance energy usage at his home with respect to energy costs; both he and the electric company would be concerned with overall usage; and yet everyone serviced by that energy company (plus a few regulatory agencies) would care about current electric rates.

2.1.6.2 Multiple Variables, Multiple Publishers, Soft-Real-Time

The relationships of interest concern the relationships of multiple variables, reported from multiple publishers, in real-time. To consider location proximity, we must simultaneously consider the location reports for all participants, as reported from their personal devices and then apply distance computations to relate the positions. When comparing traffic-routes, the component parts of the relationship originate from widely dispersed sensors. When evaluating energy consumption costs, one set of data originates from the home while the other comes from the energy company.

2.1.6.3 Identity Correlation

Each of the information components is uniquely identifiable. Each report of my location must be tagged with correlatable information that allows it to be mapped back to me.[14] It is expected that different sensors will provide different mechanisms to identify the data they produce. It is also recognized that in a variety of circumstances that uncertainty of identity exists. This topic is beyond the scope of this dissertation. Additionally, considering a class of person (e.g., illegal alien) is beyond the scope of this dissertation, but it is appropriate for follow-on research.

2.1.6.4 Recurring Reports, Perishable Data

The dynamic data is, in general, recurring and perishable. For example, my location is meaningful as “my current location” only until the next time I get a sensor update and re-report my position. We develop an information usage model, shown in Figure 2-4, for near-real-time data as a means to better understand how it must be

[14] It would be naïve to claim a simple unique identifier. For example, loop detectors are identified by route, direction, and sensor name.

mastered in a ubiquitous, context-aware computing environment.

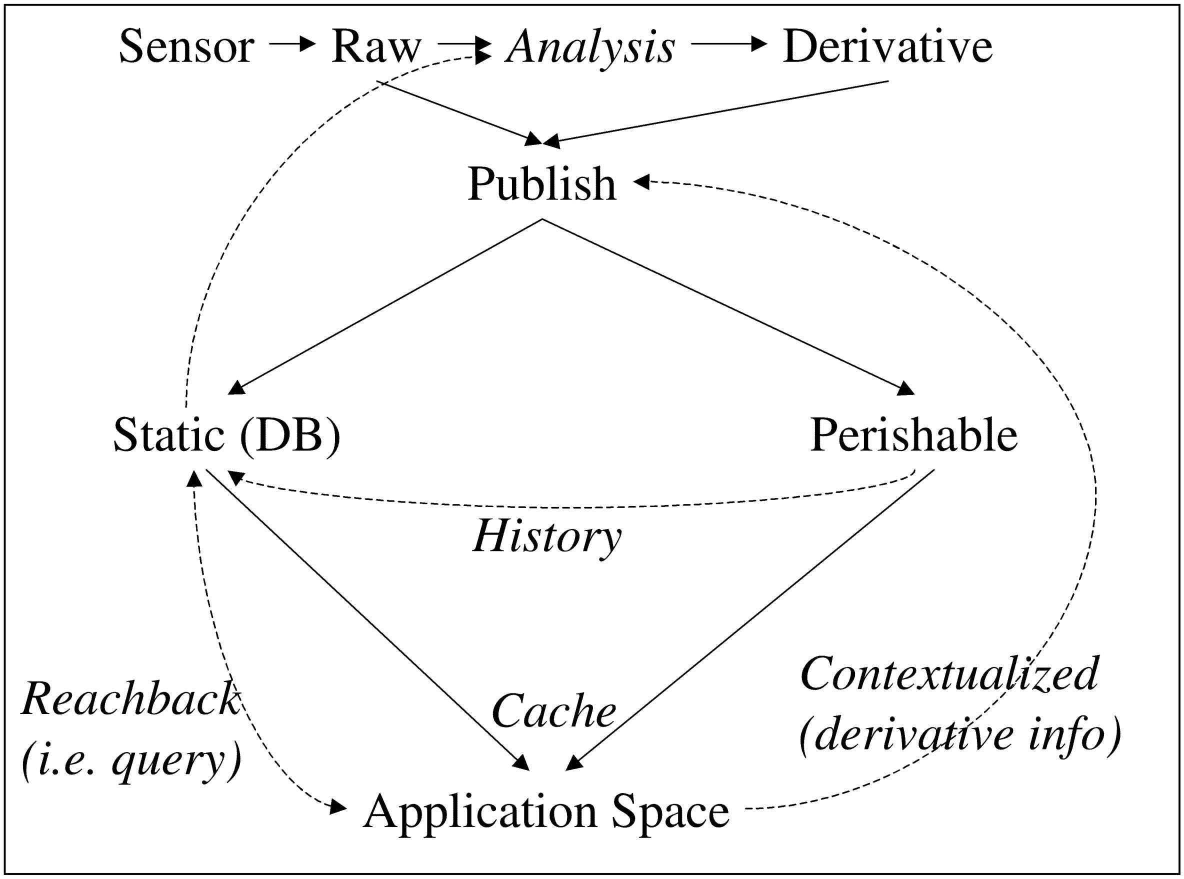


Figure 2-4 Context-Awareness Information Usage Model.

As we talk about context-aware computing, we can break the data down into two categories: static data and dynamic data. This information model shows how the data relate to one another and how they get used. “An organization’s success in a competitive environment depends critically on its ability to do a better job of assimilating information, increasing its epistemic quality to generate strategic power, and reducing decision cycle times.” [Hall-2003]

If we start with an arbitrary sensor, it may well publish its raw sensor readings. It may also reach into some database (potentially consisting of only a single datum), whether local or remote, to provide some analysis of that information. One could easily imagine that a diabetes blood sampling device might sample some blood, publish the results, check its 'database' of what constitutes healthy or low blood sugar or high insulin, and publish a second report indicating a healthy level. This analysis is a first tier contextualized report.

The need for multiple reports is awkward, but models real world behaviors. For example, in the defense community, it has been recognized that if the first entity to get the data, say some intelligence agency, performs the analysis first and later only reports their findings, then two losses are incurred. First the raw data is typically not available to other consumers that might wish to perform their own analyses to determine their own, individualized contextual-relationships. Second, a bottleneck and delay are incurred waiting for the analysis that reduces the speed of any decision cycle.

Applications with low data rates and good connectivity may choose to perform reach-back / query to determine response. Higher data rate information (typically perishable information such as a person's position) assessed with respect to a static background, results in a preferred solution of downloading static information. For example, one solution to location based services, say, within a shopping mall, is to 'pre-cache' all mall information at the end user's device [CCR-2003].

Continuously reported sensor data will enable better decision-making. In particular, today, we rely on events. However, events are nothing more than "significant" points, which may have been "arbitrarily" selected, that we choose to report (e.g., green, yellow, or red stoplights). By providing a continuous stream of sensor data, with finer granularity and more detail, the impending event can be predicted and allow for smoother transitions and better decisions.

2.1.7 Scenario Summary

We recognize a basic desire to detect information relationships across a diverse set of information kinds that are global in scope, much of which is available today – the rest could be available soon. The data relationships examined concern primarily repeating, uniquely identifiable content (e.g., Bob's location) the relationships tend to be personalized (i.e., only of interest to the entity specifying the relationship); and the relationships require the evaluation of multiple variables with respect to one another, generated by multiple publishers, and generated in real-time (e.g., with update rates on the order of 1/10 Hz to 1Hz).

To constantly be aware of all the available information of interest is not humanly possible today due to both human and technical limitations. Yet, we desire all this information and more as well as mechanisms to reduce it to easily digestible quantities. This is all a part of being more efficient with our single most precious commodity, time. The basic challenge then is to bring available technologies to bear in a way that can achieve these goals and develop new capabilities to overcome remaining technical challenges.

2.2 Technical Basis

There are two technology ideas that permeate the prior scenarios and thus form the basis for our approach. The first, *ubiquitous computing*, derives from the information generation and distribution aspects with the idea that we can freely plug-in more information providers and consumers into a public infrastructure with devices that are always-on and probably wirelessly connected and mobile. The second, *context-aware computing*, is about converting the reams of raw data into information and knowledge, with the idea of getting the right information to the right person at the right time, to enable them to make better decisions faster. In concept, the convergence of these two ideas yields Internet-scale context-awareness.

2.2.1 Ubiquitous Computing

We adopt, embrace, and come to depend on our technologies. Our expectations co-evolve with our technical abilities. This is most noticeable as we technologically enable and instantiate the ideas from Mark Weiser. Weiser claimed, "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." [Weiser-91] Although Weiser's ideas for ubiquitous computing focus on the user interface, the hardware infrastructure and ability to communicate information must first become ubiquitous. They are becoming more so every day - the trends are clear.

Of the 7 billion microprocessors sold in 2001, only 120 million (less than 2%) were intended for PCs [Pinto-2002]. It is estimated that in five years, the number of processors in the average home could grow from 40 to 280 and the number of embedded chips used to support increasingly intelligent devices could grow to over 9 billion. [Pinto-2002] We can see the demand that has driven Moore's Law, that the "complexity for minimum component cost has increased at roughly a factor of two per year." [Moore-65] Moore's Law was later reformulated and popularized as doubling the number of transistors on integrated circuits every eighteen months. [Tuomi-02, Wikipedia-2004]. One of the major limiting factors to date has been power supply. But, with miniaturization power demands are being reduced.

> *Some new sensors are getting so small–some are invisible to the naked eye–that they will be able to run on 100 microwatts. (A microwatt is a millionth of a watt. A Pentium 4 processor runs at 75 watts.) At the 100-microwatt level they could gather energy from ambient heat and photovoltaic cells, says Stephen Senturia, a specialist in microsystems at MIT. His colleagues are working on making chips so small that they can power themselves, like watches that need only the kinetic energy generated by movements of the wearer's wrist.* [Fulford-2002]

Our ability to communicate between these devices in volume (using either wired or wireless connectivity) is then generalized as bandwidth. Similar demands for communications have yielded Gilder's Law, which asserts that bandwidth grows at least three times faster than computer power. Such growth, especially in the wireless sector can be evidenced by the ubiquity of cell phones.

"In 1994, 16 million Americans subscribed to cellular phone services." [Gaudin-2001] As of April 2004, the number of cell phones in the United States exceeds 162 million (approx 55% of the 294m population) [Rosenthal-2004]. In Japan, at the end of 2003, there were nearly 79.5 million subscribers (approx 62% of the 128m population) [PHS-2004]. A recent report indicates cell phone use in China is about 286 million (approx 23% of the 1.2b population). [BWO-2004, iicsc-2004] "Some experts predict that worldwide subscribership will reach 1.2 billion people by 2005 [Gaudin-2001]. Current world population is nearing 6.4 billion. There is corroborating evidence in a more recent study by the Gartner group showing "more than 300 million handsets were sold [worldwide] between January and June [of 2004], leading to a full-year sales forecast of at least 620 million units, compared with 520 million in 2003." [AFP-2004] Assuming that the standard two to three year average meaningful lifespan for computer-based components also applies to cell phones, then the 1.2 billion estimate seems right on target.

> *Cell phones have become part of our digital lifestyle, no longer reserved for only the wealthy or business travelers; at the present it is more unusual to not have a cell phone. I even know some people who no longer have a home phone, opting to go strictly with their cell.* [Rosenthal-2004]

In places, such as Japan, China, and places without cheap or free local calling, cell phones are eclipsing the use of landline phones. "Currently there are more mobile phones than there are fixed phones in China. The country now has 259.6 million mobile phone subscribers compared to 255.1 million landlines." [CD-2003] The choice is as much about economics as it is about enabling or enhancing one's social environment, of meeting one's social environment's connectivity and communications needs.

> *For Japanese youth, leaving home without their keitai (cell phone) or letting the battery go dead, is just about the worst thing that can happen. Without a cell phone to spread news, gossip or arrange meeting points on the fly, social life is impossible. "To not have a keitai is to be walking blind, disconnected from just-in-time information on where and when you are in the social networks of time and place." "The changing dynamics of meeting-making are only the tip of the iceberg in the changes that mobile media bring to how we coordinate, communicate, and share information."* [Textually-2004a]

Clearly, cellular phones have begun to weave themselves into the fabric of everyday life and are changing the way we plan our days (or fail to plan – relying instead on just-in-time information).

In part, the usefulness of cell phones can be related to the usefulness of the network. Robert Metcalfe, inventor of the Ethernet[15] and founder of 3COM, is credited with first

[15] Metcalfe discovered that using queuing theory techniques he could take Norman Abramson's AlohaNet packet radio system concept from a maximum capacity of approximately 17% to 90%. This is the Ethernet [Gilder-1993].

formulating the statement that the value of a communication system grows approximately as the square of the number of users of the system (N^2) as a means of explaining many of the network effects.[16] [FD-2004a, Gilder-1993] Metcalfe's Law is based on the idea of a fully connected graph and the possible number of pair-wise connections N(N-1) / 2.

Like cell phones, the Internet can boast about one billion users, with approximately 200 million from the United States alone. It is estimated that there will be an increase of over 16% in 2005. This, in part, leads to the claim that the Internet has become indispensable. "*The idea is that the Internet has become so embedded in the daily fabric of people's lives that they simply cannot live without it*." [HNV-2004]

> *What we are seeing right now is a convergence of exponentials: Moore's Law meets Gilder's Law meets Metcalfe's Law. The result is more data, more users, more access devices, and more services…*
> [Papadopoulous-2004b]

> *With steady reductions in the price of processing power and memory, intelligence will continue to penetrate and populate virtually every product. Advances in wireless technology will allow low-cost, high-speed connections for hand-held devices, as well conventional appliances (washing-machines, refrigerators, etc.) to the Internet. So in the next few years almost everything will become an intelligent, connected "appliance".*
> [Pinto-2004c]

The trends that lead to this pervasive Internet form along technological and social lines that are in tension, to some extent. The technology-driven pattern of "smaller, better, faster, and connected" will yield billions of Internet-enabled microprocessors that will "*provide digital intelligence and connectivity for almost every commercial and industrial product and appliance, extending the Internet into most aspects of our lives*."

[16] The law is often illustrated with the example of a fax machine: A single fax machine is useless, but the value of every fax machine increases with the total number of fax machines in the network, because the total number of people with whom you may send and receive documents increases [FD-2004a].

[Pinto-2002] A vast assortment of sensors, devices, and software will generate continuous, real-time[17] data streams that will then be reported or accessible over the Internet.

> *With advances in digital technology, it's becoming increasingly practical to provide virtually any sensor with a wired or wireless connection that enables remote access to the devices' control inputs and data outputs. Through a variety of location technologies (e.g., GPS, Cell-ID, and Cell-ID with triangulation), both fixed and mobile sensing devices can report their geographic location along with the data they have collected.* [Botts-2003]

The market-driven, patterns of "business needs" will focus on adopting only technology that solves problems and somehow is perceived to provide a competitive advantage, while "social wants" will adopt that which makes life easier or supports social endeavors. This will place pressure on the technological gains, especially in the area of information flow and management. It does no good to have everything connected if we cannot rapidly make use of the information or control the devices.

2.2.2 Context-Aware Computing

The amount of information that an average business must process doubles every nine months [NH-2004] while the information on the Internet doubles every twelve months [Berger-2004]. The planned introduction of RFID tags throughout Wal-Mart and its top 100 suppliers [Liard-2003a, Liard-2003b] will keep this information growth on track. It also has a side effect of people discovering the ability to share more information, resulting in a desire to do so, keeping the cycle growing.

[17] "Real-time" is use throughout this dissertation to match with layman parlance meaning "quickly in the best case" even though the Department of Defense term "near real-time" would be more appropriate. However, "real-time" is usually associated with embedded, deterministic computing.

> *…information, especially on the Net, is not only abundant, but overflowing. We are drowning in the stuff, and yet more and more comes at us daily. That is why terms like 'information glut' have become commonplace, after all.* [Goldhaber-04].

Our drive toward efficiency results in expunging anything that cannot be immediately applied using the tools at hand. Often we reduce the update or query interval of useful information because the information load is overwhelming. In general, we dispose of all distractions (e.g., information) that are not immediately perceived as relevant to the current task.

An approach to a solution that does not ignore information lies in understanding the wisdom, "*Everyone is this world has some kind of burden. It's not the burden that matters – but how you carry it!*" [Unknown] It is the goal then, of context-aware computing, to assist in organizing the information burden to reduce the cognitive load.

Although Weiser does not appear to speak directly to context-awareness, when we take a more expansive view, his "technology" reference can be seen to include the essential ideas that give rise to the technologies; they are as important as the technologies themselves. This is clearly evidenced in our present focus on context-aware computing. Context-awareness is less a novel idea than it is the incorporation of artificial intelligence precepts into the fabric of our daily lives as system developers. In a 1976 speech Allen Newell prophetically spells out the ideals of context-aware computing.

> *Exactly what the computer provides is the ability not to be rigid and unthinking but, rather, to behave conditionally. That is what it means to apply knowledge to action: it means to let the action taken reflect knowledge of the situation, to be sometimes this way, sometimes that, as appropriate. [Newell-76]*

Evidence that this idea has permeated our computing lives can be seen in application features such as automatic spelling correction or the helpful “paper-clip” in Microsoft Word [Microsoft-2005], automatic detection of columns when a textual document is converted into a spreadsheet by Microsoft Excel [Microsoft-2005] as well as in applications intentionally categorized as context-aware, such as CybreGuide [AAHLKP-1997] or GUIDE [CDMP-2000].

2.2.2.1 Context-Awareness Definitions

Context-awareness is about the relationships among data; that context-aware computing takes data from a highly dynamic environment and synthesizes it into information and knowledge for decision-making. “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.” [Weiser-91] The objective is to reduce the amount of available data to only that which is pertinent. But, if one objective is to prune useless information, how do we determine what is useful? The following two context-awareness definitions are insightful.

> *Any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and application themselves.* [*DAS-2001*]
>
> *The set of environmental states and settings that either determines an application’s behavior or in which an application event occurs and is interesting to the user.* [*CK-2000*]

There are three significant points to observe in these definitions. First, when taken together, they essentially say that the information relevant to a context-aware computation may originate from ***anywhere*** and may be ***anything***. Information of interest may come directly from the device or user; it may be sensed from the physical environment; it may be derived from the computational environment; or it may be pulled from some other information environment. The physical, computational, and information environments need not be immediate and local but could well represent the activities in some process on the other side of the globe in which the user is interested. This recognition feeds directly to the pervasive Internet – to make better decisions, we will want more information – thus a requirement for more information generators as well as for effective distribution mechanisms.

Second, there is a focus on ***task***. Information is only relevant in the context of a given task and associated user (if any). Although significant overlap exists for any two people performing the 'same' task, there are differences that make each instance of a task individualized. For example, consider two people going shopping. Each has their own, unique grocery list. The store they use is different in name, layout, produce quality, prices, specials, etc. Each user has their own brand preference and willingness to pay extra to get "the best" or else to buy generic. Thus, we observe that while there may be a reuse of any task template, the information necessary to accomplish the task becomes individualized.

Third, the second definition recognizes that a user, a person, ***need not be present*** or directly involved. Thus context-aware computing can exist in a completely autonomous computing environment, say among smart agents, but in a way that is infinitely recursive – giving rise to smart agents to the smart agents or other context-aware middleware.

2.2.2.2 Context-Awareness Generations

Although we have these definitions of context-awareness and we have several surveys of the field, [CK-2000, Tarasewich-2003] none have yet described the evolutionary stages that have occurred. In considering ISCA, we are able to recognize that there are three distinct generations of context-aware computing that exist and that few have directly considered the third generation wherein ISCA falls.

2.2.2.2.1 1st Generation – Device Awareness

The first generation of context-aware computing focused on awareness about the device. This work goes back at least as far as the original screen savers, where the objective was to avoid burn-in on the phosphorescence. Follow-on work focused on preserving battery power (it being the most precious commodity in a disconnected mode). The top power management techniques address each independently controllable component, such as CPU, fan, and peripherals. For example, the screen is automatically dimmed while active on battery power, the backlight is used only when necessary, lower stale-out values after which time the screen goes to sleep; the hard-drive spins down when not in use, the fan is thermostat driven, and sometimes even reducing the clocking speed for the CPU. Newer mobile devices even provide direct control over all peripheral power consumption [Symbol-2004]

In all these cases, one single, independent, dynamic variable drives the responses. If hot, then fan-on. If inactive, then go into idle mode. In these cases, the predicate is simple, measured in terms of a single variable against a predefined constant. For example, “temperature < tempLimit” or “timeSinceLastActivity < sleepLimit” where the temperature limit is set at the factory and the sleep limit has a factory default, but can be overridden by the user.

Although we describe this as first generation behavior, it is still useful today.

2.2.2.2.2 2nd Generation – Sensor data applied against static backdrop

The second generation of context-aware computing moved away from the device and began to focus on awareness about the user's environment. However, these capabilities are still concerned with the application of single variable data against a static backdrop.

Early location based services work considers questions along the lines of should a person's cell phone be automatically set to vibrate if they are known to be in a theater. It is recognized that a person's availability differs if they are at home or at work. In these cases, location is used as a proxy for task. Location based context-awareness experiments, such as CybreGuide [AAHLKP-1997] or GUIDE [CDMP-2000]., took the user location as the determinant in displaying maps of the user on campus, as shown in Figure 2-5, in a mall, on a tour [GBBTBJS-2002, GBBT-2003]. Along similar lines, the idea that advertisers might send coupons to a shopper based on their location in a mall [CCR-2003].

Similarly, web portal customization, as found on sites like myYahoo! or my94.1 takes the user identity and applies user profile information to customize the website accordingly.

These are essentially problems with a single independent variable – one's location

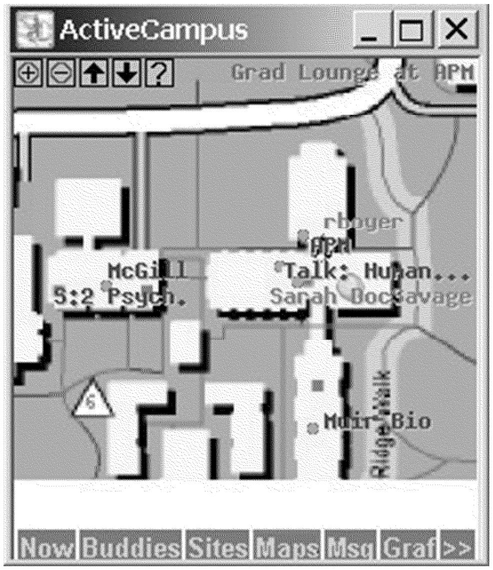


Figure 2-5 Static Background Information.

ActiveCampus users are located with respect to static background information

with respect to a static information background. Although there may be dynamic updates to some of the background data, the updates are sufficiently infrequent, on average less than once per day, such that they can effectively be treated as static information. There are efficient solutions supporting such location-based services, providing the comparison of one's location is against a collection of stationary areas of interest [CCR-2003]. These still tend to be domain-specific and are less useful for general-purpose context awareness.

Efficient solutions for spatial location awareness in such environments were developed by IBM China – where the database of information relative to a given area was pushed to the edge-device where the GPS information was gathered to avoid communication costs.

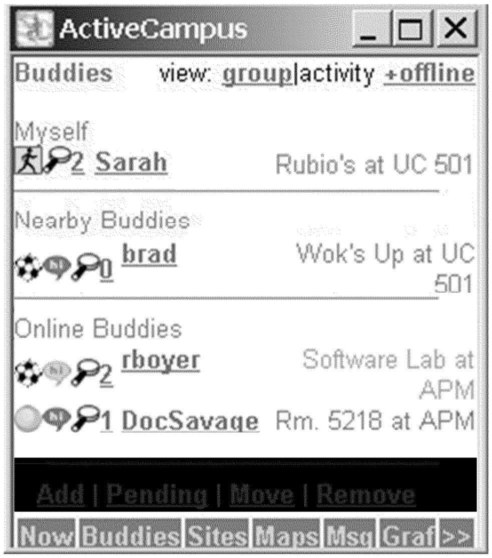
Extending these problems by referring to a persons' specified interests may require more data, but are still essentially single independent variable problems in so much as the profile information tends to be static (and in fact is dependent on the person in question).

We claim any dynamic problem with low update rates to all but one variable can be recast in a similar light.

2.2.2.2.3 3rd Generation – Multiple Publishers of Real-Time Dynamic Data

The third generation of context-aware computing shows interest in multiple data as independent variables. Buddy proximity is the proto-typical example as shown in Figure 2-6 from the ActiveCampus project [GBBTBJS-2002, GBBT-2003].

While it is recognized that context comes from the relationship between pieces of data, prior work does not call out the significance or increase in complexity based on the number of independent variables. However, a significant class of problems occurs when the context of interest depends on multiple dynamic variables as described in the motivating scenarios in Section 2.1.



ActiveCampus

Buddies view: group|activity +offline

Myself

2 Sarah Rubio's at UC 501

Nearby Buddies

0 brad Wok's Up at UC 501

Online Buddies

2 rboyer Software Lab at APM

1 DocSavage Rm. 5218 at APM

Add | Pending | Move | Remove

Now Buddies Sites Maps Msg Graf >>

Figure 2-6 Dynamic Data Associations.

ActiveCampus users locations are evaluated with respect to that of buddies.

2.2.2.2.4 Context-Awareness Summary

Through context-awareness techniques, we gain the efficiency of just-in-time action. By getting the right information to the right person (or control agency) at the right time we reduce waste. For example, we need not have a thermostat cooling cycle set for a specific time, but rather it could be set to begin cooling a few minutes before we arrive home which preserves energy in the case where we get caught in traffic or decide to run errands on the way home. In addition, this also allows us to come home early and experience the same comfort. Context-aware computing techniques are the answer for eliminating the excess.

2.2.3 Convergence: Internet-Scale Context-Awareness

Together, these ideas and their associated evolving technologies are converging to yield an environment – an Internet experience – that is different in kind. In short, it will be a *ubiquitous, real-time, highly-mobile, always-on, event-based, content-focused, individualized, context-aware* computing environment that we are coming to call the pervasive Internet.

2.3 Technical Approach

Content-based publish / subscribe (CBPS) appears to be an infrastructure that fulfills these needs. A CBPS infrastructure provides a distributed network of applications, *event-brokers*, in what is known as an overlay network. This overlay network then serves as the infrastructure to which one can independently connect information providers, known as *publishers*, and information consumers, known as *subscribers*. The infrastructure wires together publishers and subscribers based on information content. A subscriber is only presented information that matches the subscriber’s request, thus yielding a first-order contextual filtering of information. A high-level overview of the mechanics is provided in the following subsections.

2.3.1 Basic CBPS Mechanics

Although a CBPS system is stateless and intentionally order independent, we can describe it as having three logical steps in the operation of a CBPS network. First, the network is set up. Next, routing paths between publishers and subscribers are organized. And finally, information is pumped through the network.

2.3.2 Network Setup

A CBPS system is composed of event-brokers, publishers, and subscribers. For simplicity, the overlay network of event-brokers will be discussed as a star-diagram, where the center hub can be considered a root node resulting in an acyclic network.[18] Event-brokers are setup with some root node in accordance with an underlying physical layout of the Internet nodes. New event-brokers are provided with some knowledge about how to find and connect to their parent node. Publishers connect to the nearest event-broker, called an *edge* (or border) event-broker.[19] Similarly, subscribers also connect to the nearest (edge) event-broker.

2.3.3 Routing Path Setup

The routing path between a publisher and a subscriber is based solely on matching content availability and interest. First, a publisher will *advertise*, using an *advertisement*, the *event-types* that it will produce.[20] An advertisement is a conjunction of attribute constraints, effectively forming an N-dimensional bounding box in which later event

[18] There is a great deal of ongoing research considering peer-to-peer and other overlay architectures.

[19] We choose to use the word "edge" in that such an event-broker represents the edge of the network.

[20] Some researchers are experimenting doing without this step and instead flood forwarding subscriptions. We examine the tradeoffs between these two approaches in Section 3.

notifications must fall.[21] This advertisement is passed to the publisher's (edge) event-broker. The event-broker stores the advertisement and then forwards it to all neighboring event-brokers, who do the same, except they do not pass it back over the connection on which it was received. The result is a flood forwarding of the advertisement to all event-brokers.

A subscriber will *subscribe*, using a *subscription*, for information of interest. A subscription is also a conjunction of attribute constraints, effectively forming an N-dimensional bounding box in which later event notifications must fall. The subscription is passed to the subscriber's (edge) event-broker. The event-broker stores the subscription and then forwards it over all connections (except the one on which it arrived) on which advertisements were received that *match* the subscription. This is repeated by each event-broker, thus forming chains of subscriptions that lead from publisher-edge event-brokers to subscribers interested in that data.

2.3.3.1 Advertisement to Subscription Matching

Match, in this case, means that all fields of the subscription exist in the advertisement and for each matching field; there is an intersection of the content constraints. For example, an advertisement claiming the production of X and Y data, with $0 < X < 10$ and $0 < Y < 5$ would match a subscription requesting information containing X and Y data where $1 < X < 2$ and $4 < Y < 10$. Similarly, the same advertisement would match a subscription requesting information about X where

[21] The bounding box merely describes where the base-class of information must lie. Additional, information (i.e., fields) may be contained in the actual notification.

$-5 < X < 5$. The advertisement would not match a subscription requiring information about Z nor one requiring $50 < X < 100$.

2.3.3.2 Advertisement to Subscription Order Independence

By storing the advertisement and the subscription on arrival, the system is able to be order independent. That is, a publisher can declare new event-types either before or after a subscriber has made a request for matching information and the system will create the appropriate routing. When an advertisement arrives that matches a prior subscription, then the earlier subscription is forwarded over the connection on which that advertisement is received, as if the subscription had been newly received.

Similarly, a new event-broker may be connected to a running network at any time. When an event-broker connects, existing advertisements are immediately forwarded along the new connection. Additionally, separate networks evolved in isolation can be combined during runtime.

2.3.4 Information Flow

After the routing path between a publisher and a subscriber has been created, each notification from the publisher will be passed to its (edge) event-broker. Unlike advertisements and subscriptions, neither the notification nor any associated state information is stored.[22] The event-broker finds all connections over which a subscription *matching* the notification was received. This is repeated by each event-broker, thus passing the notification from the publisher to the subscriber interested in that data.

Here, *matching* means that all fields of the subscription exist in the notification and for each matching field, the notification content is contained within the bounding box

[22] Some research is experimenting with storing event notifications for a (un)limited duration.

specified by the subscription. For example, a notification producing X=1.5 and Y=4.5 would match a subscription requesting information containing X and Y data where $1 < X < 2$ and $4 < Y < 10$. Similarly, the same advertisement would match a subscription requesting information about X where $-5 < X < 5$. The notification would not match a subscription requiring information about Z nor one requiring $50 < X < 100$.

2.3.5 Benefits of CBPS

It is well understood that event-based publish / subscribe systems provide a separation of concerns by decoupling publishers and subscribers so that they can remain unaware of one another. [CABB-2004] Thus, we can readily plug-in new publishers as new sensors are deployed. Similarly we can readily plug-in new subscribers as new information needs are identified. And, the overlay network can be readily extended with additional event-brokers at any time.

One interesting feature of content-based publish / subscribe is that the subscriber is in control. Information, in the form of notifications, only flows through the system where there is specific interest in the content. This results in the subscriber receiving only contextually relevant information. This is in contrast to a channel-based publish / subscribe system,[23] where the publishers identify the information pipe down which they will provide their information and a consumer must accept (be able to process) all information coming down that channel independent of relevance to the consumer.[24]

For example let's say a subscriber is only interested in Qualcomm stock when its price dips below \$100. In a channel-based system, the channel might be identified as

[23] Also known as a topic-based or a group-based publish / subscribe system

[24] Some systems do provide limited filtering at the client-end to minimize extraneous information flow.

"stocks" and all subscribers would be required to receive all information provided for all stocks. And, in a content-based system, the data stream is filtered at each leg of the journey through the overlay network, eliminating quotes where no subscriber has indicated interest, whether in a particular stock or in a threshold for its value.

2.4 Technical Challenges

Fostering an environment of information flow to achieve Internet-scale context-awareness faces a variety of challenges. In part, this is because an inherent tension exists between the ideas of ubiquitous and context-aware computing. Context-aware computing seeks to reduce the amount of information of which a user must be aware. Yet, as we evolve into a pervasive Internet, we are increasing the amount of data available, doubling it about every 9 months [NH-2004], and placing a substantial burden on context-aware capabilities that extend from the edge of the network deep into its core. The system demands can be broadly categorized as information relationships, information access issues, personalization issues, and scalability issues.

2.4.1 Information Relationships

The most distinguishing characteristic of Internet-scale context-awareness is that it considers not simply raw information, but dynamic information with respect to other dynamic information where the information originates from multiple publishers. There are two challenges here. First, we must find a way to relate information originating from the multiple publishers. Second, the likelihood of satisfying a relationship is often minute when compared with the number of raw information events.

Assume for a moment that we have two information producers, say X and Y. Each produces data at 1Hz, with a uniform distribution along one dimension, x and y, respectively. Let each dimension range from 1 to 100. Let us say then that the relationship of interest is x=1 and y=1. Independently, the chance of X producing x=1 is 0.01, and the chance of Y producing y=1 is 0.01. However, the likelihood of our receiving a combination of events matching the desired relationship criteria is 0.0001. From the perspective of location-based services, grid a square mile into 30 feet by 30 feet cells (to match the precision of current location-based reporting techniques). The chance of being within any cell is approximately 0.00003228. Now consider the chance of a given pair of people coming into the same cell.[25] The ability to efficiently sift through the chaff to find the valuable information is critical.

2.4.2 Information Access

The information access challenge starts with the general availability of the infrastructure. In order to fulfill the ideas of ubiquitous computing and freely move information we must have a world-wide pervasive internet. It is a given that we are not going to build our own, but instead will use the public infrastructure of the Internet.

The Internet allows us to readily connect additional computational nodes. We need such capabilities in order to freely plug-in more information producers or consumers. However, the Internet limits the ability to route information based on IP addresses or URLs. Requiring the human to know the Internet "name" of a piece of information is not feasible and can be seen as one of the key contributing factors to the success of search engines such as Google. Unlike the typical search engine that deals almost exclusively

[25] Human movement is not random, physical constructs (e.g., buildings and pathways), and temporal constraints (e.g., class schedules) all work to reduce randomness.

with static content, we need to support dynamic information content.

We need a communications layer that will allow us to identify and route information by content. This then would ensure that the environment will support the ad hoc introduction of more producers and consumers in a way that does not require specific knowledge of producer hardware identity or location. CBPS does this, although it does not specifically address ontology or naming problems.

A common approach to content dissemination is to create ontologies. However, the construction of an appropriate ontology is problematic as data does not always follow nice rigid organizations imposed by an ontology. Multiple inheritance is not uncommon when the convergence of capabilities ends up grouping together disparate information. From the human factors standpoint, too much thought would be required of an average user who wants to simply "plug-in" more data. We need mechanisms that can self-organize or at least handle the ad hoc introduction of information "in the wild."

2.4.3 Personalization

The individualized nature of the information relationships over recurring reporting of perishable data creates a collection of issues.

1. We do not want to require the requestor to have to consume all the component elements, most of will be useless, to detect the relationship existence.
2. We do not want to centralize all computations
3. We do not want to have servers be required to constantly re-compute individual results (we discovered quickly in ActiveCampus that this would not scale)
4. We do not want servers to have to deal with each person's individual concerns
5. We do not want users to have to suck down all of a website's information

2.4.4 Scalability

The broad spectrum of use supported at an Internet-scale immediately imposes scalability concerns. These include the generality of the solution and the number of participants, message types, and message instances.

2.4.4.1 Generality

Perhaps best exemplified by the commodities broker scenario is the case that the data relationships need not be applied to homogenous data types, but may include a collection of different types.

A second case is that the original data producer may have in mind what the intended use of the data will be, but later, others will come along and attempt to use the data in entirely new ways. This requires an openness as no a priori knowledge about the many uses of a given sensor's data can be expected.

2.4.4.2 Participant Quantity

Metcalfe's Law describes the value of the network as being more than just the sum of its parts. The ubiquitous nature of the solution we desire requires the same. We will need to be able to extend any solution to include more publishers and subscribers and any infrastructures nodes to assist in extending the reach of the overlay network.

2.4.4.3 Message Type Quantity

The number of different message types, representing at some level, the number of different types of sensors, needs to grow without bound just as the ability to provide user-defined types in any high-level language is unlimited. An important point in the language analogy is that a simple ontology is inappropriate. The content of interest is more akin to multiple-inheritance.

2.4.4.4 Message Quantity

Three factors control the number of messages passing through the system.

First, the number of information producers and consumers provides a first order approximation to the quantity of messages one can expect to pass through the system.

Second, the general usefulness of a piece of data may determine how many people are interested in that information and thus how many paths through the network it will travel. For example, a sensor as part of a narrowly focused scientific test might provide data of interest only to one scientist while the current activities in a baseball game might be of interest to tens to hundreds of thousands of sports fans.

Third, the recurrence rate of the data involved plays a major factor. For example, a one time event, such as the breach of a nuclear reactor, while important, is only directly responsible for a single event message traversing the network. However, reporting location at a 1Hz rate will result in 3600 events each hour of every day of the year for every tracked entity.

Consider the proximity examples taken to our pervasive Internet environment. We have proximity considerations for buddies, workgroups, high-risk criminals, etc. The number of participants can reach the number of Internet users (over one billion), for each of publisher and consumer, and each person belongs to several data relationships. The significance of such large quantities of recurring data can be envisioned when one knows that, high-powered military applications struggle to achieve situational awareness with as few as 10,000 entities with updates limited to once every twelve seconds.

2.5 Summary

The ubiquitous computing movement demonstrates a desire to be able to send and receive any information, at any time, from anywhere. In conjunction with other factors, this is dramatically increasing the wealth of information available to us in real-time. To reduce our cognitive load and increase efficiency in general, context-aware computing can synthesize data from a dynamic environment into information and knowledge for decision-making. This movement demonstrates a clear desire for not just any information any time, but the right information at the right time. Together, these form the desire for Internet-scale context-awareness.

However, there are many challenges facing this Internet-scale context-awareness idea. We must be able to efficiently detect the desired information relationships across multiple publishers, publishing at high data rates in a constantly expanding environment; we must have information access; we must be able to handle the individualized nature of these relationships; and we must have an infrastructure that scales to Internet proportions.

3.0 Background and Related Work

The diversity of the motivating scenarios discussed in Chapter 2 shows the need for a general purpose solution. Any viable solution calls for four core components / capabilities. First, a clear need exists for context-awareness techniques – in particular data relationships in real-time – to reduce the human cognitive load – to get the right information to the right person at the right time. Second, it requires the ability to easily plug into the pervasive Internet environment and be scalable in all dimensions. Third, the user must be able to find the raw information of interest. Fourth, to minimize infrastructure and bandwidth costs and to enable the lightweight edge-devices of our future to be effective clients, the user must be able to express the relationships of interest in a meaningful way such that the infrastructure performs the filtering and the user need not drink from the fire hose of information.

Potential solutions to our ISCA desires arise in both context-awareness research and in publish / subscribe research. Each of these areas has an active research community. However, only a few of the most relevant research efforts will be discussed here to provide a useful background and examine the strengths and weaknesses with respect to our ISCA needs. [26] Through this process, we detail how our needs for efficient multi-publisher, dynamic data, attribute-to-attribute comparisons across event boundaries at Internet scales remain unmet.

[26] Additional related threads may be found in Sensor Networks, Streaming Databases, Agent Systems, and Active Networks research. Although related, they are too far removed and hence are omitted.

3.1 Context-Toolkit

One of the most significant pieces of early research is the Context-Toolkit, whose primary purpose is to make it easy to prototype context-aware applications [Dey-2000, DAS-2001]. This toolkit focuses on the separation of concerns between sensors, application needs, and actuators.

It offers a small set of generic 'base' classes from which an application developer can derive application specific classes. These classes are organized around high-level context-aware application functions: collecting sensor data, combining data from multiple sensors, and translating sensor data into alternate formats. Each of these classes contains the entire infrastructure required for distributed peer-to-peer storage, communication via XML over HTTP, and software event monitoring.

The *Context Widget* accepts and stores context coming from a sensor and abstracts away the details of the sensor. For example, a location widget might gather time-stamped locations coming from a GPS device. Context is stored in dynamically typed attribute–value pairs. The abstracted context information is tailored to suit the expected needs of consumer applications. To signal the arrival of new data, a context widget provides event notification functionality. Context widgets also provide a query interface that allows a user to request metadata, such as the type of sensor, its resolution and accuracy, or how that data was acquired. Context widgets serve as the base class for all other classes.

The *Context Aggregator* aggregates the context of multiple widgets, and inherits their attribute–value pairs. By listening to events from the widgets it aggregates, an aggregator can stay aware of changes in the context it monitors. In this way, logically related information from multiple sensors can be made available in a single repository, thus reducing the number of connections required for an information consumer.

The *Context Interpreter* can be thought of as raising the level of abstraction (in the eyes of the beholder). It can transform context from one representation into another, such as taking a GPS reading and producing map coordinates. Or, it can take information from multiple context sources and produce new contextual information.

The *Services* widgets are just context widgets that abstract away actions on actuators (e.g., a light switch). They are responsible for controlling and for changing state information in the environment on behalf of applications.

The *Discoverer* widgets maintain a registry to help applications find one of the above resources by name or capability in the network. The capability would require a description of information that the context widget or derived class provides.

3.1.1 Sample Application

For example, the typical Context Toolkit application contains a number of context widgets, a couple of context aggregators that join those widgets into observable entities (e.g., *user*), a few context interpreters for the translation and interpretation of context, and an application object that hooks all these up as shown in Figure 3-1. The application object then interacts with the user objects and context interpreters to produce interactive displays for end users.

Figure 3-1 shows at the bottom two representative ActiveBadge sensors that report their information (badgeId & location) to a location widget. An interpreter widget is then used (i.e., invoked) to convert the badge identity into a user name, thus transforming raw data into information. For this example, we assume location is reported in terms of room numbers. If reported in a more precise way (e.g., (X,Y)), then a location to room number

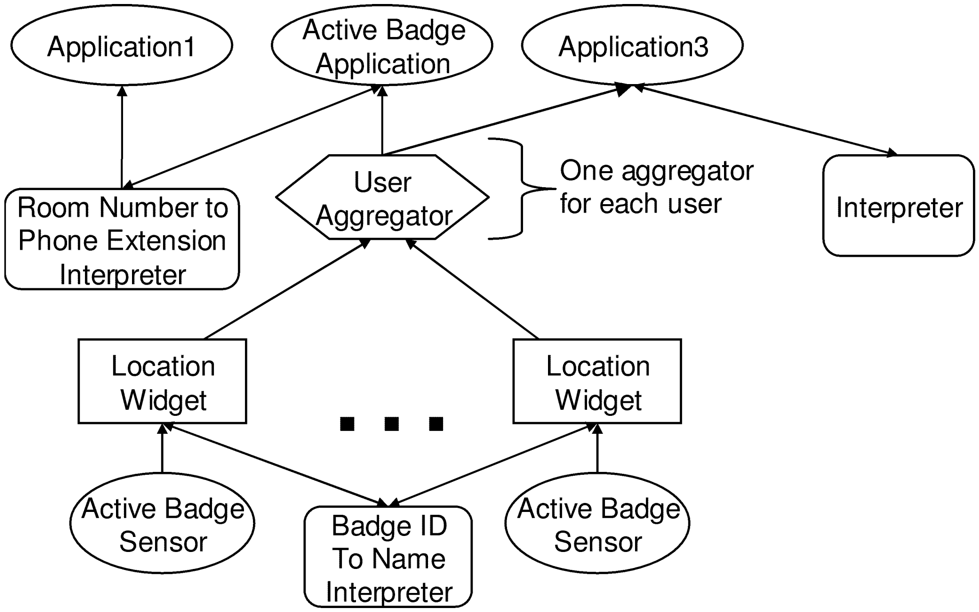


Figure 3-1 ActiveBadge Architecture Diagram.

Architecture diagram for the ActiveBadge application using the Context Toolkit

interpreter could be provided to convert the raw data into information. Then each user's aggregator widget subscribes to the location information of, say, all buddies. This provides the ActiveBadge application a single source from which to pull its data, such as to discover when buddies are near. The ActiveBadge application then uses the room number of ones buddy to determine the telephone extension at which they can be reached. Two additional applications and one extra interpreter are shown to indicate the reuse capabilities

3.1.2 Strengths of the Approach

There are a variety of benefits provided by the Context-Toolkit's approach. The key strength lies in the separation of concerns along multiple dimensions.

First, transparent, distributed communications are provided such that sensors and applications need not concern themselves with communication details. Also, raw or contextual data can be acquired from multiple, distributed network sources.

Second, it separates context acquisition and deployment from application development. An application subscribes to a context widget to get sensor data or publishes to a context widget to effectuate an actuator. These widgets provide customizable and reusable building blocks for context sensing.

Third, it supports separation of context determination. The application need not directly connect to sensor or actuators, but any context widget derived class might be connected to the application, such as an interpreter. In this way, raw data can be evaluated in the network and raised to a higher level (e.g., information or knowledge). Thus application developers can focus on their interests in the high-level context (e.g., that a meeting has started). Abstracting away each functional requirement as a class is useful in accommodating key dimensions of change.

Fourth, the resource discovery mechanism helps applications find and communicate with sensors of interest. It allows us to plug-in new information providers, from sensors to interpreters, as well as to plug-in new information consumers.

Fifth, context widgets run independently and may acquire information even when no user application is currently interested in their data. Thus they can store context, generate histories, establish trends, and predict use.

3.1.3 Drawbacks With Respect To ISCA

As is typical of a *toolkit*, the Context Toolkit stops short of dictating architectural standards that would enable an application based on the Toolkit to interoperate or integrate with applications of complementary functionality [HL-2001]. Indeed, such standards could hinder rapid development. Another concern is that the Toolkit's fine-grained distributed object model entails significant inter-object communication costs and precludes some efficient data organizations, among other issues [HL-2001, Winograd-2001].

The Context-Toolkit suffers from three major issues with respect to the multi-publisher, dynamic data relationships required for ISCA. First, developers on the information production side determine what is "good" for the information consumers. That is, the only context available is that for which a context-widget was previously developed. Additionally, the context-widget developer, as an example, determines where to run the widget.

Second, information flows are essentially one-directional. The aggregation of information is the primary model. Figure 3-1 shows bi-directional flow with the context-interpreters, but this is done only under a function call style usage.

Third, information flows are point-to-point. A separate socket connection is made for each interested party in a sensor's information. Combined with the directional information flow, complex relationships are ill supported and hence inefficient. All information potentially supporting complex relationships must to be received by the information consumer for it to determine the satisfaction of the complex relationship. As we described in Section 2, relationships often have a low probability of satisfaction with respect to the amount of raw sensor data generated.

3.1.4 Summary

The context-toolkit demonstrates the value of separation of concerns along multiple dimensions. As a result application developers can work at higher conceptual levels and focus on contextual data. Unfortunately performance considerations to satisfy ISCA needs are not addressed and not enough information control is afforded the consumer. Despite the challenges identified, we could implement our ISCA requirements using this (or any of the following frameworks) as a basis. We ultimately choose CBPS as the best foundation.

3.2 Context-Fabric

Both the context-toolkit and the context-fabric have the design goal of making it easier to build loosely coupled, sensor-based context-aware applications. Where the context-toolkit focused on separation of concerns and the development of widgets, the context-fabric focuses "more on how the data will be modeled, distributed, protected, and used." [Hong-2002] Through this it provides a ubiquitous, network-accessible, middleware service infrastructure. This infrastructure would include a context layer as middleware to provide a "dialtone" for context-awareness that would be accessible from any device, any application, anywhere, anytime. This middleware would be decomposed into sensors, services, and applications. To support this, Hong is also developing a Context Specification Language to account for the predicates, vocabulary, and queries and events of context-aware applications.

The context layer would be responsible for transforming sensor data into context data. This is where context information would be interpreted and sensor data fused. Shifting away from the precision, granularity, and accuracy of sensors into more humanistic terms (e.g., "in the movie theater") is also an intended part of this transformation.

The research aims to provide two core services. The first service is to be "automatic path creation" where an information pipeline is automatically generated based on the type signatures. In this way a monolithic application would split its computations into many small, composable services and connect them. The second service is to be a proximity based discovery mechanism that would discover nearby sensors so that a developer could use sensors that are not resident on their immediate device.

3.2.1 Sample Service – Automatic Path Creation

Imagine two users want to query the context layer to identify the nearby movie theaters. One uses a cell phone that only provides cell-id for its location while the other provides GPS. Next we consider that agents have been written to convert cell-id to GPS, GPS to zip code, and zip code to movie theaters. In this case, both user queries can be satisfied because a path from the input type to the output type can be automatically generated.

3.2.2 Strengths of the Approach

The context-fabric provides a network infrastructure, a context layer, that does not require manually wiring together the available context, but rather the system automatically identifies the relevant information and can invoke type conversion services as necessary. This puts more control in the hands of the information consumer.

3.2.3 Drawbacks With Respect To ISCA

Data still flows in a single direction, requiring aggregation of all potentially significant elements be received in order for context to be determined. A database approach is taken without consideration of any kind of demand-driven rate limiting.

No communications efficiencies are identified, with perhaps the exception that information is first searched for in a local cache. However, with the dynamic information that we expect in our ISCA environment, the cache will constantly be refreshed.

3.2.4 Summary

The context-fabric allows the information consumer easier access to desired information through its automatic path creation techniques. However, it still does not address performance or scalability.

3.3 Topic-Based Publish / Subscribe

The publish / subscribe world centers on the wide-scale distribution of raw information elements, with context-awareness introduced into the environment as an after-thought. In publish / subscribe systems, publishers publish *events* (or messages) and subscribers subscribe to *events*, with the publish / subscribe middleware doing event matching and routing. Hence, publishers and subscribers do not need to know about each other because the middleware makes information distribution transparent. The publishers and subscribers are wired together based upon some common-interest function that varies with the style of publish / subscribe as detailed below. This enables new publishers, subscribers, event types, and subscriptions to be freely added to the milieu.

Publish / subscribe technologies started with channel-based, aka group-based, aka topic-based publish and subscribe mechanisms. In a topic-based pub / sub system, a *topic* is identified to provide a pipeline over which all information of a certain type will flow (e.g., location data). All producers of location information would feed the data stream and all consumers of location information would pull data from the stream.

The topic is often configured through an interface to the middleware. This often requires an administrator to manage the middleware. Some systems provide mechanisms to dynamically create topics through an application programmer interface. Queues may also be defined that provide only a single instance of the data from which multiple clients pull data.

Generally, publish / subscribe systems are stateless with respect to the data. (Some state is required in order to maintain the bindings between publishers and subscribers.) As an ancillary feature, these systems also provide for data persistence, allowing a subscriber that has gone offline to reconnect without missing any messages. (Other non-relevant capability discussions have been omitted.)

3.3.1 Strengths of the Approach

The topic-based publish / subscribe technique provides for a necessary separation of concerns between information producers, publishers, and information consumers, subscribers. The middleware can achieve an economy of scale through distribution mechanisms like multi-cast to efficiently push information through the network.

3.3.2 Drawbacks With Respect To ISCA

First, the data stream is effectively identified by a name. This coarse granularity requires a subscriber to process all information in the stream even if it is only interested in a small subset of the data. Some systems do allow a basic level of filtering by providing for nested subtopics in the same way one would specify a directory path A/B/C. This filtering inability puts an undue strain on lightweight edge-devices in requiring them to drink from a fire-hose of information. Providing flexible filtering down to the granularity of the data elements would require that one topic be generated for each element of the power set (i.e., size = 2^n) of element tags.

Second, due to the filtering limitations, developers often cause the same data to flow along different streams so that a subscriber interested in only a specific content need only subscribe to one of the streams. A challenge arises when one subscriber listens to multiple streams. For example, a football game between the San Diego Chargers and the Los Angeles Raiders in San Diego might be published on the following topics: SanDiegoEvents, LosAngelesRaidersGames, SanDiegoChargersGames, NFLGames, SportsBrawls, etc. As a result, a sports fan may well subscribe to multiple topics and then would receive two or more messages that are identical. This requires the clients to perform duplicate screening and further complicates their logic. Duplicate screening is generally not performed within the middleware.

3.3.3 Summary

Topic-based publish / subscribe essentially started with performance considerations and is able to achieve efficient communications as well as nice separation of concerns between publishers, subscribers, and the middleware. However, the focus on distribution has been from the publisher side, such that the publisher is actually in control.

For context-aware computing, where each individual's contextual needs differ, we need finer granularity and control over the data; we need to give more control to the subscriber while preserving or improving the simplicity of the publisher side.

3.4 Content-based Publish / Subscribe

Basic content-based publish / subscribe set out to overcome the limitations of topic-based publish / subscribe in two ways. First, all data is individually tagged at the element level, such that a fine-granularity is available for the data. Next, the information channel is made transparent by enabling the subscriber to express their information desires in terms of the content as opposed to the delivery channel. Now, the subscriber asks specifically for the information of interest.

Not only does the middleware provide separation, but also efficiency: it filters out new events against subscriptions at the publisher's event-broker, exploits overlapping subscriptions, and employs multicast-like routing of events to subscribers. Efficient filtering at the publisher's event-broker is achieved by content-based pattern matching against a publisher's event in a series of independent filters (e.g., {(event.x < 10) & (event.y > 30)}). Sequences of events can be similarly pattern-matched [Carzaniga-1998, CRW-2000].

In the current state of the practice, as exemplified by Siena [Carzaniga-1998, CRW-2000], CBPS systems are composed of three components. One, a *publisher* provides *events (messages)*, each in the form of an *attribute-value* tuple sequence *{(name*$_1$*, type*$_1$*, value*$_1$*), (name*$_2$*, type*$_2$*, value*$_2$*),…}*. Two, a *subscriber* requests events of interest by using *subscription filters* in the form of an *attribute-constraint* tuple sequence *{(name*$_1$*, type*$_1$*, operator*$_1$*, value*$_1$*), (name*$_2$*, type*$_2$*, operator*$_2$*, value*$_2$*),...}*, where each *operator* will be a relational operation { =, !=, <, <=, >, >= }. Three, *event-brokers* mediate between publishers and subscribers, providing an application-level overlay network for efficiently matching and routing events, providing independence of publishers and subscribers.

Figures 3-2 through 3-4 depict an overlay network and the key information flows. In the figure, the following representations are used:

- Diamond – publisher
- Hexagon – subscriber
- Circle – event-broker

- Black lines – represent connectivity
- Colored arrows – represent data flow of advertisements, subscriptions, or notifications.

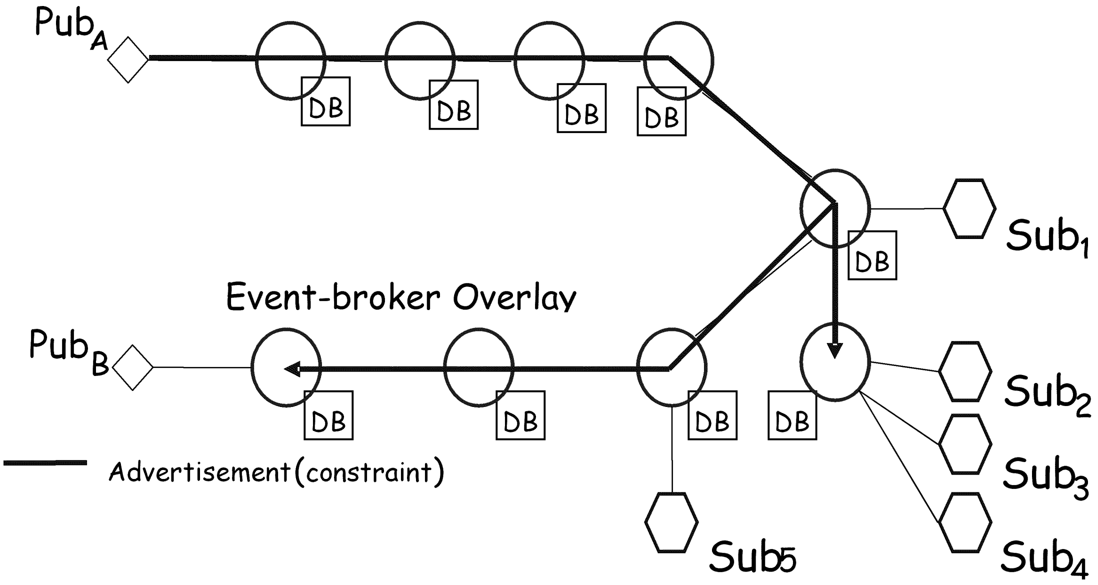


Figure 3-2 Advertisement Flood Forwarding

When a publisher connects to the (overlay) network, it *advertises*[27] its *event-types* with its event-broker. The advertisements specify event-types in terms of attribute-constraints as defined above. The constraints, however, only provide limitations with respect to the base type. That is, an attribute-constraint sequence may include specifications for elements A and B; but, the publisher may produce a derived event type with elements A, B, and C. The only requirement then is for the A and B elements to fulfill the A and B constraints.

Although the diagram depicts the flow of an advertisement through the network, in reality, a copy of the advertisement is deposited with each event-broker along the path. The advertisement is maintained in a local database along with information about the

[27] Some systems omit the use of advertisements, instead choosing to flood forward subscriptions. The consequences of such design decisions are detailed in the Fulcrum section.

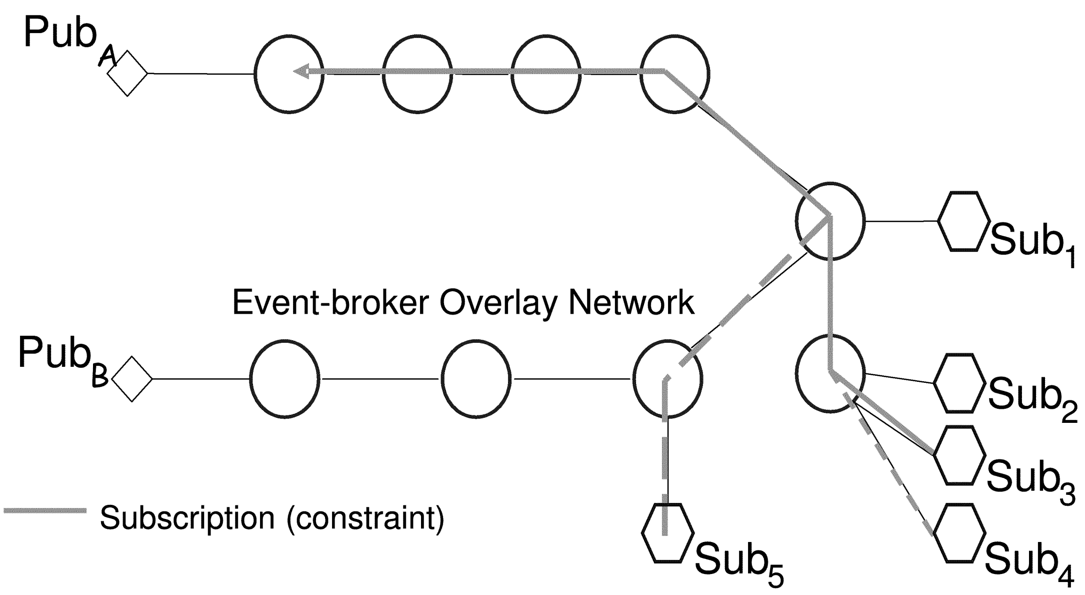


Figure 3-3 Subscriptions Flow Back to Advertisement Originator

connection over which it was received. Later, when a subscription is matched to the advertisement, the connection information is used to forward the subscription.

Subscribers register their interest in events with their event-brokers through content subscription filters. That is, they specify the event types they are willing to receive in terms of attribute-constraints as defined above. The event-brokers route the subscriptions upstream by matching them to the advertisements and following the path(s) to the first event-broker possessing such an advertisement. We call this an edge-broker because it is at an edge of the overlay network.[28]

This will enable the system to quickly suppress information at broker nodes for which no downstream subscriber is currently interested. This eliminates unnecessary network traffic and excess computation at internal event-brokers and at the end-client. Consequently, the client application is, in effect, being pushed into the network via the

[28] Other literature sometimes refers to these edge-brokers as border-brokers [FGKZ-2003, Khurana-2005].

middleware of the CBPS system.

The matching of a subscription to an advertisement occurs when all attributes listed in the subscription are also listed in the advertisement and for these attributes, there is an intersection in the constraints. The result is that the publisher may generate an event notification that matches the subscription.

Like advertisements, a copy of the subscription is deposited with each event-broker along the path. The subscription is maintained in a local database along with information about the connection over which it was received. Later, when an event notification is matched to the subscription, the connection information is used to forward it.

In the case where an event-broker receives subscriptions from multiple publishers, the subscriptions may be checked to see if a more general subscription has already been propagated upstream. In such cases, as indicated by the dashed green lines in Figure 3-3, the new, but subsumed, subscription need not be propagated.

The result is that the brokers set up a "switching fabric" between publishers and subscribers by pre-matching subscription filters to the advertised event types. The switching fabric has two performance benefits for the brokers. One, filters are only applied to events that have a chance of satisfying the filter. Two, it is possible to avoid redundant filtering for overlapping subscriptions, as well as send only one copy of each event between brokers when multiple subscribers share parts of the same pathway from the publisher to an intermediate broker.

Publishers then generate event *notifications*, attribute-value sequences, satisfying the constraints of their advertisements. These are then evaluated against the subscriptions held at each event-broker and propagated out to the subscribers when a match is detected.

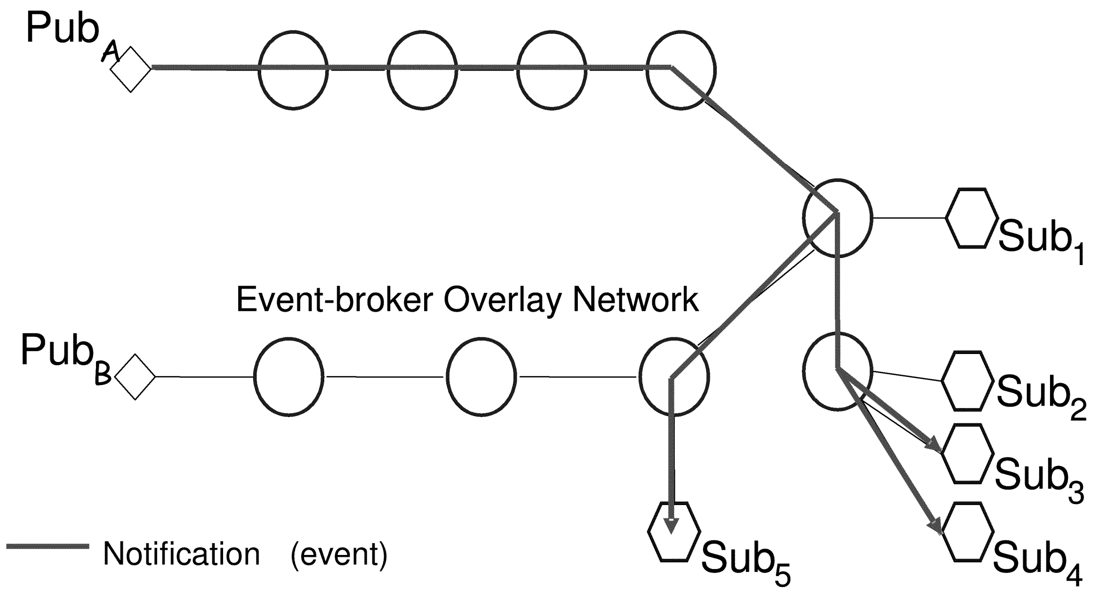


Figure 3-4 Notifications Follow Subscription Trails

The matching of a notification to a subscription occurs when all attributes listed in the subscription are also listed in the notification and for these attributes, the values provided in the notification are within the listed constraints.

3.4.1 CBPS Design Goals Review

On the surface, Internet-scale context-awareness seems well matched to use a content-based publish / subscribe infrastructure. The separation of concerns provided in CBPS is exactly what we need; CBPS systems provide efficient event distribution and are scalable. However, when we examine our wide-scale context-awareness needs as described in Chapter 2, where we are concerned with the relationships of dynamic data that originate from multiple publishers, we observe that this falls outside the focus and design intentions of CBPS systems.

CBPS has a core design goal of being able to associate publishers with subscribers based solely on the content within an event (message). The claims are that it is useful for all of one-to-one, one-to-many, and many-to-one information flows. These features, plus the underlying separation of concerns, make CBPS attractive as a basis for ISCA. However, CBPS core efficiencies lie in the ***distribution of a single event***, from one publisher to numerous subscribers, taken in isolation from all other events. What this really means is that we can consider CBPS as a one-to-many system where none of the end-points are known to each other. One-to-one behavior is a byproduct, but at a cost of flooding the network with either advertisements or subscriptions. Many-to-one behavior results where multiple layers of one-to-one can simultaneously exist.

Additionally, an implicit assumption appears to exist that events, as the name would imply, have a low frequency of occurrence. Thus, an Internet-scale system would merely need to be capable of supporting large quantities of low frequency events. However, as described in the problem section, we acknowledge that applications will emit a continuous stream of event data. If we consider a GPS enabled device, it will constantly report its position – typically at 1Hz. The transition to more sensors with higher data rates will come with the pervasive Internet and consist of frequent repetitions of status-like information. Although this kind of data and associated quantities drive core ISCA efficiency requirements, they do not appear to be an essential part of CBPS goals.

3.4.2 Expressiveness Versus Efficiency Tradeoffs

The focus on the distribution of a single event in isolation has led to performance driven design decisions that limit the expressiveness of the language. In particular, only conjunctions of predicates composed of relational operators comparing event data to subscriber defined constants are permissible. Part of this decision includes avoiding user defined types. By omission, the ability to perform attribute-to-attribute relationships is not allowed. For example, if an event contains the fields price and earnings, one cannot write a subscription including the predicate price / earnings < 10. These decisions are based on a local optimization approach – that by making the individual nodes fast, the overall performance will be fast. Consequently, there has been significant research into developing fast matching algorithms for the event-brokers [CW-2003].

To determine contextual-relationships, the data from multiple sensors (publishers) must be combined. Although there is some support within event-brokers to manage expected sequences of events (e.g., first an event matching filter 1 and then an event matching filter 2) there is no support for attribute-to-attribute relationships across events. For example, if event1 contains field price and event2 contains earnings, one cannot include a subscription event1.price / event2.earnings < 10.

3.4.3 Event Propagation Analysis

To evaluate the baseline costs of using CPBS to support our ISCA needs, we consider the average number of hops (aka event-hops) each message must travel from publisher to subscriber. To remain consistent with the common implementations of CBPS systems, we will consider a hierarchical organization centered about a single root node, typically displayed as a star-diagram as shown in Figure 3-5. [29] To simplify computations, we will assume that all nodes at a given level use the same branching factor and all are full. Based on our expectations of billions of sensors and consumers, it is not unreasonable to assume an approximation of a uniform distribution.[30]

[29] Other network structures are being researched [Muhl-2002]. The issues discussed should port at some level.

[30] Some locality is expected but cannot be evaluated at this time.

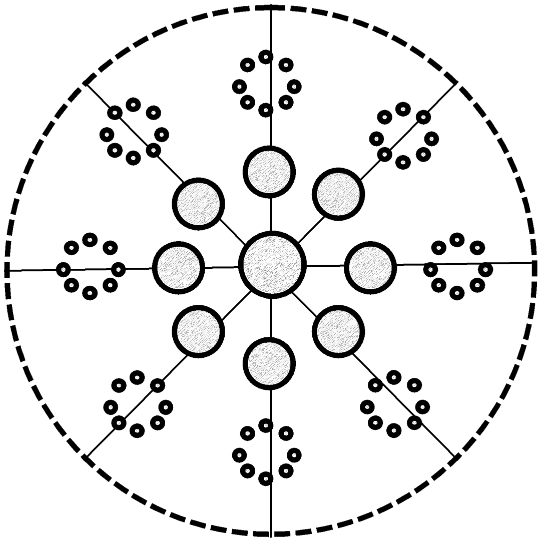


Figure 3-5 Hub-and-Spoke Overlay Network Model.

Representation of a network with a branching factor of 8.

The intuition is that if the branching factor at the root node is 8, then there is only a 1 in 8 chance that both producer and consumer will reside in the same branch. Thus, there is a 7 in 8 chance of separation, resulting in the number of hops between producer and consumer being nearly equal to the diameter of the network. We can write the following program to compute the average number of hops.

```
int         levelHops = 0 ;
double      levelExpected = 1.0 ;
double      hopsExpected = 0 ;
double      sumExpected = 0.0 ;
double      numKidsPerLevel[] ; // init elsewhere

for( int i = 0 ;   i < numLevels ;   i++ )
{
  levelHops = (numLevels - i) * 2 ;
  if  ( 0 < i )
    levelExpected /= (numKidsPerLevel[i] - 1.0) ;
  if  ( i + 1 < numLevels )
    levelExpected *= (numKidsPerLevel[i+1] -1.0) / numKidsPerLevel[i+1] ;
  hopsExpected += levelHops * levelExpected ;
}
```

We then execute the program using ten different network configurations while holding the total number of nodes constant. This yields Table 3-1. As shown in the final column, when we assume a uniform distribution, then the average number of hops approaches the diameter of the network.

Table 3-1 Average Number of Event Hops Approaches Network Diameter

Radius	#Nodes Lvl-1	#Nodes Lvl-2	#Nodes Lvl-3	#Nodes Lvl-4	#Nodes Lvl-5	Total Nodes	Avg. Hops	Dia-meter
2	4	160				641	3.5	88%
2	8	80				641	3.75	94%
3	4	10	16			641	5.45	91%
3	8	8	10			641	5.72	95%
3	10	8	8			641	5.77	96%
4	4	4	4	10		641	7.34	92%
4	10	4	4	4		641	7.73	97%
5	2	2	4	4	10	641	8.34	85%
5	4	4	2	2	10	641	9.28	93%
5	10	4	4	2	2	641	9.73	97%

We see three key results in this table. First, while keeping the number of levels constant but increasing the branching factor near the root node – at the lower numbered (higher priority) levels, then the average number of hops increases. Second, the average number of hops for each event will approach the diameter (D), let us assume a large network will have at least 10 nodes off the root, in which case we can reasonably say, for future reference, the average number of event hops from an arbitrary publisher to an arbitrary subscriber is about 0.95D. Third, we recognize that if the root node has a branching factor of B, then (B-1)/B of all event traffic will flow through the root node. The implications of these results are discussed in the next section.

3.4.4 Many-to-One Implications

There is a basic level of contextual filtering provided when event data is evaluated against the constant constraints of a subscription. However, detecting relationships from real-world sensor data is likely to be more complex. For example, the subscriber might wish to construct an event which, when satisfied, will trigger the air-conditioner to turn on. The subscription filter would be (time > 5 pm) and (temperature > 75 degrees). But, to be activated, there must be a single publisher that produces both time and temperature. Unfortunately, we can expect sensors to follow similar separation of concerns principles and be highly specialized; there are likely to be separate time and temperature sensors.[31]

The core principle for ISCA is a many-to-one relationship. However, unlike the CBPS idea of a many-to-one relationship, ISCA is concerned with many producers of information to infer one relationship. As seen in the air-conditioner example, the information for context-awareness may originate from multiple publishers. Some external clock provides the time and a thermometer (or a thermometer sensor network distributed throughout the house) provides the temperature.

Because there is no support for applying functions to multiple attributes within an event or between multiple events, such as required for the Cartesian distance between the location events from two different publishers, then to achieve ISCA, all work must be performed at the subscriber. The subscriptions for information must be managed and when events are received, all filtering must be done at the subscriber's end. This currently requires subscribers to subscribe to the raw location information and compute the distance themselves as shown in Figure 3-6, hence the efficiencies of evaluating subscriptions at the publisher's event-broker to reduce unnecessary event traffic are lost.

Perhaps most significant is the cost involved with pushing events across the

[31] This example intentionally ignores the fact that most modern thermostats include clocks.

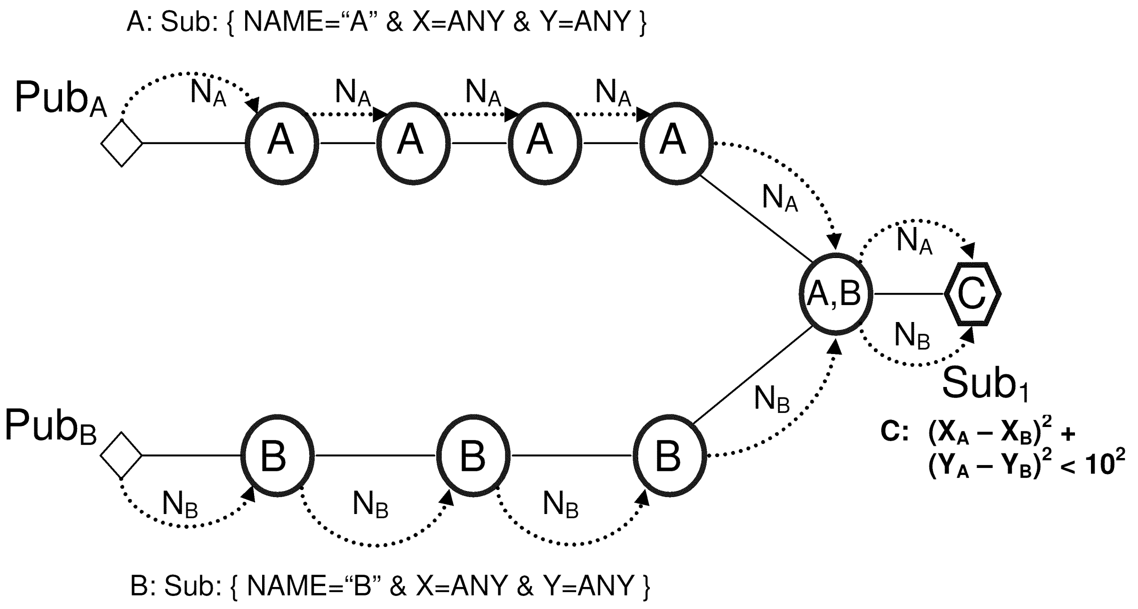

A: Sub: { NAME="A" & X=ANY & Y=ANY }
Pub$_A$
N_A
N_A
N_A
N_A
A
A
A
A
N_A
A,B
N_A
C
N_B
Sub$_1$
C: $(X_A - X_B)^2 + (Y_A - Y_B)^2 < 10^2$
N_B
Pub$_B$
B
B
B
N_B
N_B
N_B
B: Sub: { NAME="B" & X=ANY & Y=ANY }

Figure 3-6 All Data Flows to Subscriber.

All potential relationship data must flow to the and-subscriber be processed.

network from the publishers to the subscriber. We have previously determined the average number of hops is about 0.95D. Not only do we suffer the network delays and I/O costs, but each event-broker along the path from publisher to subscriber must process the event in order to forward it – approximately 0.95(D-1). By corollary, we noted that (B-1)/B events pass through the central hub. We also note that of the B' remaining events (i.e., 1/B) that will not make it to the hub, the same behavior of (B'-1)/B' events must pass through the root (hub) of the branch. The point of which is that the core of the network is heavily burdened with network traffic.

To put this into perspective, take the criminal gang member (CGM) scenario from Chapter 2. The sensors come from the "prison without bars" concept where each person is required to wear an ankle bracelet with a GPS device [BBC-2004]. The parole officer monitoring these relationships has only a lightweight edge-device.

If a gang member obeys the law and does not associated with others, then every event-hop of his location data is an unnecessary drain on the system. A gang member in flagrant disregard of the law will be quickly detained. And, a gang member who does not want to get caught will likely spend close to 99% of the time obeying the law, resulting in 99% of the events generated on his behalf being unnecessary.

Requiring the events to pass through the network and detecting the relationships at the end consumer wastes computation, hurting both the subscriber and the middleware. Who cares, Moore's Law tells us that we will soon be doubling capacity. It wastes bandwidth. Who cares, Gilder's Law claims bandwidth is growing at three times the rate of processor capacity.

We care. We know we will continue to make smaller devices; we know they have major power constraints; and we know that transmitting or even passively listening takes precious energy. This makes the whole proposition infeasible for lightweight edge-devices; yet these will be devices that the end consumers are interested in for contextual-relationships.

We care. We know many (to most) of our edge-devices are becoming wireless with a convergence into cell-phones. The bandwidth, while available, is fee based at (currently) undesirable rates, paid to the cellular service providers. We need to conserve bandwidth.

We care. As events get passed into the network, the network core, through which most of the events must pass, are required to perform more work, Thus, a bottleneck is created toward the center of the network. The insidious part of all this is that the system forms an ecology where any extra costs get shared with all other applications using the network.

3.4.5 Flood Forwarding – Advertisements or Subscriptions

The original CBPS work, SIENA [Carzaniga-1998, CRW-1999], used flood forwarding of all advertisements. Later work has led to conflicting beliefs as to whether or not advertisements and their behavior are useful – instead suggesting that using subscription forwarding is preferable. In the following paragraphs, we make the case that for the content of interest to Internet-scale context-awareness, using the flood forwarding technique for advertisements is desirable as the base behavior. We also consider why others might consider it less useful. Finally, we discover huge inefficiencies in flood forwarding either when all we really want is a one-to-one communication. This leads us to the realization that what we really need are user controls such that the context of the situation can be applied.

3.4.5.1 Repeated Data

Advertisements are primarily useful for cases where the data of a given event-type is frequently repeated. Any sensor that repeats its data – essentially providing status updates (e.g., monitoring the pressure gauge at a nuclear power plant or monitoring one's location) is well served by an advertisement. It is well served for three reasons.

First, it provides knowledge into the network that such information exists – in essence providing its own service (i.e., data) discovery mechanism.

Second, when using advertisements, subscriptions are stored only in event-brokers on the path between the subscriber and the advertisement originators (i.e., subscriptions only exists on nodes capable of receiving such information). If we use the subscription forwarding technique (without any advertisements) and a billion subscriptions exist in the system, then each event-broker must evaluate a billion subscriptions for each notification.

Assuming a Zipf distribution of data, then maybe 10% will all possess the most likely attribute, which means that every notification must be compared against 100 million items. If we have data updates at 1Hz for repeated data, from a single client, then we unnecessarily burden the system. If however, the only subscriptions that exist are those that have some potential of being met, then there may be perhaps a thousand different subscriptions that are live and 10% would result in only 100 to be evaluated.

Regardless of advertisement or subscription flood forwarding, the general presumption is that far more event notifications will be generated than either advertisements or subscriptions. Therefore, only allowing subscriptions to exist where they might possibly be met allows the system to be more efficient.

3.4.5.2 Any Source May Generate Critical Data

The case where flood forwarding advertisements makes less sense than flood forwarding subscriptions is where the data to arrive may originate from any of the publishers attached to any of the event-brokers.

This scenario is more appropriate to smaller systems, primarily homogeneous, using the CBPS techniques as a means for composing an application. However, as our goal is to provide an Internet-scale mechanism to which anyone can plug-in, it is clear that two possible remedies exist. One choice is for all publishers of such critical data to advertise their data. A second choice is to allow for flood forwarding of subscriptions. Which choice is best will depend on the context of use.

3.4.5.3 Mobility

When a publisher or subscriber is mobile, there is a transition cost as they leave a prior event-broker and re-attach to a new event-broker. During such times, there is a risk of delay in the data or, based on implementation choices, loss of data during the handoff. If a mobility service is used, then subscription forwarding adds no value (or vice versa). But, if the publisher is disconnected and reconnects elsewhere, even if it then advertises its data, a race condition exists that it might start publishing notifications before the subscriptions could be routed there. In such a case subscription forwarding must be weighed against the value and probability of lost information. Usually repeating information (e.g., status or location) does not merit subscription forwarding and critical events are unlikely to be mobile. But, they might, and thus we should allow for flood forwarding of subscriptions.

3.4.5.4 Under-Specification

An advertisement is not required to identify all possible fields it might produce. As such, an under-specified advertisement might, say, describe only half of its fields. A subscription can then only be matched against the subset of described fields, leaving the possibility that a useful subscription that might be matched does not find an appropriate advertisement.

It can be argued that this is merely a usage problem, but it leaves a hole in the system that a user might find frustrating. Flood forwarding subscriptions eliminates this problem, but at the cost of more subscriptions to be evaluated for each notification.

3.4.5.5 One-to-One Conversations

The most significant problem is the idea of one-to-one conversations. Although CBPS systems claim one-to-one support, the reality is that either advertisements or subscriptions must be flood forwarded. This is a huge cost just to allow two entities to communicate. We need mechanisms to allow a more efficient setup of one-to-one connections.

3.4.5.6 Flood Forwarding Summary

Context-awareness is about efficiency, typically through information control. The performance of flood forwarding advertisements or subscriptions is dependent on the use, and hence both need to be available for ISCA.

3.4.6 Content Dimensionality Study

One of the CBPS efficiency measures is to pass an event over a connection – either to the next event-broker in the chain or the end subscriber – only once. Therefore, CBPS can reduce the number of subscriptions that must be evaluated for a connection by detecting when subscriptions are subsumed by another, more general subscription. For example, a subscription { (0 < A < 10) and (0 < B < 10) } would be subsumed by either { (-100 < A < 100) and (-100 < B < 100) } or { (-100 < A < 100) }. Fast event matching is a key research topic [CW-2003, Kulik-2003, LNK-2005].

We tested these ideas using similar assumptions[32] to gain insight and if necessary, to develop our own algorithms. The question is how many subscriptions remain.

Test Series #1.

Dimensions: 8 to 512 dimensions (i.e., attributes) available; Zipf distribution
Dimension contents: Integer (base range: 1..10,000) where the integer value is its rank
Start: range 1..10,000; Zipf distribution
Length: range 1..10,000; Zipf distribution
Subscriptions: 8 to 65536
Attributes per subscription: 1..4 uniformly distributed.

[32] Our test were performed before the fast forwarding algorithms were published.

Subscriptions that were subsumed by another were eliminated. Subscriptions that were equal in all but one dimension and where the final dimension had an intersection were aggregated into a single subscription (e.g., Subscription 1: (0 < A< 10), Subscription 2: (5 < A < 15) would become Aggregate Subscription (0 < A < 15)). Our findings are shown in the Figure 3-7, Figure 3-8, and Figure 3-9.

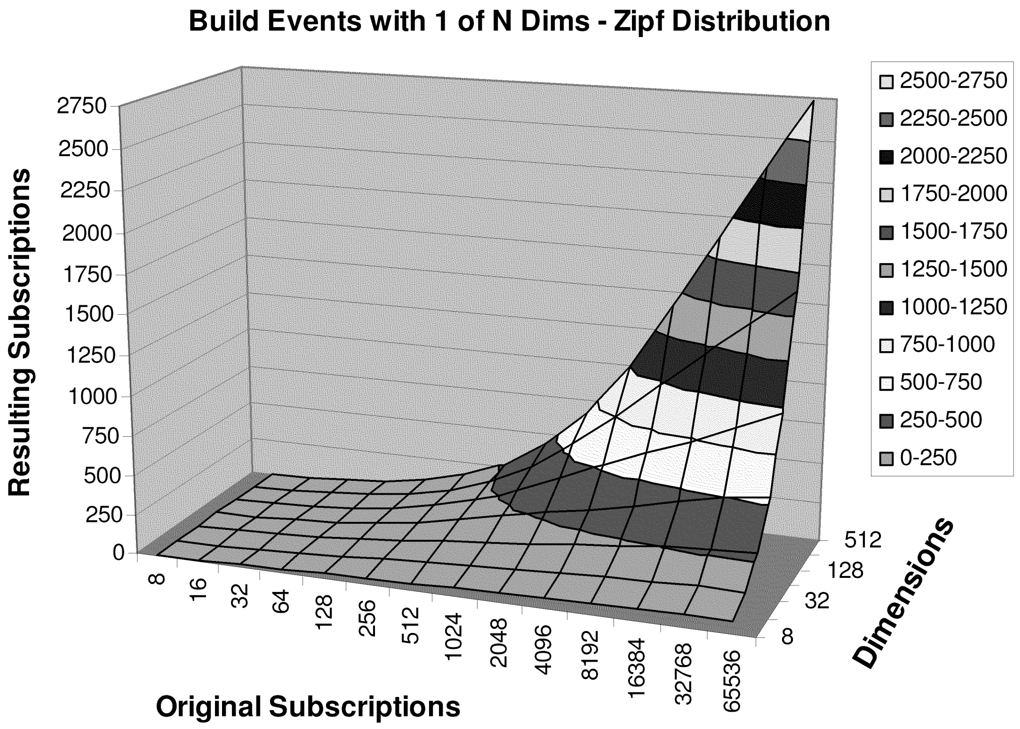


Figure 3-7 Subsuming Subscriptions of 1-Dimension.

With a single dimension for each subscription, we find that the number of resultant subscriptions is reduced as expected. But, as the number of dimensions increases, so do the number of subscriptions. This is also expected as the probabilities of overlapping data are reduced.

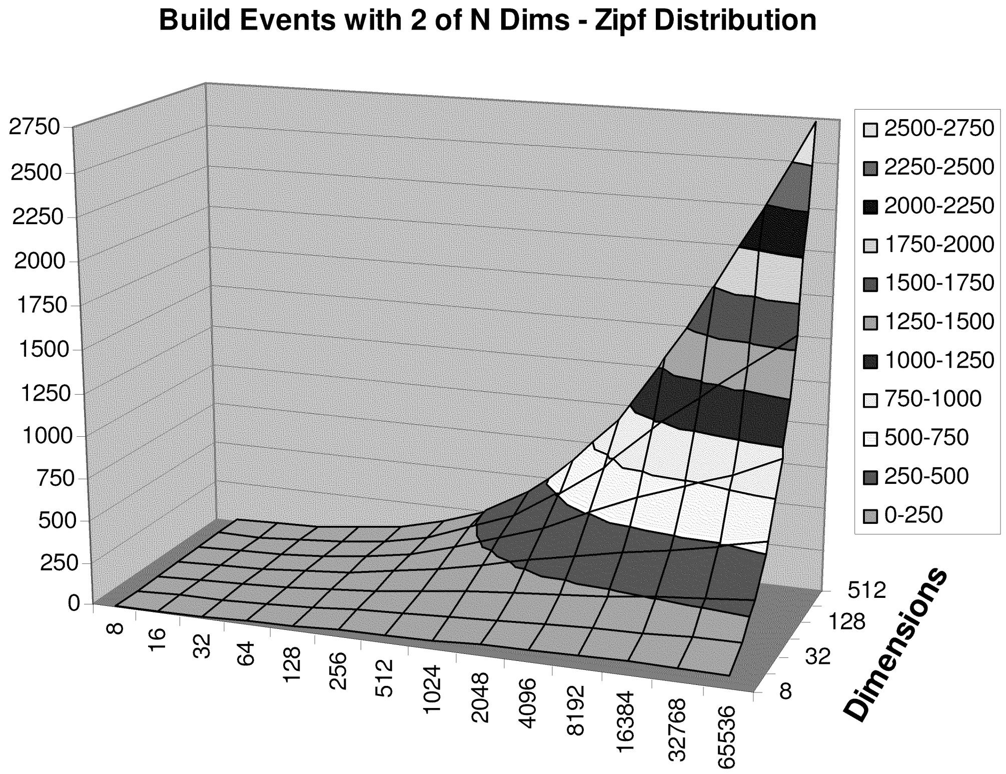


Figure 3-8 Subsuming Subscriptions of up to 2-dimensions.

With up to two dimensions for each subscription, we find approximately the same reduction. At first this might be surprising – it seemed we should have had more unique subscriptions. The reason we have fewer than expected – on the same order as if we only had a single dimension per subscription – is because of the distribution assumptions and the rules to subsume a subscription.

We have a uniform distribution in the number of dimensions for a subscription. That is, we are equally likely to have only one dimension in the subscription. That one dimension, based on the Zipf distribution for dimension selection, will tend toward the most popular. Then other subscriptions with only one dimension (i.e., 50%) will behave as they did in the prior test. Additionally, subscriptions that contain two dimensions effectively have two chances to be subsumed by a single-dimension subscription.

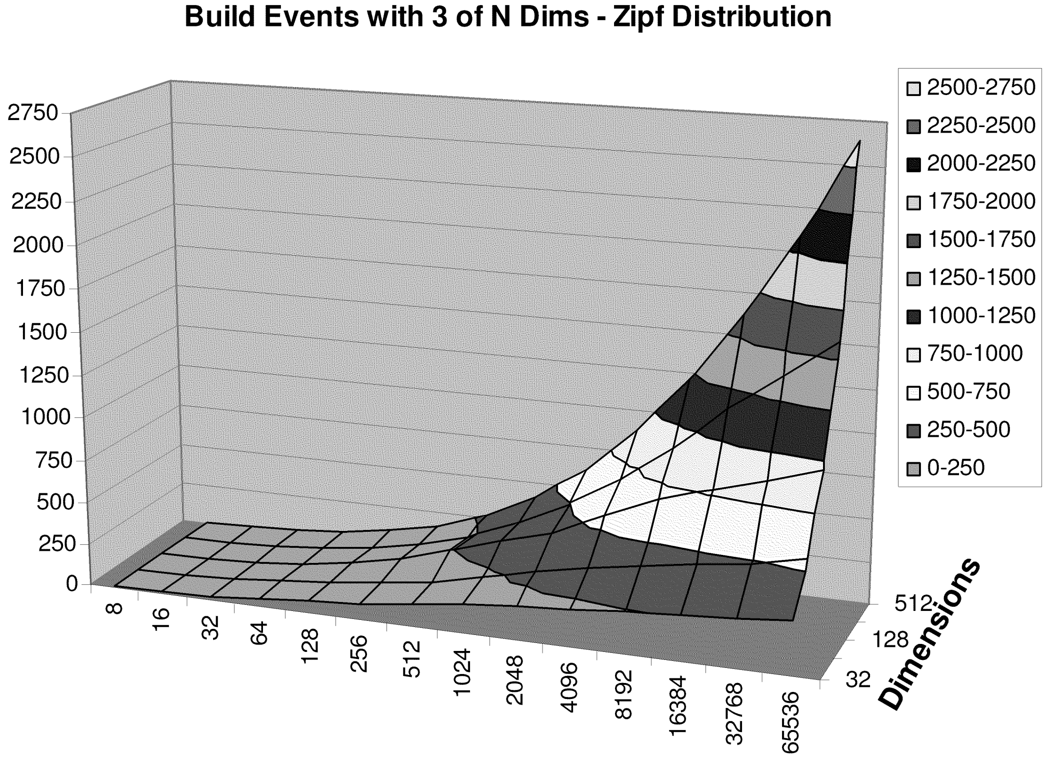


Figure 3-9 Subsuming Subscriptions with up to 3-dimensions.

With up to three dimensions for each subscription, we find a slight reduction. In light of what we learned with two dimensions, this is expected. Due to the Zipf distribution for the selection of dimensions, we had to increase the number of dimensions available to avoid a complete collapse of data into the few highly ranked dimensions. Four and more dimensions follow the same pattern of slight reductions, again with a need to increase the number of dimensions to avoid data collapse.

Given the reductions and observation about the significance of the highest ranking dimensions, one point of interest toward developing more efficient forwarding algorithms would be to first sort the dimensions in an advertisement, subscription, or notification based on a rank, possibly with adjustments due to the average number of dimensions one needs to check for the given rank, and then to search for matches based on that order.

We have previously identified the individual nature of context information. This implies that the natural Zipf distribution might not be appropriate for the content within each dimension. This leads to the test for uniform distributions.

Test setup #2.

Dimensions: 8 to 2048 dimensions available; uniform distribution

Dimension contents: Integer (base range: 1..10,000)

Start: range 1..10,000; uniform distribution

Length: range 1..10,000; uniform distribution

Subscriptions: 8 to 8192

Attributes per subscription: 1

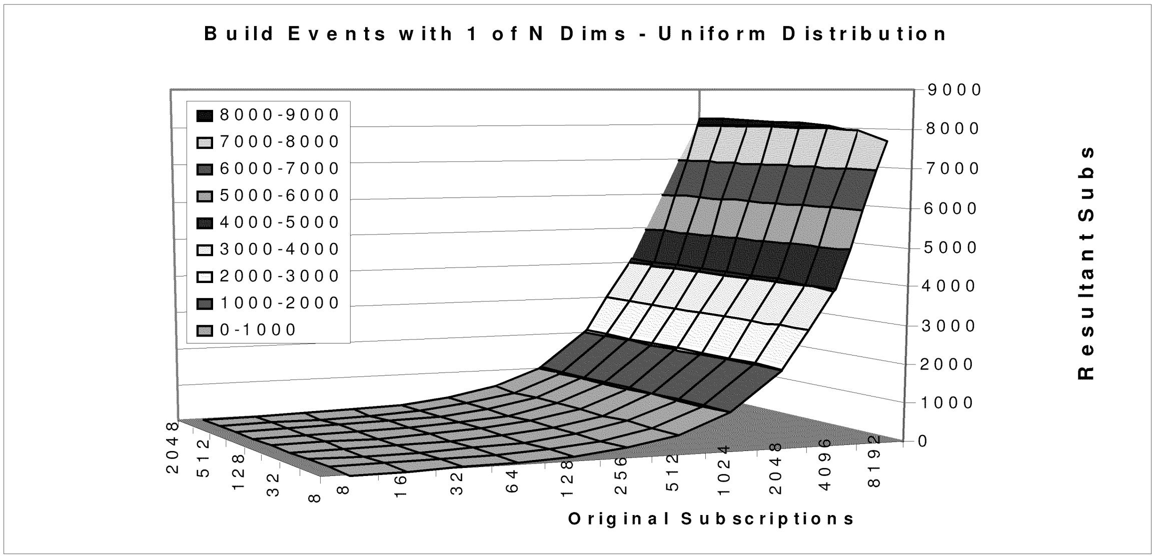


Figure 3-10 Subsuming Subscriptions Under a Uniform Distribution.

Under a uniform distribution assumption, there is little overlap between subscriptions. The implication, with respect to context-awareness, which we have determined to be dominated by individual interests, is that the event-brokers will need to manage more subscriptions than one might hope for. The effort to find subsumed subscriptions does not justify the gains. Also, the uniqueness of subscriptions indicates the potential for individually tagging each subscription for a hash-table lookup for propagation.

3.4.7 Strengths of the Approach

CBPS systems seem ideal for supporting extensible, scalable, context-aware systems. They provide the separation of concerns that we desire.

The brokered publish / subscribe paradigm makes publishers and subscribers largely independent from each other, while providing economies of scale through sharing in the overlay network. Publishers and subscribers can be readily added, accommodating new application functionalities, and new event-brokers can be added to improve the efficiency of the network, thus providing Internet-level scalability [Carzaniga-1998, CRW-2000].

By forwarding subscriptions to the event-brokers where the data enters the network, we are pushing application computations into the network, "the network is the computer." Thus we are able to short-circuit unnecessary computations by determining usefulness upstream, before the rest of the system must process the data. Pushing the filtering "service boundary" toward the publisher achieves valuable efficiencies.

The system inherently supports a basic level of context-awareness by filtering data according to a query predicate specified by the consumer. This context-awareness in a pervasive environment is the natural progression of Weiser's vision.

3.4.8 Drawbacks With Respect To ISCA

CBPS operates under a substantially different information flow, usage, and message quantity than ISCA. ISCA is concerned with the relationship of dynamic data and relevance to individual users. As identified in the problem section, an essential ingredient of context-awareness is that it is individualized. So, where CBPS could take a sports score and broadcast it to millions of listeners efficiently, the aim of ISCA is to take data from multiple publishers that match the criteria of a single relationship.

Most CBPS systems use similar subscription languages, with the relational operators { =, !=, <, <=, >, >= } as a basis. They are differentiated by the richness of the set of comparison operators they provide [RDJ-2002]. But, the relational operators are limited to comparisons with statically defined values, yielding filters like $X < 10$ and $\text{Price} = 100$, resulting in simple composition filters.

For context awareness, many interesting behaviors occur as the result of relationships between dynamic information. Sometimes the relationship of interest resides entirely within a single event. We might want to know if $X < Y$. If provided, we would then want the capability of a translated line $X - Y < 10$ or with a different slope (e.g., $\text{Price} < 5 * \text{Earnings}$).

Unfortunately, even the richest subscription language is inadequate for many interesting behaviors. A subscriber cannot subscribe to complex relationships because the expressiveness of the system does not provide support to specify attribute-to-attribute comparisons across event boundaries, where the events might originate from multiple publishers. Instead all events that potentially support such a relationship must be passed through the network and processed at the end-client.

Efficiency is measured primarily in terms of reduction in the number of events (hence event-hops), recognizing the added costs incurred at the edge-brokers to do so. Scalability is evaluated primarily in terms of reducing bottleneck situations.

The expressiveness limitations cause the service boundary to be pushed back to the client subscriber obviating earlier gains. The costs within the network and for lightweight edge clients make this infeasible. We need to push the responsibility back upstream toward the data provider.

3.4.9 Summary

CBPS gives us the increase in control over topic-based publish / subscribe and even provides a basic level of contextual filtering. Based on the original design goals, CBPS systems tradeoffs were made between expressiveness and efficiency. The consequence of these tradeoffs is that to determine complex relationships, all events are likely to traverse nearly the entire diameter of the network, yielding unreasonable costs – especially for the lightweight edge-devices of our future. We need to determine ISCA-style relationships earlier in the network which requires greater expressiveness and efficiency.

3.5 CBPS with Context-Awareness Extensions

Recent work on CBPS enables the aggregation of attributes from multiple data streams with more complex processing and filtering performed within the network [CK-Solar-2002a, JS-2003]. Common aggregations and transformations can also be shared [CK-Solar-2002a]. It is possible, then, to evaluate the proximity relationship at the first common event-broker node (***1st CN***) that connects the publishers with the subscriber.

Gryphon [JS-2003] and Solar [CK-Solar-2002a,CK-Solar-2002b] are exemplary of systems beginning to meet these requirements.

Gryphon employs a relational database model to events. The proximity of the gang members A and B would be computed by specifying a join of their location views at the 1st CN, followed by a select on the one-row table with a predicate like $\{(X_A\text{-}X_B)^2 + (Y_A\text{-}Y_B)^2 < D^2\}$. The authors acknowledge the need for upstream event suppression, and are working on an idea called "selective curiosity", which makes the aggregation node responsible for pushing event-reduction clues back to the information providers.

Solar employs an acyclic dataflow model, extending CBPS with deployed-code operators to filter, aggregate, or transform event data. With Solar, the proximity would be determined with operators that aggregate buddy location events at the 1st CN, transform them into distance, and then apply a filter for the distance constraint.

Both provide for context-awareness, while off-loading the subscriber. They still

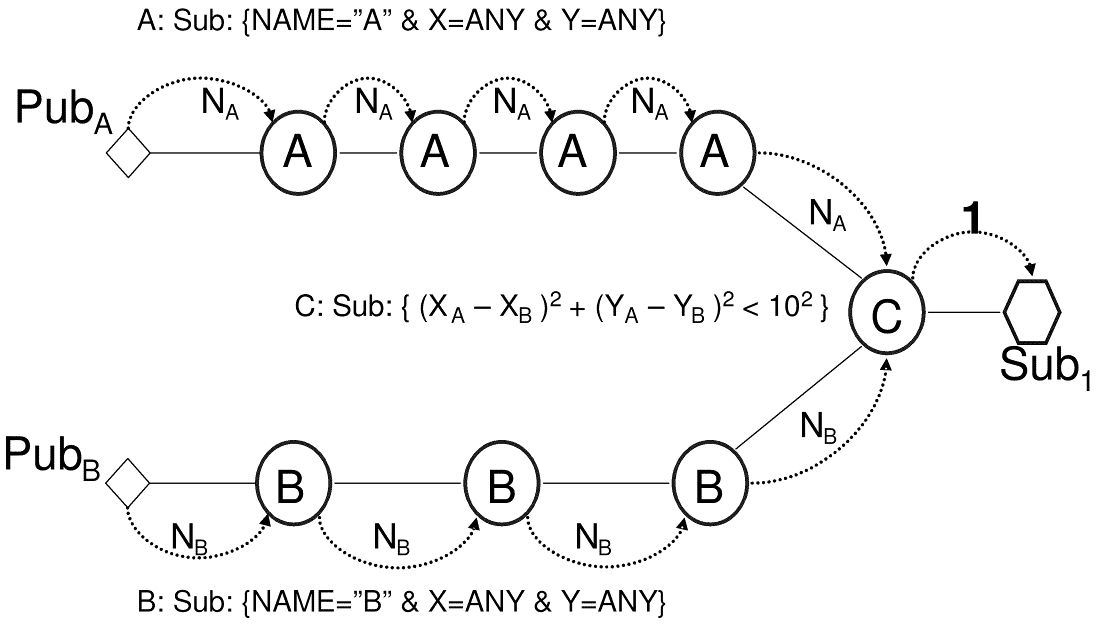

A: Sub: {NAME="A" & X=ANY & Y=ANY}
Pub$_A$
N$_A$
N$_A$
N$_A$
N$_A$
A
A
A
A
N$_A$
1
C: Sub: { $(X_A - X_B)^2 + (Y_A - Y_B)^2 < 10^2$ }
C
Sub$_1$
Pub$_B$
N$_B$
B
B
B
N$_B$
N$_B$
N$_B$
B: Sub: {NAME="B" & X=ANY & Y=ANY}

Figure 3-11 Aggregation at 1st Common Node

load the network with O(|events|) event traffic and processing at every node between the publishers and the 1st CN, as shown in Figure 3-11. Solar can save some processing through rate-limiting filters and event transformation.

3.5.1 Strengths of the Approach

The key strength is the development of mechanisms to maintain state about the information content. (Remember, pub/sub systems already maintain state about the potential sources and types of content.) This provides necessary support in that data in a relationship may be produced at different rates.

Relationship detection occurs earlier in the network to reduce the number of events hops that must be serviced. Because we know that most events pass through the hub of the network, then we can also recognize the hub as the most likely 1st CN, and its children as the next most likely set of 1st CNs (although with a drastically reduced chance). As a result, we can say that the relationship detection reduces the amount of event traffic by approximately half.

3.5.2 Drawbacks With Respect To ISCA

In architectures where there is no central hub, publishers may be distant from the 1stCN and each intervening event-broker node must process and forward all events.

In the hub-and-spoke architecture we have been using as our model, the 1st CN is most likely the central node or its immediate children. Consequently, all the complex relationship detection occurs at the network core. Already we have identified this core as a bottleneck for pushing events through the network. This problem is only exacerbated by making these same nodes perform the computations for relationship detection, thus adversely affecting scalability of the system.

3.5.3 Summary

To get the best possible performance requires evaluating context-aware relationships at the publisher's event-broker. To do so efficiently requires knowledge of the modeled relationship as well as how it will be used. Only the subscriber has this domain knowledge, and currently has no way of sharing it with the middleware. The expressiveness needs to be increased in order to improve efficiency.

3.6 Additional Observations and Insights

Common throughout the collection of published papers regarding publish / subscribe architectures are simplifying abstractions that limit one's ability to properly think about the problem.

3.6.1 Acyclic Graphs

The acyclic graph view implies that all information flows in a single direction, from sensors (publishers) on one side to actuators (subscribers) on the other. When context is composed of information from multiple providers, this leads to a funnel where significant processing intelligence must occur at the 1st CN.

As will be explored in depth in Chapter 4, efficient management of some problems (e.g., gang member proximity) require multi-directional information and filter flows – essentially creating a control loop. Gryphon shows an appreciation for this with their selective curiosity idea [JS-2003]. This idea is intended to be used solely to reduce message traffic, through the use of control signals; it does not intend to carry real content.

3.6.2 End-to-End

The end-to-end view ignores internal behaviors and costs in a network-brokered situation, as if the system is a point-to-point communications system. As we have shown, in a hub-and-spoke architecture, as is commonly presented in the literature, the internal nodes bear the brunt of all communications as the average number of event hops approaches the diameter of the network.

3.6.3 Event-Brokers Stand Alone

Prior CBPS instantiations presume a modular, plug-and-play of components to achieve scalability. In so doing, optimizations are performed at the component level and no ability exists to achieve system level efficiencies. This is akin to a greedy algorithm (component level) that may not achieve higher order optimizations.

One example of this behavior can be observed in the amount of traffic that must be endured to achieve an ISCA relationship. A second example of this behavior can be observed in the fact that every event-broker must re-compute every subscription, at every stage through the overlay network. But, we know the subscription must have started at some subscriber, therefore when an event matches a subscription at the publisher's event-broker, then there is at least one complete path through the system that should be able to be known.

What is required is a systems approach that recognizes that the event-broker nodes are not independent computational units, but rather part of a collaborative whole. The CBPS overlay network plus all the publishers and subscribers form an ecology – a single system – where the actions of one publisher, subscriber, or event-broker can affect the overall performance of the entire system. The knowledge that a publication matches a subscription should not need to be recomputed, it should be able to be reused.

3.6.4 Black-Box Information Hiding

Despite the ability to readily plug-in additional publishers, subscribers, and event-brokers, CBPS systems are highly abstracted systems – the proverbial black box. And, like all black boxes, are only useful as far as the original design considerations are consistent with current needs. As we have just seen, ISCA has a variety of additional demands beyond that provided for by CBPS. For our context-awareness needs, CBPS provides neither the expressiveness to enable the consumer to say what he really wants nor adequate performance because dynamic data relationships currently require all event traffic to pass through the network and be managed at the end-client or 1st CN.

3.6.5 All Information Is Content

The core premise of CBPS is that all information is content and thus subject to the subscription matching rules. This creates problems in two important areas. First, it inhibits providing contextual data (metadata) about the event itself. Second, a payload cannot be moved without being evaluated. The following paragraphs address both issues.

Context-awareness is based on the idea of information relationships to support decisions. This rests on the ability to acquire information – an information transparency – which runs counter to information hiding principles. The systems we have looked at have placed information hiding first (e.g., making the communications transparent). Unfortunately, no metadata about the system is available externally or internally.

Systems need to expose metadata. In exposing metadata to itself, many of the objectives can be more readily accomplished. Additionally, making metadata available to the user can enhance the experience along the lines of quality-of-service, such as a providing a form of internal documentation detailing how many milliseconds an event has been within the middleware.

As well as acquiring information, we must also be able to request the system act on it. For example, if a client knew something about the locality of information and wanted to limit the number of hops through the system, they should be able to express that information in a way that would allow the event-broker network to support their needs (e.g., (numHops <= 3)).

A similar need arises when an information producer (publisher) wishes to know about subscriptions to its data. For example, the publisher should be able to subscribe to subscriptions that match its advertisements. This would enable the development of smart publishers that would not need to transmit any information (beyond their advertisement) until someone subscribes to their data, thus preserving precious battery power.

By design, CBPS systems treat all data as content. However, this makes it impossible to return the results of a subscription to a subscription, as an example, without the system attempting to treat the replies as original subscriptions. A need therefore exists to support the idea of a "reply" because otherwise sending the logical reply of a subscription to a subscription might then look like a subscription itself. Both dispatcher-stateful-reply and message-stateful-reply have been considered in [HKCH-2002]. Using techniques like these we can generalize the idea of a ***payload*** concept and allow it to be part of either an original message or the reply. Then the payload could also include such information as deployable code or even security hidden encrypted data for which it makes no sense to examine. It may also have encrypted data that we then want the nodes to attempt to decrypt and validate before pushing forward.

3.7 Summary

We observe that a systems approach exposes key vulnerabilities within these existing systems, especially under the microscope of ISCA needs. We recognize the need to provide control-loops for efficient processing of multi-publisher data relationships; that the ecology of the system can be better leveraged; that metadata regarding the state of the system and content being managed by the system will be useful. And we observe that support to allow the end-client a way to inject their contextual knowledge into the network is necessary. These become our opportunities.

Existing mechanisms are inadequate at meeting our ISCA needs. Of the available options, content-based publish / subscribe systems are a more natural substrate for context-aware applications because they provide the right separation of concerns, efficient event distribution, extensibility, and scalability. However, the idea that economies of scale will be achieved in distributing an event from a single publisher to multiple subscribers runs counter to context-awareness environment where events of interest are highly individualized and complex relationships of interest require data from multiple publishers.

The separation of concerns afforded by CBPS middleware precludes publishers from collaborating with each other to efficiently detect context-aware conditions for publication as events. The black box design approach to CBPS middleware also hides hardware, software, and network performance characteristics as well as brokered content in violation of the new transparency requirements of context awareness. Furthermore, this approach results in reduced middleware performance because the constituent components are not treated as a coherent system.

4.0 Open Implementation Extensions to CBPS to Support ISCA

Our solution addresses the three key problems. First, a middleware system, by itself, does not have enough knowledge about both sensor reporting and user needs to make the appropriate efficiency (algorithmic) decisions. Second, "users" who do have the knowledge, are not provided adequate expressiveness to share their domain knowledge with the system. Third, the ecology formed by a middleware system and its associated users is rife with under-utilized contextual information, resulting in higher than necessary event hops and bottlenecks.

It might seem like the addressing the added requirements of ISCA might require abandoning CBPS, but to the contrary our solution generalizes the use of content addressability to solve the numerous problems introduced by ISCA. Our solution to these CBPS problems is based on open implementation techniques and the power of content-addressability. First, we provide a conceptual overview of the solution. Next, we explore our solution in the context of a proximity relationship. Finally, we flesh out additional concept details for the ideas presented in the first two passes.

We show in our extensions to CBPS, as implemented in Fulcrum,[33] that increasing expressiveness and information availability enables better collaboration among system components for more efficient and scalable solutions.

4.1 ISCA Overview

If for the moment we assume these key issues have been resolved, what would an

[33] Fulcrum implementation details are provided in chapter 6.

ISCA system look like? To answer this we must first put a few other assumptions in place.

1. We choose a CBPS infrastructure, as it possesses great strengths that we want to provide to users. In particular, the separation of concerns provided through content-based routing decouples subscribers from publishers.
2. We assume the sensors (publishers) are dumb; they connect to the system, advertise their event types, and report data.
3. Context-aware applications (subscribers) connect to the system and subscribe for a relationship (event) of interest, say to detect proximity parole violations of two gang members, A and B.

What then should the proximity subscription look like? More importantly, how will the system efficiently determine satisfaction of the proximity relationship in a way that is also scalable? This overview provides specifications that we implement in the subsequent sections.

4.1.1 Filters at Event-Entry

We start by following the lead of SIENA, Gryphon, and Solar, which have all shown the value of pushing the application into the network. SIENA pushes a simple filter all the way to the event-entry. Gryphon and Solar push complicated filters to the first common-node (1st CN). We need the best of both; we need to push complicated filters to the event-entry edges of the network. Thus, when a location event is received at the event-entry edge-brokers, each event will be discarded immediately unless it satisfies the proximity relationship, in which case it will be reported to the end user. Due to the low probability of relationship satisfaction, most events will be discarded.

Performing proximity detection at the event-entry edge of the network serves two purposes. First, the network is not burdened with useless event traffic. Second, the computational costs incurred at the event-entry nodes scale well. In a hub-and-spoke model, with a consistent branching factor of B, there are just over B-1 times more edge nodes than internal nodes. Thus, we can distribute complex filtering over a greater number of processors. And, the different relationships for which an edge-broker will be responsible are directly related to the sensor data it receives first-hand. Thus, (dedicated) edge-brokers are inherently under less computational demand than internal-brokers and thus better able to support the additional demand.

4.1.2 Distributed Algorithms

With multiple publishers, it is likely that the devices of the two gang members, A and B, are reporting to different edge-brokers, which means that the two brokers need to collaborate with each other in order to determine satisfaction of the proximity relationship. For this we need a distributed algorithm configured as shown in Figure 4-1. The two diamonds on the left side are publishers, Pub_A and Pub_B, producing event-types

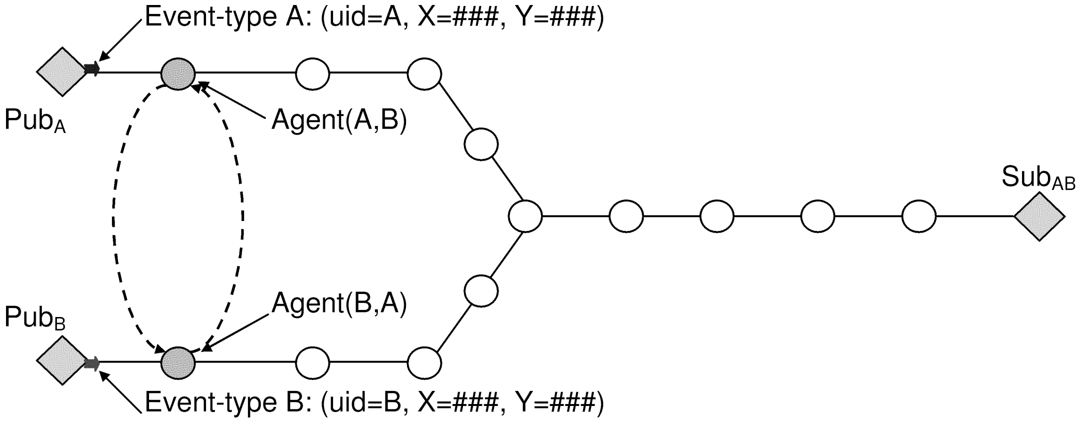

Event-type A: (uid=A, X=###, Y=###)
Pub_A
Agent(A,B)
Sub_{AB}
Pub_B
Agent(B,A)
Event-type B: (uid=B, X=###, Y=###)

Figure 4-1 Desired Goal State. We want istributed algorithms to filter events at network entry

A and B, respectively. For example, event-type A = (uid=A, X=#, Y=#). The diamond on the right side is the subscriber interested in a specific relationship between data from event types A and B. The circles are event-brokers. The two rose-shaded circles adjacent to the publishers are paired instances of the distributed algorithm, hosted on event-brokers. The dashed arrows show the logical communications between the algorithms required to effect collaboration (e.g., agent(A,B) subscribes to events generated by agent(B,A) and vice versa).

4.1.3 Subscriber-Provided Algorithms

We could build a distributed algorithm for proximity into the infrastructure. However, as seen in the motivating scenarios, there are a wide variety of sensors and combinations into which a user might desire to create relationships among them that prohibit our building solutions directly into the infrastructure. The kinds of efficiency we need require the use of domain-specific knowledge, held only by the subscribers. We need to provide the end-subscriber with mechanisms by which he can express the specific relationships of interest. Therefore, the subscriber must be able to provide their own special-purpose distribute algorithms, presumably as code, and be able to specify event-brokers in the infrastructure where the code should be executed. This must be achieved while preserving CBPS's separation of concerns.

4.1.3.1 Aside: Open Implementation

The open implementation software design technique [Kiczales-96, KLLMMM-97] was developed for just such situations. It describes a logical second interface, called the meta-interface, through which efficiency guidance may be provided by the user (typically the programmer). The user, then, assists in the selection of the module's implementation strategy. The open implementation research defines four varying levels of user control, described as four interface layers.

1. Standard API – No guidance.
2. Descriptive hints – e.g., lots of inserts, lots of searches, few deletes.
3. Identified implementation strategy – e.g., named strategy such as hash-table.
4. User-provided implementation strategy – e.g., class that adheres to a well defined interface specification.

This approach allows the client to optionally tailor the module's implementation strategy to better suit its needs, thus retaining the advantages of closed implementation modules (i.e., the traditional black box).

Creating an open implementation system is something of an iterative process. First, it requires the developer to provide layer 4 mechanisms because actual usage patterns are unknown. Once these become evident, appropriate implementation strategies can be coded and embedded in the system allowing a user to reference them by name. (These may cause the interface standard to be revisited.) Next a “language” allowing a user to describe the problem or algorithm behaviors can be created.

4.1.3.2 Open Implementation Applied to CBPS

These four interface layers applied to CBPS would appear as follows. The first layer is equivalent to receiving our data through a CBPS system. This is available by default. Layers 2 and 3 require the system to possess a variety of implementation strategy possibilities from which the system can choose based on the user’s description (layer 2), or from which user can choose because he knows the available list of strategies (layer 3). Layer 4 provides the ability to the user to write code to be executed by the middleware.

As Internet-scale context-awareness is in its infancy, we are currently only providing layer 4 capabilities. In so doing, we are able to give the end-client specific control over what *service* (e.g., subscription filtering) is to be performed.

4.1.3.3 Algorithm Deployment

How then can the user code be deployed into the network without exposing network details or violating separation of concerns? Separation of concerns would be lost if the subscriber learned the structure of the network to load the code on the publisher. Fortuitously, CBPS gives us access to the powerful content-based routing mechanisms for use in open implementation.

Consider the problem of routing user code to the event-entry. We already have a mechanism for routing a subscription to the event-entry. Can we use the same mechanism to deploy code? We need a feature analogous to a subscription that will allow us to trace advertisements back to the publisher, to which we can attach the user code we wish to deploy into the network. We call this subscription-like concept a *deployment slip* and use an e-mail metaphor where we attach the deployment slip to the user code in a way that lets us add as many addresses as desired as shown in Figure 4-2.

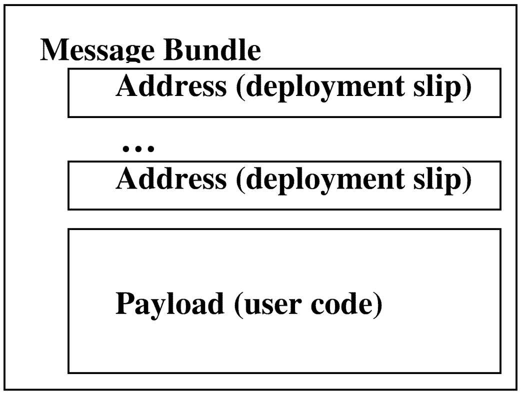


Figure 4-2 Actual Deployment Slip Usage

Using the content-addressability feature in this way is allows us to name specific edge-brokers without violating CBPS separation of concerns. In open implementation terms, this is equivalent to using interface layer 2.

Although we want the content-addressability of a declarative subscription, we do not want the deployment slip to actually perform the subscription behavior of passing event notifications back to the subscriber. This means we need to separate the content-based routing capabilities from the semantics of advertisements, subscriptions, and notifications; we need to make each behavioral aspect an independently composable feature. Having content-addressability available as a separate mechanism will permit us to use it in a number of new ways.

Now, we are able to consider the evolution of the network. First, as shown in Figure 4-3, we note that every event-broker receives and maintains a copy of all advertisements. Then, Sub_{AB} uses a deployment slip to route user code back toward the advertisement's origins as shown in Figure 4-4.

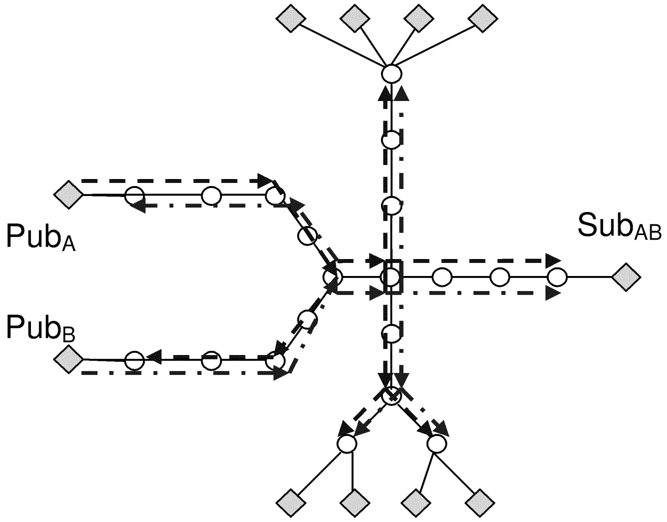


Figure 4-3 Flood Forwarding Leaves Advertisement Copies at All Nodes

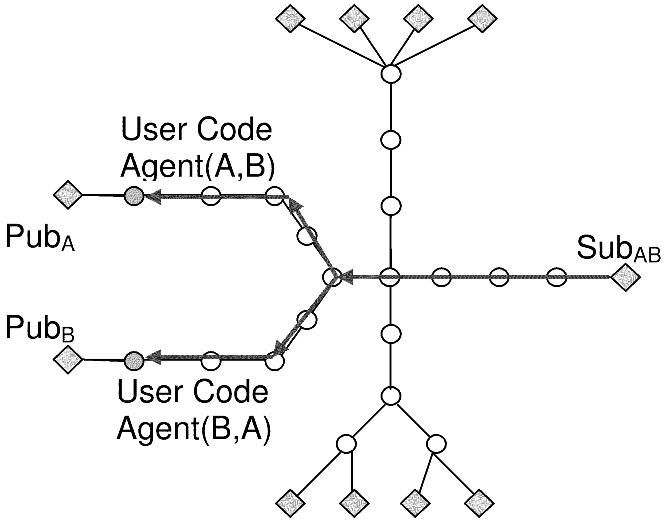


Figure 4-4 Deployment Slip Routes Toward Advertisement Origins

The user code is deployed when it reaches the edge-brokers (adjacent to the publishers). Initialization parameters cause the components to be instantiated as Agent(A,B) or Agent(B,A).

4.1.3.4 Message Bundling

As seen in the previous section, we need to bundle a payload, such as the user code of an active subscription, with the regular CBPS content-addressing information components. We call this *message bundling* and use it in a variety of ways to combine information components for distribution as a single, logical entity. On arrival at an event-broker, each element of the message bundle is individually processed as appropriate, but when forwarding is called for, the entire bundle is passed along. In essence, it is merely a reminder that any CBPS-based middleware is a distributed message-passing system. The break from traditional CBPS is that we are providing the concept of a separate payload such that now not all message content is available for matching.

Coincidently, the message bundle also allows us to provide more of a chunky message interface instead of a chatty one. Naturally, bundling multiple messages together can reduce message overhead with respect to content, both over the wire and in general processing. For example, all the advertisements of a publisher could be bundled in a single package, which would then use the standard flood forwarding distribution mechanisms.

ISCAMsg – a container
  Advertisement 1
  Advertisement 2
  Advertisement 3

4.1.4 Communications Setup

How then should the newly deployed components of the distributed algorithm connect with each other in order to establish collaborative communications? The components should not know enough about each other, or the nodes on which they are deployed, to directly connect. We prefer using the CBPS communications infrastructure that is already in place to maintain separation of concerns and to preserve its standard communications interface. Thus, the same algorithm could be deployed anywhere in the network, including at the client.

However, to create the connectivity for this one-to-one *conversation* in a basic CBPS system would normally require the entire network to be burdened by a flood of forwarded advertisements (subscriptions). Agent(A,B) would advertise its collaboration events (Collab_{AB}); Agent(B,A) would advertise its collaboration events (Collab_{BA}); Agent(A,B) would subscribe to Collab_{BA} events; and Agent(B,A) would subscribe to Collab_{AB} events. The advertisements would each cost O(|network|). That would be too expensive for a one-to-one conversation. Instead, we should leverage our knowledge that the components are hosted at content-addressable edge-brokers of the user's choosing –

the components were deployed into the network using a deployment slip after all.

In Figure 4-3 and Figure 4-4 we saw how the subscriber, $\mathrm{Sub}_{\mathrm{AB}}$, was able to use the existing advertisements to deploy distributed algorithm components to the edge-brokers using a deployment slip. Now, we ought to be able to use a similar technique to provide the communications pathway between the algorithm components.

We introduce, then, a *routing slip* concept analogous to a deployment slip, for the purpose of making the "connection" between the various components of the distributed algorithm. Instead of deploying user code into the network, the routing slip "deploys" other subscriptions (e.g., to $\mathrm{Collab}_{\mathrm{BA}}$ events). As before, we use the e-mail addressing metaphor to allow multiple addresses as shown in Figure 4-5.

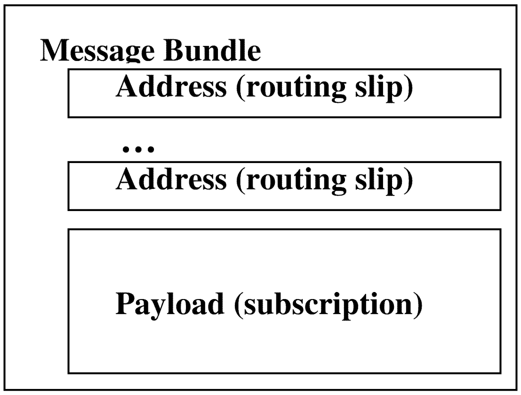


Figure 4-5 Actual Routing Slip Usage

As the routing slip traverses the network back toward the advertisement originator, the real subscription is deposited at each event-broker along the way as if it were tracing an advertisement back toward a publisher as shown in Figure 4-6.

The deployed subscription may then be used in the normal way – as the conduit over which we pass future notifications. Like the deployment slip, the routing slip needs to be inert. Also, it must not be subsumed by other subscriptions because its purpose is to

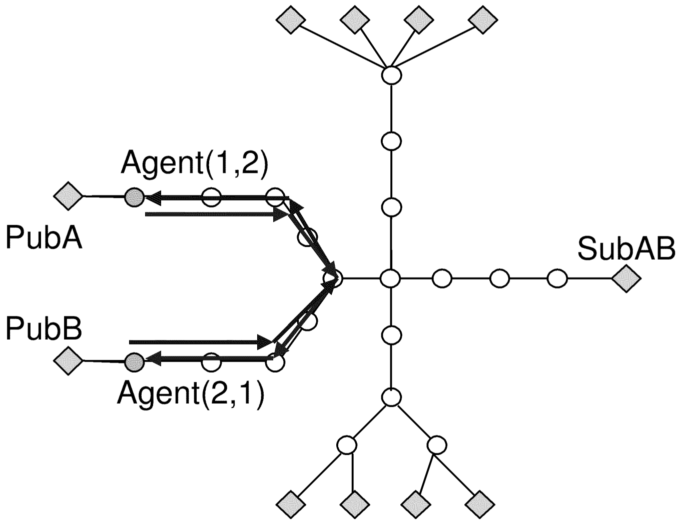


Figure 4-6 Connection Setup via Routing Slips

provide a link between two entities in the network.

At the risk of getting ahead of ourselves, if we are to use the routing slip to deploy subscriptions, a natural extension is to also support point-to-point deployment of notifications and even advertisements. Similarly, the routing slip need not be limited to content-based routing using subscription semantics, but should also be allowed to use content-based routing using notification semantics. We therefore tag the routing slip with a type-identifier to indicate its use of subscription or notification semantics. This flexibility serves the variety of ways that distributed algorithms might wish to set-up communications without sacrificing separation of concerns or violating the content-based routing paradigm.

4.1.5 Aside: Efficient Communications

The efficiency of the distributed algorithms is measured less by local processing time, which is assumed to be approximately equal regardless of where the processing occurs, than by the amount of event traffic required to synchronize the components of the algorithm.[34] How frequently should the components synchronize?

Sending each event to the opposing broker is really no better than sending each event to the subscriber. Although some locality could exist, it cannot be assumed. If we have a more complex relationship than just pair-wise proximity, say with N algorithm components, then the appropriate information needs to be shared with N-1 other event-brokers. If we're not careful, this could become N-1 times more communication than just sending all data to the end-client. This means our distributed algorithms need to perform contextually relevant down-sampling of the sensor reports, yet at a rate sufficient to maintain consistency between the algorithm instances. This is, the selective curiosity idea of Gryphon. We must provide an algorithmic model that can effectively reduce the number of event-hops through the system.

Effective down-sampling is achieved by decomposing the context-relationship into a collection of non-overlapping regions for each sensor such that independent filtering may occur at each sensor. That is, providing the data being reported by each sensor remains within its assigned region, then satisfying the relationship is impossible. This is where the application of domain knowledge becomes critical. Once data for a sensor moves outside its assigned region, it creates a synchronization event to collaborate with the other distributed algorithm components to cause them to refine (e.g., shrink) their filter region. Large filter regions (with respect to the rate of change of sensor data) are representative of relationships unlikely to be satisfied in the near future. As the filtering regions shrink, sensor event data is more likely fall outside the filter regions and more events will be generated. This results in reporting event data more frequently – providing

[34] We do however have to consider hotspots and load balancing,.

finer granularity of data – as the likelihood of satisfying the relationship increases. We expand on this idea with detailed examples in the Chapter 5.

4.1.6 Distributed Memoization / Caching

Finally, when the decision to propagate an event to the user occurs, or for that matter, to communicate synchronization events between the components of a distributed algorithm, how can it be efficiently delivered? ISCA events are likely to satisfy only a single consumer. The collaboration between the distributed algorithm components is also likely to be unique to the given conversation. Therefore, receipt of an event by an event-broker is likely to result in only a single forwarding (i.e., no multi-cast occurs). Unfortunately, CBPS design effectively requires an event to be blindly evaluated against all possible subscriptions at every event-broker along the way because each broker stands alone.[35]

One solution lies in the fact that the collection of event-brokers, and the associated publishers and subscribers, form an ecology. Once a notification is matched against a subscription at an edge-broker, then we know some end-subscriber is waiting for the event and some path through the overlay network exists as a series of event-brokers holding matching subscriptions. Therefore, we should be able to use the memoization technique to create a *distributed routing cache*. That is, once a routing calculation has been performed upstream, we should not need to re-compute it. We should be able to perform an O(1) computation to determine a match and forward the information immediately. We should be able to do this for notifications matching subscriptions as

[35] Fast forwarding algorithms are used to limit the costs to only those subscriptions with attribute name intersections [CW-2003].

well as for subscriptions matching advertisements.

We can readily achieve this in two steps by leveraging our knowledge of the content-based routing mechanisms. First, each persistent, matchable entity (e.g., advertisement or subscription) is given a cache identity, based entirely on the declarative part of the subscription such that all event-brokers would compute identical values. We compute a unique hash key using the message digest (e.g., MD5 or SHA1) of the entity.

The cache identity can now be treated as an attribute to be matched. We do not add it to the existing entity, but rather create a new subscription (advertisement) with a single attribute, cache identity, and store it with the original.

When a notification is matched to a subscription (or a subscription is matched to an advertisement) by an edge-broker, an *express forwarding slip* is created. This is effectively identical to the routing slip, introduced earlier. Now, the express forwarding slip is bundled with the regular subscription and forwarded to the appropriate destination.

As each subsequent event-broker receives the notification, it looks first for the express forwarding slip. It is tagged as a notification (or subscription as appropriate) and matched against subscriptions (or advertisements) with a cache identity attribute. When found, we immediately route the notification without incurring the normal event matching costs. On a successful match, the entire message bundle is forwarded. We know that at least one connection must have a match. If by chance other connections have provided an identical subscription, they too will have a reference to the same cache identity. Otherwise, the other connections may have different matching subscriptions, so the regular notification must be evaluated against their available subscriptions. If no express forwarding slip is found in a message bundle, then the event-broker is receiving the notification directly from a publisher and will begin the process. The result is that we pay the evaluation cost only at the event-entry node and then reuse the cached value along the rest of the path.

Each subsequent connection is treated as if we are receiving the data directly from a publisher such that when a subscription is matched, a new express forwarding slip is attached to the notification (replacing the old one) as it is passed down the line.

This technique reduces the costly matching burden from the internal nodes by allowing them to reuse the results computed by the edge-brokers, their antecedents. Collaborating to reduce internal costs increases scalability.

4.1.7 Contextual Controls

So far the discussions have primarily been about the disparity between basic CBPS and efficiently achieving our ISCA needs. Little attention has been paid to the other aspects of context-aware computing. If one were to advertise a lunch coupon for a restaurant in San Diego, everyone around the world would be required to receive it. Spam at its worst! We need contextual controls that would allow a user to limit the distribution or reception of data to a certain number of hops.[36]

A particularly useful example of this occurs when an implementation strategy agent is deployed to the edge of the network and subscribes to raw sensor data. Only the immediate sensor data is desired; the agent does not want to receive data from sensors connected at other parts of the network that produce similar data. Or, we might also wish to control distribution or reception of data with temporal constraints (e.g., location data

[36] We recognize that “hops” in an Internet setting often bear little resemblance to geographic constraints. Thus, using the number of hops is only an approximation to limit geographic distribution.

may only be good for one second). These contextual controls should be concurrent with the normal content-based controls.

To achieve such capabilities requires that the event-brokers expose contextual information, such as hop count or time within the middleware, and also that contextual filters be provided to the users. The context filter is essentially an attribute-constraint list and the contextual data an attribute-value list. The core capabilities already exist in CBPS, we only need to bundle these with advertisements, subscriptions, or notifications.

4.1.8 Overview Summary

We want the "user" to be able to create an efficient implementation strategy that can be attached to a subscription and pushed to the sensor-edges of the network. The implementation strategy is an efficient distributed algorithm that encodes domain-specific knowledge to immediately filter events on entry into the network unless they satisfy the relationship. Deployment of the implementation strategies is performed using deployment slips that leverage CBPS content-based routing mechanisms. The algorithm instances are connected for one-to-one conversations using routing slips that leverage the CBPS content-based routing mechanisms. When event notifications are passed into the network, matching costs are paid by the edge-broker to determine forwarding, and in doing so an express forwarding slip with the associated cache identifier is attached to the message to avoid duplicating matching costs at downstream nodes. Contextual controls are also provided to ensure only the local sensor data is processed.

Two key concepts of expressiveness and information availability permeate the prior discussion. Expressiveness occurs in allowing the subscriber to directly ask for the information of interest and to inject information about the usage context down into the network, while information availability is about contextual information from the network components flowing up for use in collaboration.

4.2 Fulcrum Implementation of ISCA – Proximity Example

In this section we describe how we achieve the above specifications with our reference implementation, Fulcrum. Here we present the conceptual details; the programming details are presented in Chapter 6. We show in the implementation of Fulcrum that increasing expressiveness and information availability enables better collaboration among system components for more efficient and more scalable solutions.

To put these ideas into context, we continue with our proximity relationship example. We start with the flow of the original advertisements and then incrementally add the new capabilities discussed in the overview section. The following discussion is Fulcrum centric; it discusses the flow of information and behaviors from the perspective of the middleware. The following is a brief outline of the process to be presented.

1. Advertisements are created by each publisher and flood the network.
2. The proximity relationship subscription and an efficient implementation strategy are generated and pushed into the network as a message bundle.
3. The implementation strategies are deployed as agents.
4. The agents subscribe to the raw location data.
5. The agents set-up one-to-one conversations with each other to exchange semantically significant events.
6. The proximity relationship is satisfied and the event is passed back to the subscriber.

4.2.1 Publishers Advertise Their Event Types

All examples begin with publishers advertising their event types. For the proximity example between gang members A and B, the message flow through the system begins with the advertisements from publisher A (Pub_A, blue)[37]

```
<MessageBundle>
   <Advertisement>
      <ac name=”uid” type=”string” op=”EQ” value=”A”/>
      <ac name=”X” type=”long” op=”ANY”/>
      <ac name=”Y” type=”long” op=”ANY”/>
   </Advertisement>
</MessageBundle>
```

and from publisher B (Pub_B, green)

```
<MessageBundle>
   <Advertisement>
      <ac name=”uid” type=”string” op=”EQ” value=”B”/>
      <ac name=”X” type=”long” op=”ANY”/>
      <ac name=”Y” type=”long” op=”ANY”/>
   </Advertisement>
</MessageBundle>
```

as shown in Figure 4-7. As before, diamonds are publishers or subscribers and circles are event-brokers.

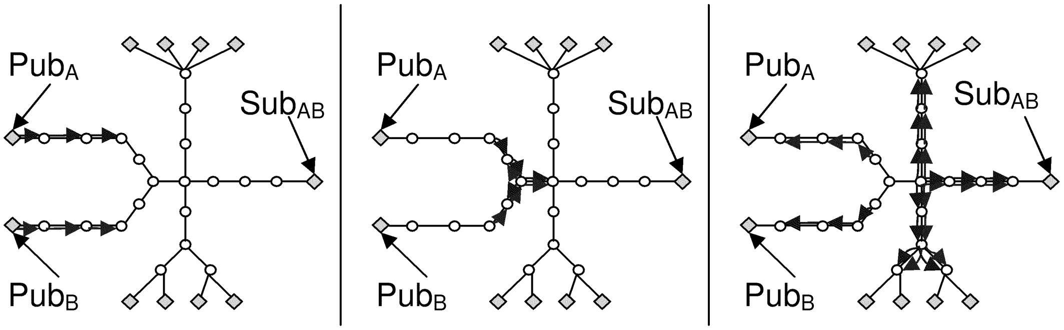


Figure 4-7 Flood Forwarding Advertisements. Each frame represents three steps.

The advertisements are to be flood-forwarded to every event-broker. At the first event broker, a message context is generated and attached to the advertisement as shown

[37] The “ac” tag stands for “attribute-constraint”. The “av” tag, used later, stands for “attribute-value”. A complete explanation of the tags and values are found Chapter 6.

in Figure 4-8.

```
<MessageBundle>
   <MessageContext>
      <av name=”hopCnt” type=”long” value=”1”/>
      <av name=”arrivalTime” type=”long” value=”1234567890”/>
      <av name=”travelTime” type=”long” value=”10”/>
   </MessageContext>
   <Advertisement>
      <ac name=”uid” type=”string” op=”EQ” value=”A”/>
      <ac name=”X” type=”long” op=”ANY”/>
      <ac name=”Y” type=”long” op=”ANY”/>
   </Advertisement>
<MessageBundle>
```

Figure 4-8 Message Context Sample

At each step of the forwarding, contextual information about the message is dynamically updated. E.g., the hopCnt is incremented by 1 and the travel time is incremented by the current time minus arrival time at the current event-broker (in ms).

At each event-broker, a copy of the advertisement is deposited in the event-broker’s local database with references to the connection over which it was received as is customary for a CBPS system. For Fulcrum, the advertisement also contains links to the message bundle and hence the message context for additional contextual filtering.

Additionally, the first step of the distributed memoization process for this advertisement occurs. The N-attribute advertisement is converted into a single-attribute advertisement, as shown in Figure 4-9, that is also stored in the local database, with a link back to the message bundle. [38]

```
<Advertisement>
  <ac name=”cacheId” type=”string” op=”EQ” value=”[AdvA]”/>
</Advertisement>
```

Figure 4-9 Distributed Memoization Advertisement Sample

[38] As discussed in Section 4.1.6, the memoization process actually creates a message digest for the advertisement (or subscription) to use as the cacheId. For simplicity and traceability, we have used a shorthand notation of “[” <type> <key identifier> “]”., for example., “[AdvA]”.

This secondary advertisement will later be matched against an express forwarding slip.

4.2.2 Subscriber’s Proximity Relationship

Next, the subscriber, the parole officer, aka Sub$_{AB}$, sets-up to subscribe to the proximity relationship between gang members A and B. First, she creates a proximity relationship subscription. She then develops an efficient distributed proximity algorithm as her implementation strategy and converts it into an active subscription with detailed deployment controls. The subscription and active subscription will be bundled together and pushed into the network. In the following sections, we break out each piece individually.

4.2.2.1 Proximity Relationship Subscription

First, Sub$_{AB}$ creates the proximity relationship subscription as follows:

```
<Subscription>
  <ac name=”uid” type=”string” op=”EQ” value=”(A.x-B.x)^2 +(A.y-B.y)^2”/>
  <ac name=”distance” type=”double” op=”LE” value=”10”/>
</Subscription>
```

Figure 4-10 Proximity Relationship Subscription Sample

The uid (unique identifier) is selected to uniquely represent the relationship. Because we do not control the entire namespace, we derive the uid from the unique identifiers of the participants as well as an expression of the relationship.[39]

4.2.2.2 Creating Distributed Proximity Algorithm

Sub$_{AB}$ creates a distributed proximity algorithm to take advantage of the domain knowledge about the structure of a proximity relationship. This algorithm will follow the open implementation’s interface layer 4 style and adhere to the pub / sub client implementation strategy interface specifications set-up for Fulcrum. The implementation

[39] Superscripts are not in the normal ascii string; they are used here only to show the distance formula..

strategy must behave as a first-class publish / subscribe client and provide initialization, threading, and communication APIs as detailed in Chapter 6. The algorithm may be tailored to apply other contextual knowledge, such as the relative speeds or course schedules of the observed subjects, A and B.

Rather than digress here to detail the distributed proximity algorithm, we instead present only a high-level overview and save the details for Chapter 5. We also note that the essence of open implementation is that the user is not locked into any one algorithm, but is allowed to create new algorithms to take advantage of evolving contextual knowledge.

For now, let us say that we want to run identical algorithms for each of the two publishers. Each instance of the algorithm will be initialized with the appropriate data to recognize that it is filtering A or B data on behalf of an AB proximity relationship. The algorithms create bounding regions around the sensor data produced, filter-out new events within the bounding regions, and synchronize with the paired instance when necessary. When a relationship is satisfied, only one instance reports the result.

4.2.2.3 Converting Implementation Strategies into Active Subscriptions

In our distributed environment, we must provide enough information for the implementation strategy to be instantiated at an appropriate place in the network. This means we must provide the executable code, name of the class to be instantiated, initialization parameters, and deployment control information. Instead of attempting to provide the implementation strategy as a parameter to a subscription, we instead view it as an extension that we will attach to the basic subscription. This attachment, is called an *active subscription*. We refer to each potential algorithm instance as an implementation strategy.[40] Once instantiated, we refer to them as agents.

The executable code may be composed of multiple classes. We choose to develop this work in Java, so one can think of passing .jar files. The components of a distributed strategy may be different and even come from a variety of packages; therefore we choose to manage executable code at the class level. Because our network interface is XML-based, the implementation code is provided in hex. Further implementation details are provided in Chapter 6.

We eliminate redundant transmission of executable code between event-brokers as follows. Each time executable code is received by an event-broker, it is cached and tagged with a unique hash key computed from the message digest of the executable code (e.g., MD5 or SHA1). Each time it is forwarded to another broker, that knowledge is stored, such that in the future only the hash key need be forwarded.

The name of the class to be instantiated lets the system execute the appropriate code. The class is required to fulfill the pub / sub client implementation strategy interface, as described in Chapter 6. Because the class name is identified separately for each implementation strategy instance, we can deploy different algorithms or algorithm components as necessary.

The initialization parameters are critical in that they tell a given instance what task it is to perform. For example, a proximity algorithm needs to know to compare A and B locations instead of C and D locations. These can be thought of as constructor

[40] Strictly speaking, both instances of the distributed proximity algorithm combine to form a single implementation strategy.

arguments, function parameters, or command line arguments. Sometimes, we use these parameters to inform a given component of its responsibility to report the satisfaction of the relationship. Because initialization parameters are provided, a common algorithm can be reused. The deployment control information is provided by deployment slips as introduced in the overview section.

For our proximity example, the active subscription would be bundled with the subscription as shown in Figure 4-11.

4.2.2.4 Deploying the Subscription and Active Subscription

In this active subscription, there are two implementation strategy instances identified – one for the sensor data being reported by Pub_A and one for Pub_B data. Inside each, the deployment slip identifies how to find the data source of interest. When Sub_{AB} submits this message bundle to Fulcrum, it is forwarded through the network, ultimately resulting in instantiating agents at the event-entry edge-brokers as shown in Figure 4-12 and detailed below.

Consider a partial network architecture as shown in Figure 4-13 and Figure 4-14 (equivalent to, but modified from Figure 4-12). The objective is to deploy the implementation strategies (lettered boxes) to the event-brokers (numbered ovals) that receive the original events from the sensors (Pub_A and Pub_B on the left). The deployment sequence is detailed below.

```
<MessageBundle>

 <Subscription>
   <ac name=”uid” type=”string” op=”EQ” value=”(A.x-B.x)^2 +(A.y-B.y)^2”/>
   <ac name=”distance” type=”double” op=”LE” value=”10”/>
 </Subscription>

 <ActiveSubscription>
    <ImplementationStrategy>

       <DeploymentSlip>
         <ac name=”uid” type=”string” op=”EQ” value=”A”/>
         <ac name=”X” type=”long” op=”ANY”/>
         <ac name=”Y” type=”long” op=”ANY”/>
       </DeploymentSlip>

       <ImplementationClass name=”ISCA.Agent.LocationObserverAgent”/>

       <InitializationParameters>
         <uid value=”A”>
         <partnerUid value=”B”>
         <lowRange value=”8”>
         <highRange value=”10”>
       </InitializationParameters>

    </ImplementationStrategy>

    <ImplementationStrategy>

       <DeploymentSlip>
         <ac name=”uid” type=”string” op=”EQ” value=”B”/>
         <ac name=”X” type=”long” op=”ANY”/>
         <ac name=”Y” type=”long” op=”ANY”/>
       </DeploymentSlip>

       <ImplementationClass name=”ISCA.Agent.LocationObserverAgent”/>

       <InitializationParameters>
         <uid value=”B”>
         <partnerUid value=”A”>
         <lowRange value=”8”>
         <highRange value=”10”>
       </InitializationParameters>

    </ImplementationStrategy>

    <ImplementationCode> hex code here </ImplementationCode>
 </ActiveSubscription>

</MessageBundle>
```

Figure 4-11 Proximity Active Subscription Sample

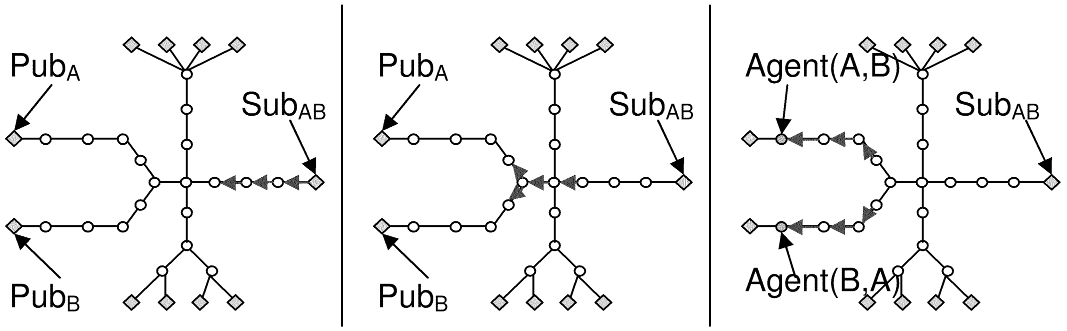


Figure 4-12 Deployment of Active Subscription. Each frame represents three steps.

1. Remember, advertisements for A and B data were previously forwarded through event-brokers 2 and 3, respectively, leaving copies of the advertisements at each broker. Additionally, these advertisements were memoized and given cacheId's of [AdvA] and [AdvB], respectively. Advertisements are drawn adjacent to the event-brokers in which they are stored to indicate the connection over which they arrived. Subscripts are used to indicate the hopCnt as maintained in the associated message context. The memoized versions of the advertisements are not explicitly shown, but run with the primary advertisement copy.

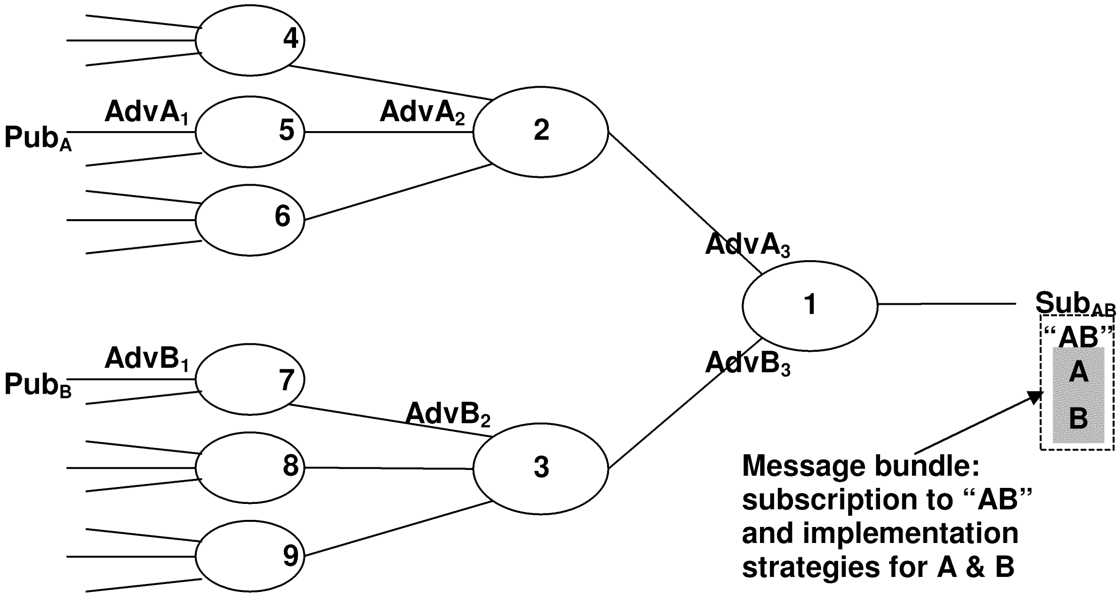

4
PubA
AdvA1
5
AdvA2
2
6
AdvA3
1
SubAB
"AB"
A
B
AdvB3
PubB
AdvB1
7
AdvB2
8
3
9
Message bundle: subscription to "AB" and implementation strategies for A & B

Figure 4-13 Proximity Agent Deployment (Before)

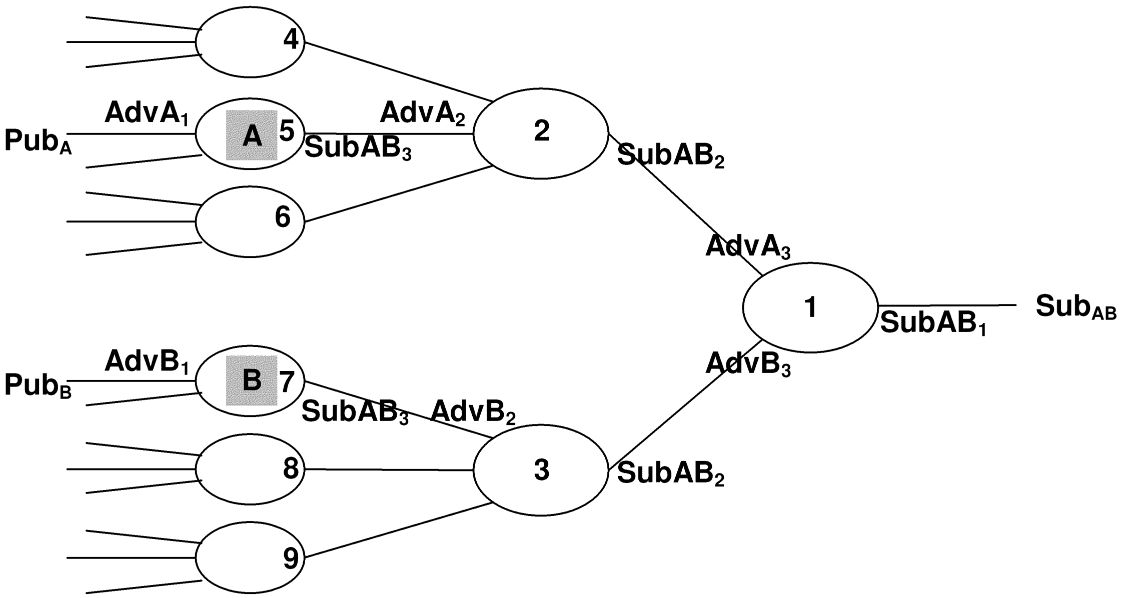


Figure 4-14 Proximity Agent Deployment (After)

2. On receipt of the ISCA message bundle at event-broker #1, the broker stores both the subscription and active subscription information. Subscriptions are drawn adjacent to the event-brokers in which they are stored to indicate the connection over which they arrived. Subscripts are used to indicate the hopCnt as maintained in the associated message context. Additionally, the first step of the distributed memoization process for this subscription occurs. The N-attribute subscription is converted into a single-attribute subscription that is also stored in the local database, with a links to the message bundle.

```
<Subscription>
  <ac name="cacheId" type="string" op="EQ" value="[SubAB]"/>
</Subscription>
```

   This secondary subscription will later be matched against an express forwarding slip.

3. Event-broker #1 determines whether to forward the message bundle to broker #2.

   a. First, it fails to match the relationship subscription.

   b. It then checks to match the deployment slips within the list of implementation strategies as if they were normal subscriptions. The first deployment slip matches the advertisement for A data. No contextual filters were specified and the advertisement’s message context indicates hopCnt=3, so forwarding needs to occur. To take advantage of system knowledge, we can now create an express forwarding slip to reduce the matching costs through the rest of the system. The cacheId for the matching A advertisement is inserted into an express forwarding slip

   ```
   <ExpressForwardingSlip type=”subscription”>
     <ac name=”cacheId” type=”string” op-“EQ” value=”[AdvA]”/>
   </ExpressForwardingSlip>
   ```

   and attached to the message bundle, which is then forwarded to event-broker #2. Once the message bundle is forwarded, no further matching is necessary.

4. Event-broker #1 determines whether to forward the message bundle to broker #3.
   a. First, it fails to match the relationship subscription.
   b. Next, it fails to match the A deployment slip.
   c. The next deployment slip matches the advertisement for B data. No contextual filters were specified and the advertisement’s message context indicates hopCnt=3, so forwarding needs to occur. Again, we take advantage of system knowledge by creating an express forwarding slip to reduce the matching costs through the rest of the system. The cacheId for the matching B advertisement is inserted into an express forwarding slip

   ```
   <ExpressForwardingSlip type=”subscription”>
     <ac name=”cacheId” type=”string” op-“EQ” value=”[AdvB]”/>
   </ExpressForwardingSlip>
   ```

   and attached to the message bundle, which is then forwarded to event-broker

#3. Once the message bundle is forwarded, no further matching is necessary.

We now effectively repeat the above sequence for event-brokers #2 and #3.

5. Event-broker #2 repeats step 2.
6. Event-broker #2 determines whether to forward the message bundle to brokers #4, #5, and #6. It fails for brokers #4 and #6, so we focus on the behavior of #5.
    a. First it checks its express forwarding slip for a match. This matches the stored, memoized copy of the A advertisement. No contextual filters were specified and the advertisement's message context indicates hopCnt=2, so forwarding needs to occur. The whole message bundle is immediately forwarded and no further matching is necessary.
7. Event-broker #3 repeats step 3.
8. Event-broker #3 determines whether to forward the message bundle to brokers #7, #8, and #9. It fails for brokers #8 and #9, so we focus on the behavior of #7.
    a. First it checks its express forwarding slip for a match. This matches the stored, memoized copy of the B advertisement. No contextual filters were specified and the advertisement's message context indicates hopCnt=2, so forwarding needs to occur. The whole message bundle is immediately forwarded and no further matching is necessary.

We now effectively repeat the above sequence for event-brokers #5 and #7.

9. Event-broker #5 repeats step 2.
10. Event-broker #5 determines whether to forward the message bundle to along its various connections, including to Pub$_A$. It fails for all other connections, so we focus on the behavior with respect to the Pub$_A$ connection.

11. First it checks its express forwarding slip for a match. This matches the stored, memoized copy of the A advertisement. No contextual filters were specified but the connection is detected as not accepting subscriptions or active subscriptions and the advertisement's message context indicates hopCnt=1 as shown in Figure 4-15 indicating that the advertisement comes from a publisher directly.

```
<MessageBundle>
   <MessageContext>
      <av name=”hopCnt” type=”long” value=”1”/>
      <av name=”arrivalTime” type=”long” value=”12345678”/>
      <av name=”travelTime” type=”long” value=”0”/>
   </MessageContext>
   <Advertisement>
      <ac name=”uid” type=”string” op=”EQ” value=”A”/>
      <ac name=”X” type=”long” op=”ANY”/>
      <ac name=”Y” type=”long” op=”ANY”/>
   </Advertisement>
<MessageBundle>
```

Figure 4-15 Publisher Local Data - HopCnt=1

Therefore, the event-broker instantiates the specified proximity agent instance and initializes it with the parameters provided

12. . Event broker #7 behaves equivalently with respect to the B advertisement.

4.2.3 Deployment Wrapper

After the implementation strategy arrives at the edge-broker, we must apply it. We do this through a *deployment wrapper* – an internal class object used by the event-broker. It serves three purposes. First, it is used to instantiate an implementation strategy instance, an agent. Second, it isolates the user-code from the rest of the system, so as to be able to provide security, etc., which we have not discussed yet. Third, it makes the user-agent transparent to the event-broker. That is, to the event-broker, the agent looks identical to any other pub / sub client. The deployment wrapper and these key relationships are detailed in Chapter 6 and shown in Figure 6-3.

4.2.4 Agent Subscriptions to Raw Location Data

Once deployed and activated, the agent code will generate subscriptions to the local data identified in their initialization parameters. (I.e., the agent is coded to convert the initialization parameters into a subscription.) Agent(A,B) subscribes to Pub$_A$ data

```
<Subscription>
  <ac name=”uid” type=”string” op=”EQ” value=”A”/>
  <ac name=”X” type=”long” op=”ANY”/>
  <ac name=”Y” type=”long” op=”ANY”/>
</Subscription>
```

Agent(B,A) subscribes to Pub$_B$ data.

```
<Subscription>
  <ac name=”uid” type=”string” op=”EQ” value=”B”/>
  <ac name=”X” type=”long” op=”ANY”/>
  <ac name=”Y” type=”long” op=”ANY”/>
</Subscription>
```

By subscribing to the publisher location data at the publisher’s node at the edge of the network, we logically filter it from the remainder of the network.[41] These subscriptions are, not coincidentally, identical to the deployment slips found in the active subscription shown in Figure 4-11. To maintain separation of concerns and allow the agents to be completely independent from their deployment, these subscriptions do not directly use the deployment slip. In Chapter 5, we show a traffic routing example where deployed algorithm components subscribe to different data than that used in the deployment slips.

Subscribing to data from the network edge does not ensure the data to be received arrives from the adjacent publisher. Therefore we need to filter based on the context of the message itself (i.e., its metadata) as first exemplified in Figure 4-8. We elaborate on the *message context* under the concept details. For now, the key piece of information

[41] Because the agents are hosted on the edge-brokers, there are no message sequences to present.

required is the hopCnt – i.e., how far has the message traveled through the system. When the hopCnt = 1, then the data has arrived directly from a publisher.

```
<MessageContext>
  <av name=”hopCnt” type=”long” value=”1”/>
</MessageContext>
```

Although this is metadata, we can treat is as regular data – the context becomes content. Now, if we bundle a *message context filter* with a subscription, as shown in Figure 4-16, then we can limit the receipt of data to that which originates locally.

```
<MessageBundle>
  <MessageContextFilter type=”incoming”>
    <ac name=”hopCnt” type=”long” op=”LE” value=”1”/>
  </MessageContextFilter>
  <Subscription>
    <ac name=”uid” type=”string” op=”EQ” value=”A”/>
    <ac name=”X” type=”long” op=”ANY”/>
    <ac name=”Y” type=”long” op=”ANY”/>
  </Subscription>
</MessageBundle>
```

Figure 4-16 Message Context Filter Sample

The filter is logically concatenated with the subscription as a regular conjunctive attribute comparison. Each time a notification matches the subscription, the message context associated with the notification is also checked against the message context filter to ensure the data originates locally.

As a performance enhancement measure, when Fulcrum sees a message context filter like this, it limits propagation of the message bundle to only those event-brokers that might produce satisfying data (i.e., message context data that will satisfy the message context filter). Each time the message bundle is forwarded, the hopCnt is reduced until it reaches zero (0), at which point the message is no longer forwarded. The message context and message context filter are described in more detail in the concept details section.

4.2.5 Agent Communication Setup

The communications pathways between the agents occurs next, as described in Section 4.1.4. Conceptually, Agent(A,B) subscribes to Agent(B,A) data and vice versa while avoiding the network flooding costs normally incurred. Each agent constructs a message bundle composed of a routing slip and subscription as follows:

Agent(A,B) constructs a subscription-based routing slip to Pub$_B$ data

```
<ISCAMsg>
  <RoutingSlip type=”subscription”>
    <ac name=”uid” type=”string” op=”EQ” value=”B”/>
    <ac name=”X” type=”long” op=”ANY”/>
    <ac name=”Y” type=”long” op=”ANY”/>
  </RoutingSlip>
  <Subscription>
    <ac name=”rangeRing” type=”string” op=”EQ” value=”AB”/>
    <ac name=”range” type=”double” op=”ANY”/>
  </Subscription>
</ISCAMsg>
```

Agent(B,A) constructs a subscription-based routing slip to Pub$_A$ data.

```
<ISCAMsg>
  <RoutingSlip type=”subscription”>
    <ac name=”uid” type=”string” op=”EQ” value=”A”/>
    <ac name=”X” type=”long” op=”ANY”/>
    <ac name=”Y” type=”long” op=”ANY”/>
  </RoutingSlip>
  <Subscription>
    <ac name=”rangeRing” type=”string” op=”EQ” value=”AB”/>
    <ac name=”range” type=”double” op=”ANY”/>
  </Subscription>
</ISCAMsg>
```

As with the deployment slips, these routing slips will chase the A and B advertisements back to the event-broker immediately adjacent to Pub$_A$ and Pub$_B$, respectively. At each event-broker along the path, the rangeRing subscription will be stored for future use.

Implementation strategy A subscribes to B notifications using a routing slip addressed with B’s sensor identification (provided in the initialization parameters).

Because the A and B sensors advertised themselves and the advertisements were flood-forwarded, the advertisement exists at the A and B event-brokers. By reusing existing advertisements, and not generating a new advertisement for each of Agent(A,B) and Agent(B,A), we avoid the O(| network |) cost of flood-forwarding. Connectivity is thus achieved without burdening the network with additional advertisements. The message flow is shown in Figure 4-17.

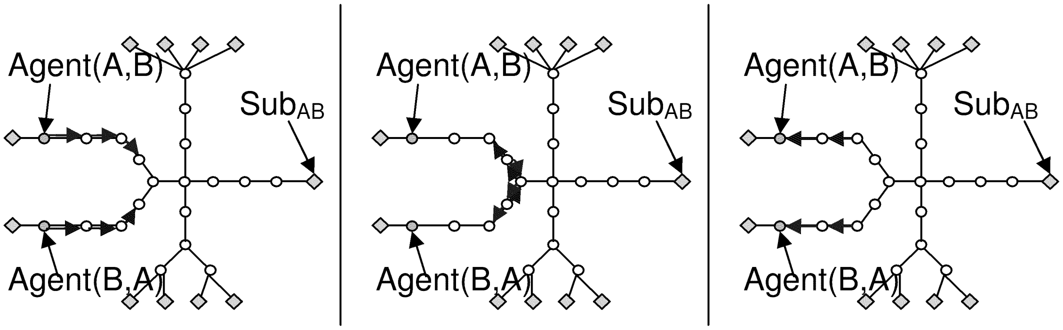


Figure 4-17 Agent Communication Setup Using Routing Slips

The routing slip from Agent(A,B) (green) references Pub$_B$ event types while the routing slip from Agent(B,A) (blue) references Pub$_A$ event types. As a consequence of bundling the subscription and routing slip together, a subscription trail from Agent(A,B) to Agent(B,A) and from Agent(B,A) to Agent(A,B) is left at the event-brokers along the way.

As before, when this associated subscription is deployed into the network, each event-broker will create and store a secondary, memoized subscription as follows:

```
<Subscription>
  <ac name=″cacheId″ type=″string″ op=″EQ″ value=″[cacheId]″/>
</Subscription>
```

4.2.6 Agent Synchronization

Events are generated by the publishers and received only at the agents. When synchronization is required, the agents will generate collaboration notification events that will flow only to their counterparts, as shown in .

```
<Notification>
     <av name=”rangeRing” type=”string” value=”AB”/>
     <av name=”id” type=”string” value=”A”/>                 (or B)
     <av name=”X” type=”long” value=”123456”/>               (any #)
     <av name=”Y” type=”long” value=”123456”/>               (any #)
     <av name=”range” type=”double” value=”123.321”/>        (any #)
</Notification>
```

Figure 4-18 Collaboration Notification Example

When this notification is matched at the edge-broker, an express forwarding slip is created as part of the distributed memoization technique and attached to the notification.

```
<ISCAMsg>
  <ExpressForwardingSlip>
    <av name=”cacheId” type=”string” value=”[cacheId]”/>
  </ExpressForwardingSlip>
  <Notification> … </Notification>
</ISCAMsg>
```

From now on, all subsequent event-brokers first match the express forwarding slip to determine forwarding. By only checking one attribute, cacheId, we are able to speed these coordination / synchronization messages through the system.

4.2.7 Proximity Notification

When the proximity relationship occurs, one agent generates a notification to indicate the relationship has been satisfied.

```
<Notification>
   <av name=”proximity” type=”string” value=”AB”/>
   <av name=”distance” type=”double” value=”9.875”/>
</Notification>
```

This notification flows back to the original subscriber, using the same distributed memoization / cache process. At this point, “backoff” strategies are employed by the agents (as coded into the algorithms as discussed in Chapter 5), to avoid a continuous stream of events while the two publishers remain in close proximity. Alternatively, SubAB could unsubscribe to tear-down the agent setup.

4.3 Concept Details

Several of the concepts are not exposed by the proximity example. And, there is much more to the other concepts than was reasonable to present in either the overview or proximity example sections. We remedy this with an in-depth presentation of those issues here.

4.3.1 Active Subscription Deployment

We have seen how to set-up efficient communications using a routing slip, where a subscription is deposited at each event-broker that the message bundle visits. This avoids having to flood the network with an advertisement for one subscriber to reference. By bundling the relationship subscription with the active subscription, we have used the same concept to deploy the relationship subscription at the same time as we deploy the user code.

One question is whether or not we lose anything by not broadcasting that the relationship event is generated. The individualize nature of ISCA relationships is such that the small amount of reuse lost is a fair trade for the costs that are saved. The same is true for the collaboration events passed between the distributed algorithm components.

4.3.2 Explicit Controls

The beauty of CBPS is that one need only submit an attribute-constraint list in order to receive all data of interest without regard to where the data originates. However, in using CBPS for the core communications infrastructure, we discover there are contextually relevant behaviors we simply cannot accomplish or which result in costly system behaviors. Therefore we need to provide the ability to have more explicit control over message distribution for these situations.

We provide this in two ways. First, we allow filtering based on the context of the message itself. This takes the form of providing a *message context* and a *message context filter*. Second, we allow direct control over the flood-forwarding decision.

4.3.2.1 Contextual Filtering

To clarify the purpose of message context filtering, we provide two small motivating examples. Messages are representative of the real world. As such, they need to support certain real world, contextually relevant, behaviors.

As one example, a user might need to limit information (advertisement, subscription, or notification) to be no more than N hops, based on known locality properties of the system. (E.g., an implementation strategy is only meant to process the local sensor data as shown in Figure 4-15.) We could then limit the incoming notifications to only those which come directly from the sensor by setting a constraint on the hop count to be equal to 1. Similarly, there is no need to have the outgoing subscription propagated into the rest of the system if it can never be satisfied except at the local event-broker.

As a second example, a publisher may know that the data being produced is valid for one second only and no one should be allowed to receive the data after that time. Conversely, a subscriber may wish to not receive data older than five seconds.

A common feature of both these examples is a need for both an incoming filter and an outgoing filter. To achieve these controls in a consistent and extensible way that can be used to support advertisements, subscriptions, notifications, and active subscriptions, we first defined the message context (metadata) component and then a message context filter component, each of which may be added to a message bundle.

4.3.2.1.1 Message Context

To make contextual information about the messages passing through the system available, we provide a message context component and add it to the message bundle to be passed along with each message and to be updated accordingly as it travels. The message context component is an attribute-value list, equivalent to a notification. In essence, it is a notification about the message. The message context is normally generated by the system, but, to provide maximum freedom, it may be boot-strapped, if necessary or desired, by a user application. The context maintained by the system about a message component is currently as follows:

Origin – is the identity of the publisher responsible for the original data creation. If not specified, this defaults to the most recent publisher.

Sender – is the identity of the publisher most recently responsible forwarding the message. The sender gets updated on each receipt and is used as an identity by which to address the network connection.

HopCnt – is the number of hops that the given message has traveled through the system. It is initialized to 1 and is incremented on each subsequent hop.

ArrivalTime – is the timestamp at which the message arrived at an event-broker, currently in milliseconds (the Java hi-resolution time is not yet available). This will be used to compute how long the message took to be processed.

BrokerLatencyTime – is the end-to-end processing time within an event-broker. It is essentially current time – arrivalTime.

TravelTime – is the length of time the message has been in the system. It is essentially the sum over all brokerLatencyTimes. A small challenge is to estimate time over the network or in the queues.

The first event-broker to receive a message will check for the existence of a message context component. If none exists, then an entire set will be created with the defaults listed above. If (partial) information already exists, missing data is created with the defaults and existing data is subject to updates as described above. Message context filters can then be defined to limit distribution or reception of data.

4.3.2.1.2 Message Context Filter

The message context filter component is equivalent to an attribute-constraint list, like a subscription. In essence, it is a subscription to the message context. Multiple filters are allowed such that both incoming and outgoing controls may be applied. Each is annotated to indicate directionality. Two examples are shown below. Let them be related such that #1 is a notification that satisfies the subscription of #2:

```
ISCAMsg – #1
    Message Context
    Message Context Filter (type=outgoing)
    Notification

ISCAMsg – #2
    Message Context
    Message Context Filter (type=incoming)
    Message Context Filter (type=outgoing)
    Subscription
```

An outgoing filter is applied against the message context of its associated message.

For example, the message context of #1 is evaluated against the outgoing unfilter of #1. Thus, in the locality example of the publisher declaring the data as only being valid for subscribers within three hops, the message context is updated each hop and the filter is applied each time to ensure the message does not travel further than desired.

An incoming filter is applied against the message context of the message being matched. For example, after the notification of #1 is discovered to satisfy the subscription of #2, then the message context of #1 is evaluated against the incoming filter of #2. Thus, in the locality example of the subscriber declaring the data as only being valid if it originates within three hops, the message context is updated each hop and despite matching the subscription, it may not be forwarded.

Internal efficiencies can be achieved for metadata tags known to the system. For example, if we set the incoming filter associated with a subscription to limit data to notifications which arrives from within three hops, then we do not need to wait for the notification hop count to reach three before filtering it; we can instead reduce the acceptable hop count each time the subscription is propagated until such point as the constraint is impossible to fulfill – at which time the subscription need no longer be propagated. If the subscription does not exist at an event-broker, it cannot be matched, and thus, we inherently prefilter events – only potential matches are evaluated.

4.3.2.2 Flood Forwarding Control

As indicated earlier it is undecided whether flood forwarding advertisements or flood forwarding subscriptions is the better approach. We contend that for run-time efficiency, assuming publishers do not frequently come and go, advertisements are preferable. However, as previously discussed, the advertisement technique only provides constraints over a minimum set of information such that a subscriber is only guaranteed to receive data described as part of the “base class” of the message. Thus, a subscriber is not guaranteed to receive every notification that matches, only those whose “base class” matches. Therefore, there may be occasions where flood forwarding the subscription is desirable, despite its propagation cost. In short, which model to use depends on context.

Rather than having to choose one model or the other, we provide controls that allow the user to control which model is used as befits the context of use. This is handled by adding an optional floodForwarding flag to both advertisements and subscriptions. The default is that advertisements flood-forward and subscriptions do not. To override the default, the user can flag the advertisement as floodForward=false or flag the subscription as floodForward=true.

4.4 Summary

Internet-scale context-awareness (ISCA) applications demand the right information be delivered to the right person / device at the right time while avoiding information glut and without saturating the network or (low powered) end-user devices. The costs of computing relationships at either the end-client or at the 1st CN are unacceptable in context-aware environments, where there can be thousands of publishers (devices) publishing events at high rates. Ideally, the middleware would permit suppressing location events at their entry nodes, only forwarding those that could satisfy a derived relationship event. Efficiently achieving Internet-scale context-aware computing requires a level of expressiveness and information availability not previously found.

With only a few minor changes and without giving up the valuable separation of concerns of CBPS, we are able to develop Fulcrum to leverage the CBPS content-based routing aspects to achieve our ISCA performance goals. Fulcrum is a context-aware publish / subscribe system that employs open implementation to give subscribers the ability to efficiently and transparently subscribe to relationships. That is, in addition to the ability to subscribe to derived relationships (where there is no publisher, per se) like Gryphon and Solar, the subscriber can specify an implementation strategy for the subscription. Because the subscriber is specifying the implementation strategy, the strategy can exploit domain-specific properties that the middleware could not know.

An implementation strategy is nothing less than a distributed algorithm for evaluating the subscription to detecting complex relationships, while reducing event traffic. The strategy component instances, like Solar *operators*, [CK-Solar-2002a] are first-class publish / subscribe Java applets deployed to the brokers where events first enter the network, using deployment slips, so as to immediately filter unnecessary events. Strategy instances are responsible for:

1. Using routing slips to efficiently setup communication pathways through the publish / subscribe infrastructure to enable collaboration via regular event notifications.
2. Subscribing to the raw event data comprising the relationship.
3. Publishing the derived relationship.

Note that the same property can be implemented by different strategies as appropriate to the context of use. Similarly, the same implementation strategy can be reused among properties with similar semantics (e.g., the notion of distance).

A strategy is kept separate from a subscription through a strategy design pattern and a factory for instantiating the subscription-strategy pair [GHJV-1995]. The separation provides several benefits, including the ability to:

1. parametrically substitute different implementation strategies for different environments;
2. permit non-expert programmers to specify an implementation strategy written by an expert programmer;
3. prototype an application's behavior by writing only its subscriptions, followed later by declaratively attaching an appropriate optimizing implementation strategy;
4. reuse an implementation strategy for subscriptions that are similar in structure.

We also provide three other small extensions. First, we introduce express forwarding slips to reuse event matching computations performed at the edge-brokers. Next, we increase information availability when we expose system context by extending all messages with information describing their passage through the network. This information is then available as additional content on which to filter a message using message context filters. Finally, we add explicit controls to control flood forwarding.

Taken together, these techniques allow us to treat the entire system – publishers, event-brokers, and subscribers – as a collaborative ecology of components that go beyond the local optimization limitations of black box information hiding and instead achieve global optimizations.

5.0 Efficient Distributed Algorithms for Context-Aware Relationships

One of the key questions regarding the Fulcrum approach to ISCA is whether it is expressive enough to effectively express the typical context-awareness subscriptions that we anticipate. To answer this question, we have developed solutions to three different context-awareness problems. Interestingly, the design and implementation of these solutions are stylistically similar and suggest an idiomatic approach that comprises a broad class of problems. Identifying the idiom makes it easier to solve new problems and enable reuse.

Evaluating subscriptions at the event-entry nodes is a necessary but not sufficient condition for the most efficient evaluation of relationship subscriptions. In particular, an efficient distributed algorithm for relationship detection must be implemented at the event-entry nodes and it must be tailored to the contextual needs of the user. This is the genesis for allowing users to encode domain-specific knowledge and push it into the network using open implementation strategies. In this chapter, we develop a style of implementation strategies that achieves efficient collaboration that is applicable to our motivating scenarios and we detail how to architect specific instances.

5.1 Overview

When an implementation strategy is deployed to the event-entry edge of the network, it is instantiated as a first-class pub / sub client, agent. In doing so, the agent is really serving as a proxy to the publisher (sensor) as well as being part of a distributed algorithm. Having the full expressiveness of a programming language, it is able to transform and filter event data in ways that a basic subscription cannot.

There are two core principles in the implementation strategies we have developed. First, we focus on the relationship, not on the raw event content of its constituent parts. Second, we look to find a way to place the information on a continuous number scale. These principles derive from the application of the "law of continuity".

The law of continuity states that for an entity to change from one state to another, all the intermediate states must be visited. We can use this principle for many context-aware relationships. For example, when concerned with proximity parole violations between a pair of gang members, we are first interested in the distance between them and only later interested in their actual locations. If we are concerned with which route to take on our daily commute, we are concerned with some fitness function of our choosing. Usually this is expressed as which route is fastest (all other costs being irrelevant). In most cases, we can represent the possibilities along a number scale. Similarly, energy consumption costs can be reduced to number scales.

Consequently, we can think in terms of distance between two alternatives along a number line – only the units change across domains (physical, financial, temperature, pressure, etc.). The distance is inherently continuous. Thus, we know that the distance must first shrink by half, by half again, and so on before a desired proximity can be reached.

This proximity behavior is a natural fit for many relational operators – for example, the proximity between two people or energy consumption monitoring. We will also show that problems with N elements can be decomposed to create relationships of pair-wise relationships. Examples include discovering when a specific group of people become proximate or selecting the best of N routes.

In the following sections, we detail two different pair-wise proximity algorithms. We consider how best to handle a collection of proximity requests with one common member (e.g., proximity to all my buddies). We then demonstrate the scalability of the techniques with more complex relationships. This includes providing two ways to implement group proximity (e.g., a busload of children on a field trip). Then we look at the complex problem of selecting the best traffic-route.

We show through our examples that all many problems possess the same basic characteristics. We demonstrate that one example is efficient and that the other algorithms effectively reduce to it Consequently, we can support a reasonable set of classes to make reuse easier. The performance metrics for select algorithms are presented in Chapter 7.

5.2 Pair-wise Proximity Range Ring Algorithm

From the law of continuity, we know that to go from one end of the scale to the other we must visit all the points in-between. Thus, the data must traverse a series of half-way points. This has a nice logarithmic property we intend to exploit.

We apply this to our desire to filter events with a distributed algorithm. Imagine the relationship we desire to achieve is the proximity of two people, say within 10.[42] We know that proximity is impossible until half the distance is traveled by one person. In fact, we recognize that a mirror image exists for both persons. Each may travel about freely until one of them crosses the midpoint. Consequently, we can create a bounding region around each person, a range ring, within which we know that proximity with the

[42] Feet, yards, or meters are irrelevant, provided that all data is converted into the right (e.g., same) units.

other person is impossible. Thus, all events that fall within that region can be filtered as unnecessary without requiring constant re-evaluation of the distance. Most importantly, these events can be filtered independently by sensor, despite the relationship requiring information from multiple sensors.

In our *Range-Ring Strategy*, these “half-way” points are the only events communicated between the collaborating implementation strategy instances at the event-entry nodes. Thus, as entities are far apart in their relationship, the data is reported infrequently and may be consider coarse grained. As the half-way points become small, data is reported more frequently with effectively finer granularity.

The algorithm sets up two range rings (in N-dimensions as fits the source data despite the distance being reduced to a single dimension). Each range ring is initially centered on the person under observation. The radius of each ring is set to the (initial distance – desired proximity) / 2. To this is added an invariant that the person under observation must always remain within their range ring. This is shown in Figure 5-1.

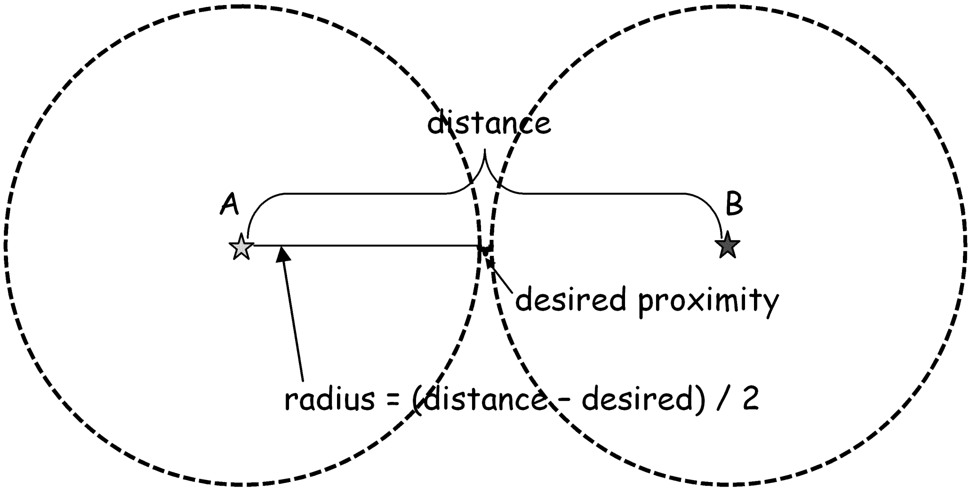


Figure 5-1 Range Ring Implementation Strategy Setup

Therefore we hope to achieve a reduction in the event traffic from O(c * |events|) to an expectation of c * lg_2(distance), where c is approximately 0.95 * diameter as previously computed to convert events to event hops. Noteworthy is the change in measurement units. O(|events|) refers to the original number of events; lg_2(distance) refers to the range that must be covered to achieve the proximity, which depending on the quality of the sensor, the reporting rate and the person's motion may be more or less than lg_2(|events|). This insulates the system from sensor changes. For example, replacing the old sensor with a new one that reports with more precision at higher data rates only increases the number of events filtered, because the only measurement that matters to the algorithm is distance away from one's origin. Figure 5-2 shows filtering of all events within the bounding region.

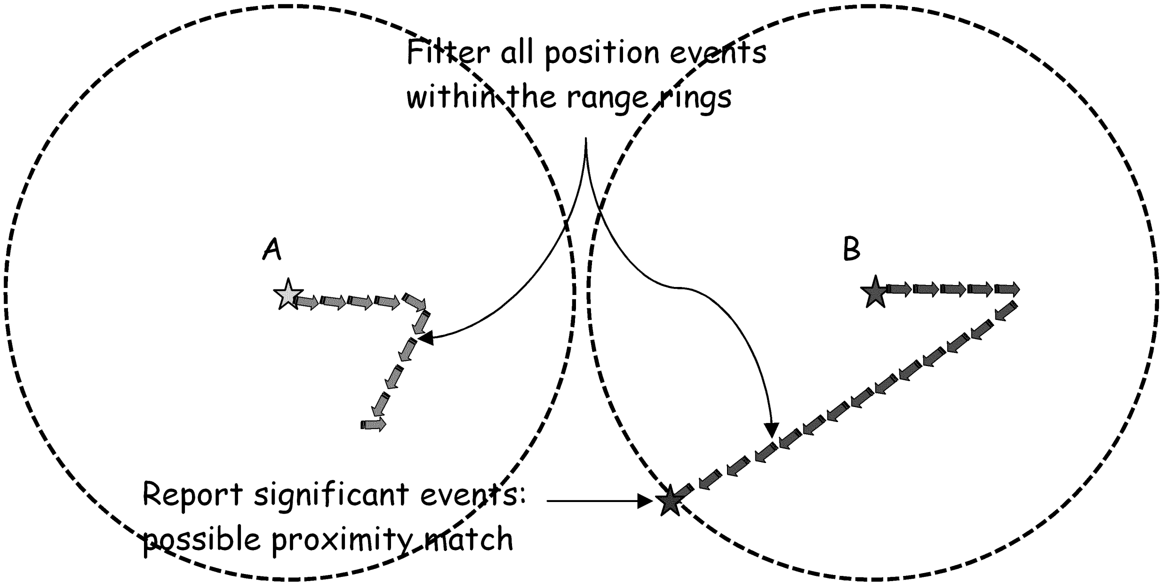

Filter all position events within the range rings
A
B
Report significant events: possible proximity match

Figure 5-2 Filter Useless Events. All events within the bounding region are filtered.

When one of the persons travels outside their assigned range ring (e.g., B) then the possibility of satisfying the proximity relationship exists. If so, then a notification is published, passed through the network, and received by the subscriber. If not, then we must take action to preserve the containment invariant, as depicted in Figure 5-3.

1. B's new range ring is computed based on the last reported position of A.
2. B's new location, the location used for A, and the new range ring are published as an aggregated event and delivered to A.
3. On receipt, A knows exactly what data was used in computing the new range ring and resets its internal variables accordingly (i.e., updating its suppression filter).
4. Finally, it is necessary for A's strategy to atomically check whether its current location is outside the new range ring. If so, the strategy behaves as if has just then moved outside its range ring and repeats the same process. It may follow-up with its own new range ring event to ensure preservation of the invariant.

The race condition of the users simultaneously leaving their range rings is handled in part by this atomic check, and in part by the strategies keeping the new position and the smaller of the two rings that are exchanged.

An essential component is that the instances remain in synch by using common knowledge. It would not be possible to accurately determine proximity if the two instances were allowed to evolve independently.

We note that $\lg_2$(distance) is an expectation, not a guarantee. If A and B walked at equal speeds directly at each other, then after initial setup, only two reports would occur. If A were stationary and B walked directly toward A, we would achieve $\lg_2$(distance), independent of reporting rate. The worst case situation is when the motion is "follow the leader" where both persons maintain a consistent separation. In such a case the filtering only reduces the event traffic by a constant equal to the radius / (velocity / reporting rate). Under normal motion, we can only show empirical results as presented in Section 7.

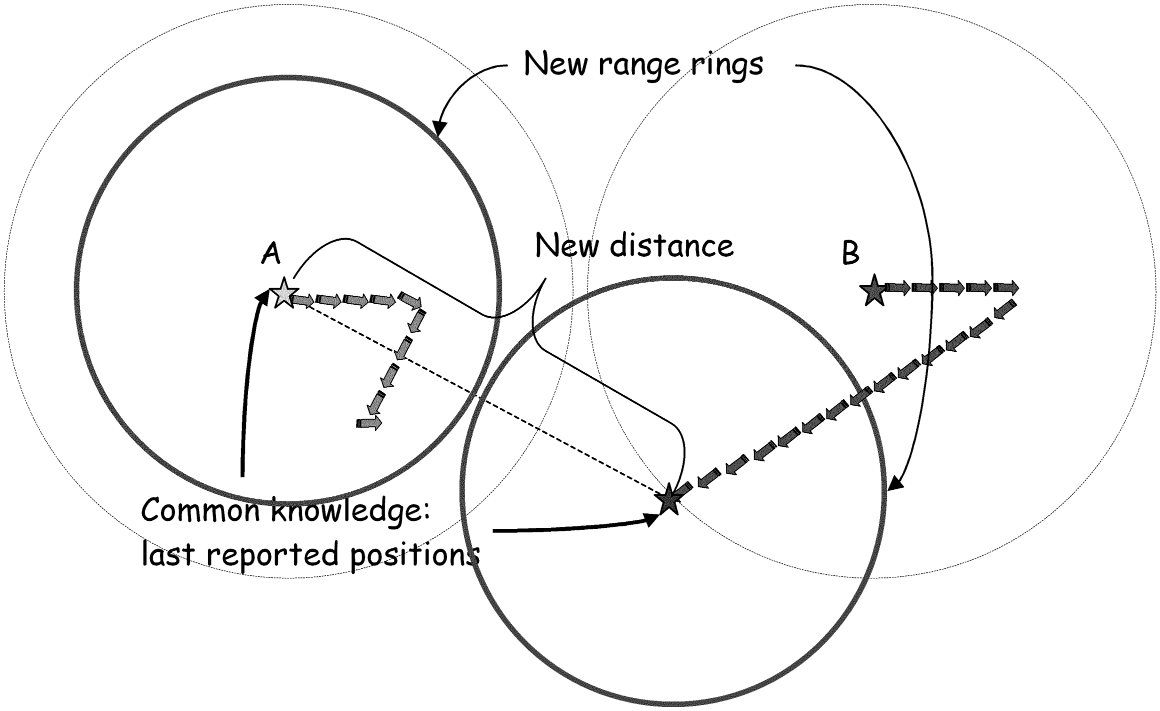


Figure 5-3 Resynchronize Distributed Algorithms.

For this range ring implementation strategy, it does not matter if the new distance is less than or greater than the old distance, each algorithm instance computes new range rings around the last reported positions to restore the invariant that a person is always within their range ring as shown in Figure 5-3. The algorithm assumes good connectivity and timely communications.

5.3 Parallel Line Algorithm

The range-ring is only one implementation strategy possibility.[43] Another would be a parallel line strategy. In this strategy, we draw a line, between the two people as before and then add two parallel lines in the middle that are perpendicular to the first as shown

[43] It was selected for its simplicity and ability to extend easily into N-dimensions.

in Figure 5-4. The distance between the parallel lines is the desired proximity. This strategy uses exactly the same principles as the range ring strategy. The algorithm then monitors for events revealing that their subject has crossed the line. Again the positions and distances would re-synch. The main difference from the range ring algorithm is that in the parallel line algorithm, the entities are allowed much greater freedom of movement (away from each other) at the cost of computing a different bounding region. The parallel line strategy also differs in complexity as we change the number of dimensions. In one dimension, we need only a point, in two, a pair of lines, in three a pair of planes, and in four or more dimensions, we need a hyper-plane.

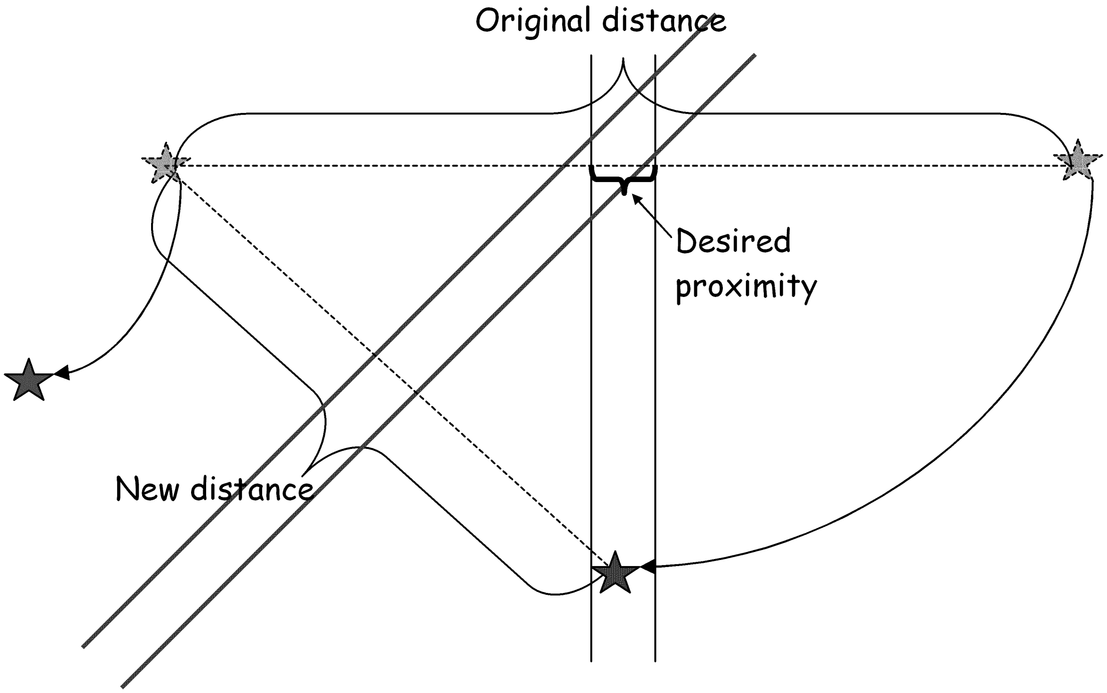


Figure 5-4 Parallel Line Implementation Strategy

5.4 Content-Volatility

One-half of the available separation is convenient when entities are equivalent.

However, we can define *content-volatility* as the risk times the magnitude of the information changing. In this case, risk is the likelihood of an update with changed data, for example a student traveling between classes has a risk factor near 1.0, while a student in class has a risk factor near 0.0. Magnitude is the expected size of the data change. For example, if we have one person on a bicycle and another on foot, then the magnitude for the cyclist is perhaps three-times that of the pedestrian. Consequently, when determining bounding regions, we can scale each region to reflect its subject's content-volatility as a means of further reducing event messages, if only by a constant factor. For example, with a distance of D, the radius of one ring might be 0.9D and the other might be 0.1D. In essence, content-volatility is metadata about the sensor data.

5.5 Avoiding Redundant Notification Race Conditions

One of the strengths of CBPS is that an event satisfying multiple subscriptions is reported only once. In creating mechanisms, especially with symmetric distributed algorithms, where multiple conditions can be met effectively simultaneously, then we run the risk of reporting the satisfaction of a relationship from multiple starting points.

A simple solution that satisfies most uses is to arbitrarily select the instance responsible for reporting satisfaction through some internal criteria, such as the lowest lexicographically unique identifier, or through an applet parameter explicitly specifying responsibility. Then the algorithms incorporate a design pattern where they communicate with each other the satisfaction of the relationship and then only the one item reports the result to the user.

This may incur a cost of twice the latency of the longest path across the network, in the case there the relationship satisfaction is discovered at the user's node but the reporting responsible entity is on the far side of the network.

To avoid the latency costs, one can trade off the report-only-once requirement, and let the user's system receive multiple notifications of the same relationship satisfaction and filter out duplicates itself. This risk of receiving multiple events or else paying the latency cost is a tradeoff that the user client can make by selecting the appropriate implementation strategy – putting the control into the hands of the user.

5.6 Scalability

Scalability has been evaluated in terms of reducing event hops and the architectural concerns to avoid bottlenecks and perform distributing event processing. We must also ask some questions in terms of whether the implementation strategies can do more than previously demonstrated. Can we achieve the kinds of expressiveness desired?

1. How does one efficiently support multiple, but similar, relationships.
2. Can we support more complex relationships?
    a. Can we support a larger number of publishers in a relationship?
    b. Can we support a wider class of problems?

5.7 Multiple Pair-wise Relationships

The proximity of gang members is an $O(N^2)$ problem with respect to the size of the gang. Buddy proximity is similar without guarantees of cliques. Even when they do occur, cliques of friends may be bridged by highly social individuals. How should these be handled?

In part they need to be handled with respect to the task, the data rates, and the ability or desire to impose centralized control. To monitor the proximity of gang members, one could create a central authority to perform the computation, but if we are to share a public infrastructure,[44] at what point should it change from a distributed to centralized architecture. In the wild of the Internet, centralizing control over buddies is infeasible. What techniques are appropriate to reduce the burden?

A naïve approach is to use an implementation strategy associated with each user that monitors relationships with all of one's associates and when an update need is required, broadcast the entity's new location to allow everyone to re-synchronize. The failure of this approach is demonstrated in Figure 5-5.

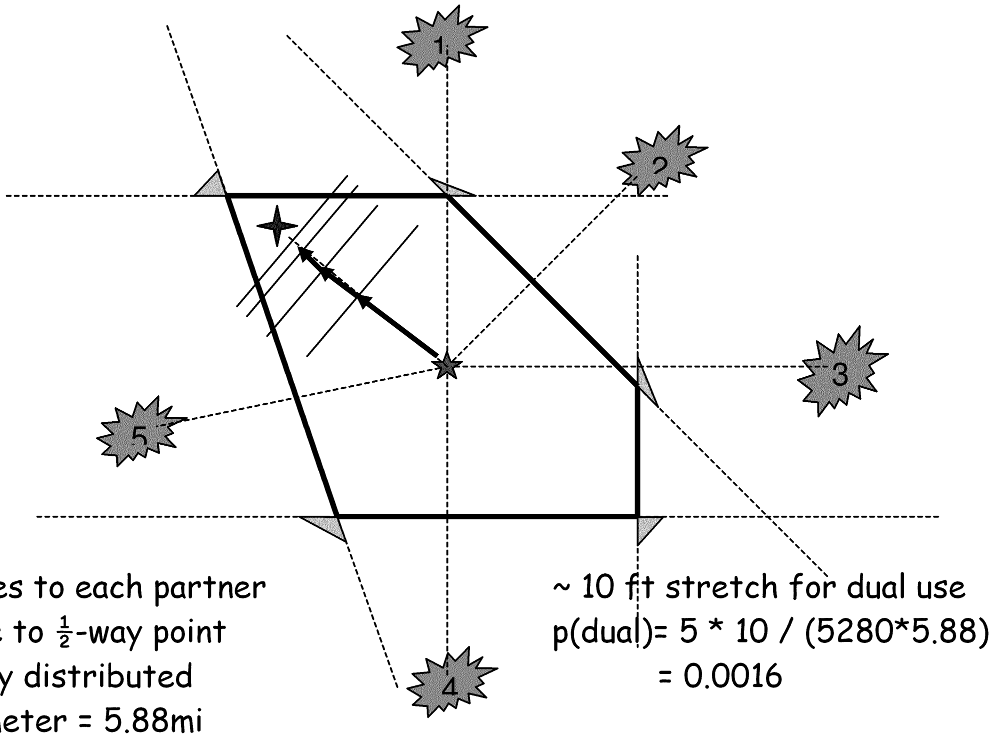


Figure 5-5 Multiple Proximity Relationships

[44] And we evaluate the solutions solely on technical merits and ignore the various political ramifications.

The central red star represents an instance of the algorithm. The blue star and the other numbered stars are the partners in proximity relationships. Using a multi-user extension of the parallel line algorithm, we can create a series of bisections between the user and each partner. The little green triangle represents the space into which the user would have to walk in order to have dual use in sharing his new position with his partners.

To simplify our computations we say each numbered partner (1-5) is two miles distant, making it one mile to the half-way point. Assume the partners are evenly distributed, so the bisection line between two sets of relationships crosses at a 72-degree angle with a perimeter of 5.88 miles. And let the primary user be walking at about 4.4 feet per second (i.e., 3 mph) with a reporting rate of 1Hz. Consequently, the dual use zones are approximately 10 feet along the edge of our pentagram. Thus the probability of passing into a dual zone is

(5 zones * 10 ft / zone) / (5.88 miles * 5280 ft / mile) = 0.0016

This means that broadcasting an update, received by all partners, is only useful to two of them about 0.16% of the time.[45]

A second look at the problem encountered with this broadcast style is shown with the partner represented by the blue star, who is within the bounds created by the other partners. In such a case, each half-way report published as the user approaches this partner is broadcast to all others.

Based on data taken from ActiveCampus experiments in 2001, the number of buddy relationships ranged from 1 to 159 with an average of 3.37. Thus, the example provided

[45] It would be more useful if distances were shorter or more relationships existed

above is close to experiential data. We see then that the N^2 concern is mitigated and the number of events per person dominates. Even when there is a common desire to know about a single participant, the relationships are disjoint in a way that maintaining each relationship independently from the others is generally going to be more effective.

If we had a large clique, such as a gang to monitor for parole violations, and the ability to impose centralized control, then we might make a case for aggregation at the 1st CN. The number of relationships would be N (N-1) / 2. If the pair-wise algorithm reduces the number of events per relationship from O(|events|) to about lg_2(distance), it may be possible that (N (N-1) / 2) * lg_2(average distance) exceeds N * events per person, in which case from an event hop count standpoint, the centralized approach would be better.

5.8 More Complicated Relationships

Relationships become more complicated as we increase the number of participants, compose relationships, or include different behaviors. As one example of increasing participants, we can extend proximity to groups. How can Fulcrum efficiently determine that a busload of school children were all properly grouped (e.g., no child has wandered off)? As another example exhibiting all three complications, imagine we want to support context-awareness with respect to traffic routing selections. We must compare the travel time of two (or more routes); the route times are composed from the travel time across each segment of the route; and the travel times are determined by transforming sensor speed.

The techniques described thus far are fitting for implementation strategies designed to directly interact among all components. Our original proximity algorithm was written such that each component directly communicated with its partner. This tight binding provides an efficient solution; however, the tight binding makes it difficult to scale in two key dimensions; the algorithm would be required to interact with all its partners, and the cost of communicating changes across all partners would become prohibitive.

Therefore, we decouple the distributed algorithms by placing an observer / mediator implementation strategy component between the basic implementation strategy components to make the techniques generally applicable and scalable. This decouples the individual components from each other with a single component responsible for efficiency decisions. Under normal observer / mediator usage, the observed component need not be aware of the controller if the controller can unobtrusively listen for events. This is a fitting description for our environment.

However, as identified previously, we want to keep each relationship separate. Yet, we also know there will be situations where the raw data is reused (e.g., gang member A has relationships with both B and C). In such cases it may be useful for the associated implementation strategy to service multiple relationships. Therefore, we associate a relationship identifier with our data to setup efficient communications within the CBPS environment.[46] In this way, we are able to setup unique subscriptions per relationship.

Finally, we can replicate this pattern to form a hierarchy of relationships to give us greater flexibility and power, as demonstrated in the traffic routing example. In the following sections we describe the implementation of several complex relationships,

[46] In that the controlling observer / mediator (OM) is often representative of the relationship, it is convenient to use the OM's identity as the relationship identity.

describe the components used to achieve these implementation strategies, and discuss the underlying techniques and lessons learned.

5.8.1 Center-of-Mass Proximity Algorithm

Center of mass proximity is useful for a known group of entities, say a busload of school children on a fieldtrip. There are two forms for this, the first is that the entire group must be present and the second is that some subset, say a quorum, is present. Although we have increased the number of participants, we will show that center of mass proximity can be handled in much the same way as we consider the proximity of two entities. Two possible implementation strategies will be discussed.

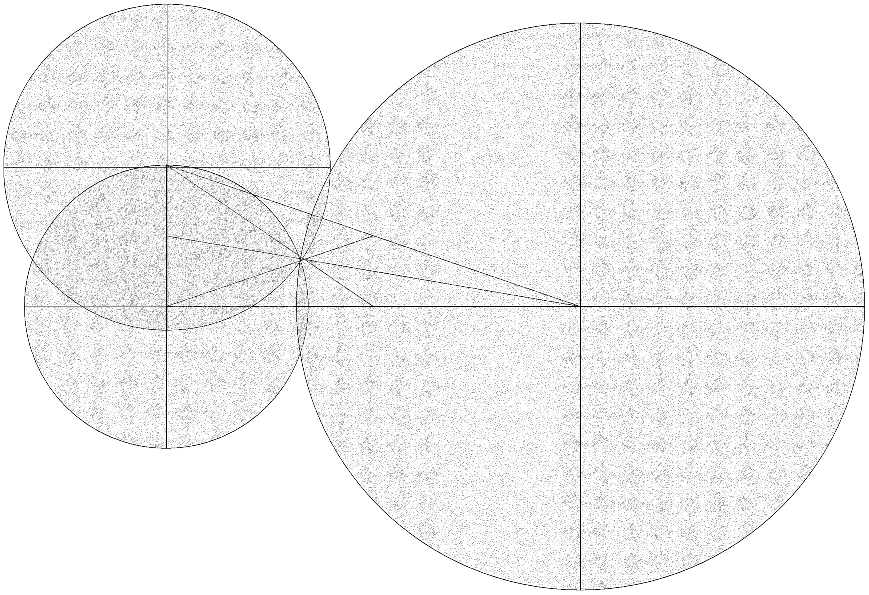


Figure 5-6 Center of Mass Proximity

Mediated Approach. As shown in Figure 5-6 with three entities, the center of mass can be easily computed and a range ring identified for each entity that is their distance from the center of mass. Again, like two-person proximity, we may wish to give the center of mass an area (in 2D terms), such that the radius is first subtracted. As before, in order to achieve center of mass proximity, at least one of the entities must pass beyond its bounding region.

We construct this strategy using the techniques discussed for pair-wise proximity. In particular, each entity is associated with an origin and a range ring. This time, when a participant exits their range ring, the report is sent to the observer / mediator. The observer mediator then manages all the range rings and updates associated participants accordingly. This is the goal of Gryphon's selective curiosity.

When a participant is currently within the center of mass, then we want to treat their range ring as what it would take to move them out of the center region.

The value in this approach is that only N + 1 implementation strategies need to be deployed into the system and the number of events are minimized for this function. The challenge is that this is a single purpose approach with a fixed set of participants. It does not leverage existing system information, such as outstanding buddy relationships, and is unlikely to be able to be leveraged by other tasks. In being single purpose for the entire group, it might be able to detect and report a few stray participants, but it is ill suited to detect a subset of the group, say 5 of N, 5 << N, as being co-located.

Counting Approach. This approach attempts to leverage existing implementation strategies. We assume that pair-wise proximity relationships already exist among all parties. Then, we introduce one additional implementation strategy, the counter, and initialize it with the list of people in the group to detect and deploy it to the location of a given individual. The counter then subscribes to the pair-wise proximity reports generated for that person with respect to each person in the group. We add the requirement that a pair-wise proximity will also report when proximity has been terminated. When all pair-wise proximities have been reported and counted, then the group proximity report is generated.

To perform subset detection, we can deploy one counter to each participant. Through an agreed upon method similar to that described with pair-wise proximity, only a single event is generated for any given group.

The value is that we can leverage existing deployed strategies and introduce only one small additional piece of code into the network for a complete group or N for the entire group. In the worst case, all the pair-wise proximity and counting strategies could be deployed and the system could be set to avoid duplicates.

The challenge, especially 'in the wild', is to collaborate with the existing pair-wise proximity strategies; they may use different sizes (e.g. one pair has a proximity of 10 and another of 25); they may not normally report proximity termination. Although the possibility exists, leveraging and coordinating existing deployed components would be a challenge.

5.8.2 Traffic-route Monitoring Algorithm

Major cities are criss-crossed with highways and primary arteries allowing people a variety of ways to get from their homes to their jobs. In most cases there are two paths you can go by and usually more.

Using a hierarchy of observer / mediators, we can handle the evaluation of route choices which are relationships between relationships. First, each route is composed of multiple road segments (whose sensors report in units of miles per hour); then the travel times along the routes must be compared to one another. This hierarchy maps well to a generalized tree structure, as shown in Figure 5-7. We will later map the tree to the structure of the overlay network to show how the implementation strategies and the control hierarchy are deployed and coordinate with each other.

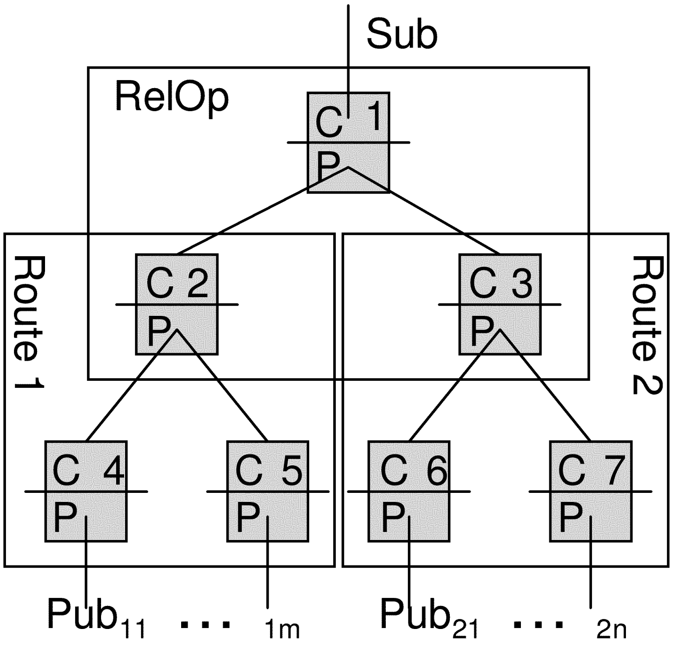


Figure 5-7 Complex Relationship Hierarchy – Traffic Routing Example

In the tree, every node (1-7) serves in both child (C) and parent (P) capacities. The root node serves in a child capacity with respect to the subscriber, while the leaf nodes serve in a parent capacity with respect to publishers. Any number of internal nodes may exist as necessary. The three rectangular containers show the logical (functional) groupings for Route1, Route2, and the relational operator that evaluates their differences.

Within the tree, parents subscribe to result information produced by their children; data flows up. Conversely, children subscribe to control information from their parent; control flows down. As shown, each deployed strategy element serves both capacities. The subscription, active subscription combination representing the relationship of one route being 5 minutes faster than the other might be described as follows:

Subscription: uid=Route1.vs.Route2, ctrluid=none, timeDelta>=5

ActiveSubscription:

```
Implementation Strategy - 4
  Deployment Slip = (uid, string, EQ, 4 )
  Imp Class = ObserverAgent.class
  Init Params: uid=4, ctrluid=Route1, ctrlAddr=4, scalar=12, reciprocal=true
Implementation Strategy – 5
  Deployment Slip = (uid, string, EQ, 5 )
  Imp Class = ObserverAgent.class
  Init Params: uid=5, ctrluid=Route1, ctrlAddr=4, scalar=23, reciprocal=true
Implementation Strategy – 6
  Deployment Slip = subscription (uid, string, EQ, 6 )
  Imp Class = ObserverAgent.class
  Init Params: uid=6, ctrluid=Route2, ctrlAddr=7, scalar=8, reciprocal=true
Implementation Strategy – 7
  Deployment Slip = (uid, string, EQ, 7 )
  Imp Class = ObserverAgent.class
  Init Params: uid=7, ctrluid=Route2, ctrlAddr=7, scalar=15, reciprocal=true
Implementation Strategy – 2
  Deployment Slip = (uid, string, EQ, 4 )
  Imp Class = CombineAgent.class
  Init Params: uid=Route1, ctrluid=Route1.vs.Route2, ctrlAddr=4, observers {4, 5}
Implementation Strategy – 3
  Deployment Slip = (uid, string, EQ, 7 )
  Imp Class = CombineAgent.class
  Init Params: uid=Route2, ctrluid=Route1.vs.Route2, ctrlAddr=4, observers {6, 7}
Implementation Strategy – 1
  Deployment Slip = (uid, string, EQ, 4 )
  Imp Class = RelOpAgent.class
  Init Params: uid=Route1.vs.Route2, ctrluid=none, observers { Route1, Route2 }
Imp Strategy Code – think JAR file
```

As before, each of the implementation strategies is deployed into the network based on the routing slip. The key differences from the proximity example are the hierarchical elements. We presume some locality between sensors so we want to forward-deploy the observer / mediator. In fact we can guarantee locality if we co-locate the observer / mediator with one of the sensors. All this is accomplished using a deployment slip. The data gathering agents subscribe to control messages using the ctrluid parameter, which essentially identifies their parent.

We map the implementation strategy agents to the event-broker nodes as shown in Figure 5-8. By choice, several controllers are co-located with data providers. Although the communications are completely transparent as they are handled entirely by the normal content-based routing mechanisms set up by our routing slips. This co-location choice minimizes inter-node events.

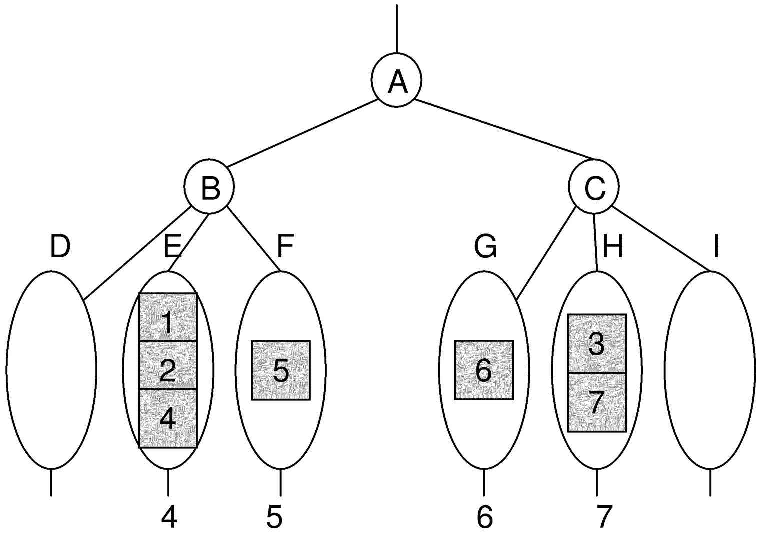


Figure 5-8 Map Implementation Strategy Instances to Event-Brokers

Observer agents 4-7 receive raw sensor data in miles per hour and transform it into travel time using the equation scalar / speed. The scalar value identified in the active subscription (12, 23, 8, 15), is passed to the agent as an initialization parameter; it represents 60 minutes per hour * distance of the road segment ‘controlled’ by the sensor (e.g., 60 min per hour * 0.2 miles = 12 minutes mph). This then is divided by the speed (mph) reported by the sensor to get travel time.

The newly computed time is compared with the last reported information and the bounds set previously by the controller (e.g., |oldTime – newTime| < bounds). If the data is out of bounds, it is published using the relationship identifier – in effect, it is sent up to a controller. Say the provider is 7, then the semantically significant data is passed on to the controller 3, to determine whether the combined data is within its tolerance, in which case it sends a reset message to a subset of its children, or else it sends an update to its controller (1). The message from 7 to 3 remains within the event-broker, while the message from 3 to 1 travels H-C-A-B-E.

5.9 Reusable Implementation Strategy Components

Both the Context Toolkit and Solar discuss widgets or operators that transform, aggregate, and filter. However, Solar uses a centralized 'star' to determine code deployment [CK-Solar-2002a] and the Context Toolkit does not provide such capability. And, for these systems, information context generally flows only one direction.

Fulcrum lets the system self configure (in that the deployment slip is used to identify where code should be deployed) and it uses the full extent of the CBPS infrastructure, plus the new routing slips, to provide bi-directional information and control flow between the collaborating algorithm components.

Through the development of several implementation strategies, we have discovered several common, reusable components described below.

5.9.1 Abstract Base Class for Several Implementation Strategy Components

The implementation strategies for our relationships have all been developed with a common set of behaviors. Using the model of the range ring over some numeric domain, each component has an origin and a range. The origin is typically the last reported (or assigned) position and the range provides the bounding region about the origin. All components then accept commands to reset their origin and / or their range. Generally the origin and range are set simultaneously. As initialization parameters, all components need to be assigned a unique identifier (uid) and receive a unique relationship identifier – usually set to the identifier for the controlling agency (ctrluid).

5.9.2 Observer / Transformer / Filter (OTF) Components

The most common implementation strategies are components that will be deployed to the edges of the network to listen for specific data, transform the data into a normal (or task-specific) form, and filter the data until some change threshold is reached. For example, a loop detector on a freeway is a sensor that reports traffic speed in miles per hour (e.g., 60 mph). To use that data to determine which route home is desired, we first normalize it into travel time over the stretch of road represented by the sensor (e.g., 0.2 miles @ 60 mph = 0.2 minutes). Then the travel time is evaluated against the boundary condition set by some controlling agency (e.g., 0.2 minutes (actual) < 0.25 minutes (threshold)) and discarded when within limits.

Typically, transformations follow a linear equation (e.g., y=mx + b). Therefore these coefficients may be provided as optional parameters labeled as *scalar* and *offset*. Additionally, as discovered in the freeway speed to travel time transformation, we needed the reciprocal of x and so have provided another optional initialization parameter called *reciprocal*.

The instances currently developed are tied to the specific data production (i.e., sensor id and data). However, they could be parameterized to accept as initialization parameters the identifying characteristics of the data to be observed. Also, we choose to use the range-ring strategy. As previously discussed, these components will accept messages, to reset their origin and / or range. We call this the OriginRange interface.

The three stages, observe, transform, and filter are implemented internally as separate functions or classes to enable reuse and evolution.

5.9.3 Aggregation / Filter (AF) Components

It is not enough to deal with relationships among individual data components; we find it necessary to aggregate results from multiple sensors as a cluster (e.g., a route is composed from multiple sensors). That is, instead of difference, we need summation, and again we need to be efficient. Luckily we are not looking for arbitrary, most recent summation, but summation with respect to a task. The task therefore creates either a specific target (e.g., price of energy consumption greater than ten cents per hour) or is involved in a higher order relationship where we can again examine the difference between values on a number line.

Therefore, the AF component can follow the same basic strategy of having an origin and a range and hence also implement the OriginRange interface. As a consequence, we can freely interchange OTFs and AFs as we would in any normal programming sense to create a hierarchy of collaborative filters.

Now though, the AF must manage the individual OTF components. The AF is inherently an observer / mediator. Children components are managed by giving each a fraction of the available range according to their content-volatility. As discussed in the techniques and lessons learned section below, we prefer a shallow hierarchy over a deep one so that the mediator, the AF, can act as a buffer to minimize the number of events generated.

5.9.4 Proximity Components

A proximity algorithm may be useful any time the data of interest can be reduced to the relationship between variables that are represented as numbers. It may physical, financial, temperature, pressure, etc. Changing scales between sensors is best handled using observer agents, with a scalar, and an observer / mediator component.

Pair-wise proximity detection was implemented using the range ring strategy. The same technique can be applied to groups, where a person's proximity to the center of mass is the key relationship. In such a case, the proximity manager representing the relationship would act in an observer / mediator pattern. The proximity manager could easily be adapted to report the identities of sensors that are not within the required proximity. Then a busload of school children might be more easily monitored.

5.9.5 Relational Operator Components

One of the most common event types comes from binary inequality equations (e.g., A < B). This was experienced as we evaluated traffic routing (e.g.,is route1 +5 < route2). The primary objects (e.g., route1, route2) may be OTFs or AFs because they implement the same interface. The offset (e.g., +5) meaning that route1 needs to be five minutes faster than route2, is provided as an initialization parameter, as are the primary object identifiers.[47] The relationship manager acts as an observer / mediator.

5.9.6 Minimum / Maximum Components

In attempting to determine which route home is best, it became clear that we really want the best of N routes home. In general, best could be the minimum (e.g., travel time) or the maximum (e.g., expected ROI over a set of investment possibilities). Thus, we let OTFs or AFs get the basic data necessary and instantiate a minimum (maximum)

[47] It should be noted that any value listed as an initialization parameter can also be reset. The deployed implementation strategy sets up a subscription for updates. This is most valuable in situations where control data fluctuates, such as currency exchange rates.

observer / mediator.

5.9.7 Implementation Strategy Techniques and Lessons Learned

In developing, deploying, or analyzing various strategies, several techniques were discovered and the following lessons were learned.

5.9.7.1 Back-Off Strategies

Because users are interested in being alerted to a possibility, but not continually harassed, upon satisfaction of the relationship, the deployed implementation strategy must either be torn-down or temporarily disabled. Yet, the possibility for which the subscription was originally set may not have been realized, in which case we want to keep the subscription in place to avoid the tear-down and setup costs. Thus, we want to temporarily disable the subscription in what we term a *back-off* strategy. For example, after a proximity relationship of 10 yards is achieved we avoid repeated notification by setting an implementation strategy specific back-off distance, say 250 yards. No more notifications will be sent until this separation has first occurred.

No additional messages are required as this is built into the implementation strategy. When the user or application is satisfied that the task has been accomplished, then we can unsubscribe to the relationship. This can be performed either by sending a message to disable the monitoring until further notice or to perform a complete tear-down of the deployed code.

In the traffic routing case, to be discussed shortly, a user wishes to know when one route is faster than another by more than a few minutes. This has a natural back-off strategy. Each time the relationship is satisfied,

e.g., time(route1) + 5 minutes < time(route2), then we want to know the opposite condition,

e.g., time(route2) + 5 minutes < time(route1).

When the traffic routing relationship is to know the best route among several alternatives, the relationship might be set such that we want to know when the next minimum travel time is 5 minutes less than the current time of the prior reported best route and better than all other route times.[48]

5.9.7.2 Observer / Mediator Pattern

The observer / mediator pattern is good in normal software development for the separation of concerns it provides. It performs equally well in this distributed environment. Using this technique we are able to easily add as many components to a relationship as necessary without altering each instance. And, we are able to use it to create hierarchies of relationship management. Together these features greatly enhance the scalability of our solution.

5.9.7.3 Asynchronous Control Over Managed Entities

In the original proximity strategy development, only two entities existed, such that immediately reporting bounding region violations directly to the partnering strategy worked well; the range rings were always synchronized and it mattered little whether we used the range-ring or parallel line strategy. However, when a large collection of sensors are being monitored to create a composite result (e.g., traffic routing), then synchronizing all components, using either range rings or parallel lines, require N entity change reports. This is excessively burdensome; it is as bad as reporting all messages to the end

[48] Systems performing collaborative radar tracking of aircraft determine reporting responsibility transition when one radar has a “track quality” (i.e., smaller area of uncertainty in a log scale) of two better than the current reporting unit.

subscriber. However, by letting the mediator manage each child independently, using separate one-to-one conversations, then the mediator can serve as a buffer. Thus, we can use the range-ring strategy and when a child moves away from the target, the excess space can be confiscated by the mediator. Then when another child needs extra space, it is first drawn from the mediator's available cache. Only when no excess exists will a child's update perturb the other components because it encroaches on their space.

If some cache exists and it is at least 80% of the need, then it is given and treated as if all the space necessary to avoid encroaching on the space allocated to the other children and hence minimizing the frequency that all children must be resynchronized.[49] When a resynchronization occurs, a fraction of the total space is allocated to the mediator to cache. Thus, the first time an entity increases, there is some cached space available, and thus we again are able to reduce resynchronization frequency.

It seems intuitive therefore to try to keep the hierarchies flat. This way, a larger pool of unused space may be accumulated and doled out such that the fewest number of synchronization messages are necessary. Unfortunately, when the available space is gone, then all children must be notified. Therefore, future work should consider evaluating depth versus breadth hierarchies.

5.9.7.4 Sparse Data

In filtering out unnecessary events, the greatest gains occur when the data of interest is sparse; that is, when the likelihood of satisfying the relationship is low. As an example of sparse data, the proximity of two people, calculated in two dimensions, bounded primarily by the UCSD campus of approximately three square miles is sparse

[49] 80% was arbitrarily selected in an attempt to minimize extra events.

when the objective is a proximity relationship of less than ten yards. We were able to achieve reduction from O(N) to an expectation of ln(N) events. By contrast, data constrained to freeway speeds, reported in integral units increases the probability of achieving inequality relationships and hence reduces the effectiveness of our solution. Consider, as an analogy, the log curve with the sparseness of the data increasing along the X-axis and the number of events generated along the Y-axis. As data becomes sparse, fewer events, proportionately, are generated. As data fills the available region (i.e., approaches one) the number of synchronization events generated with respect to the number of original sensor events is larger).

5.9.7.5 Controlling Event Generation

Tightly packed data naturally occurs as a relationship approaches satisfaction. For example, if the range ring strategy is employed, then as two entities approach each other, the range rings get smaller in size and events are generated more frequently. We would like to reduce the number of events generated under these conditions. We have developed three techniques to achieving such reductions.

5.9.7.5.1 Target Identification

Many of the relationships can be divided into sides (e.g., an upper and lower half) with a target (e.g., midpoint) to be approached from either side. To make this more effective, we found that the component could filter more efficiently if it knew whether its result would be used as part of another relationship, such that the only hard boundary was either an upper or lower bound. If it has only one hard boundary, then the OTFs monitored can be given more leeway when moving in the opposite direction. This is essentially changing from the omni-directional range-ring strategy to the parallel-line strategy.

5.9.7.5.2 Origin Offset / Increased Range

There are a variety of reasons that we cannot always use the parallel line strategy, such as when we want to aggregate multiple sensors. As an example, take an AF with N OTFs, where N/2 OTFs move away from their targets at the same speed as the other N/2 move toward their targets. The result is an essentially static AF.

The parallel line strategy will not generate any events as the N/2 entities moves away from their target, but each of the other N/2 moving toward their target will periodically result in N+1 messages (1 to the AF and N back to all the OTFs) for a total of ((N+1) * N / 2). This may be ok if the number of sensor reports is substantially greater. Using the range-ring strategy, generates events in both directions. Thus, as the N/2 move away from their targets, each generates 1 message to which a reply comes back from the mediator, and as the other N/2 move toward their targets, each generates 1 message to which a reply comes back from the mediator, for a total of 2N. Therefore, we choose to use the range ring strategy for children of an AF.

But, by using a range ring symmetric about the last reported position of a datum, we find ourselves in situations where, as we approach the target, the range ring shrinks such that almost any activity causes a new report. This is aggravated when poor sensors or environmental conditions result in fluctuating data. This is too costly.

We need a cross between the behavior of the parallel line and range ring strategies that will reduce the amount of events as (fluctuating) data backs away from the target yet will also cache available space to avoid synchronizing all children. This hybrid approach is shown in Figure 5-9, overlaid on the basic range ring approach. In short, we make the range ring 50% larger than usual. When we would previously have halved the range ring is now only ¾ the size. Any proper fraction will do. We want the same effective behavior though, so we keep the proximity the same, which causes these range rings to bulge on the back side. Thus, the center is offset from the actual data and the range is increased. The consequence is that we are allowed more freedom of movement to back away before generating a report, such that fluctuating data is given more wiggle room to avoid causing spurious events.

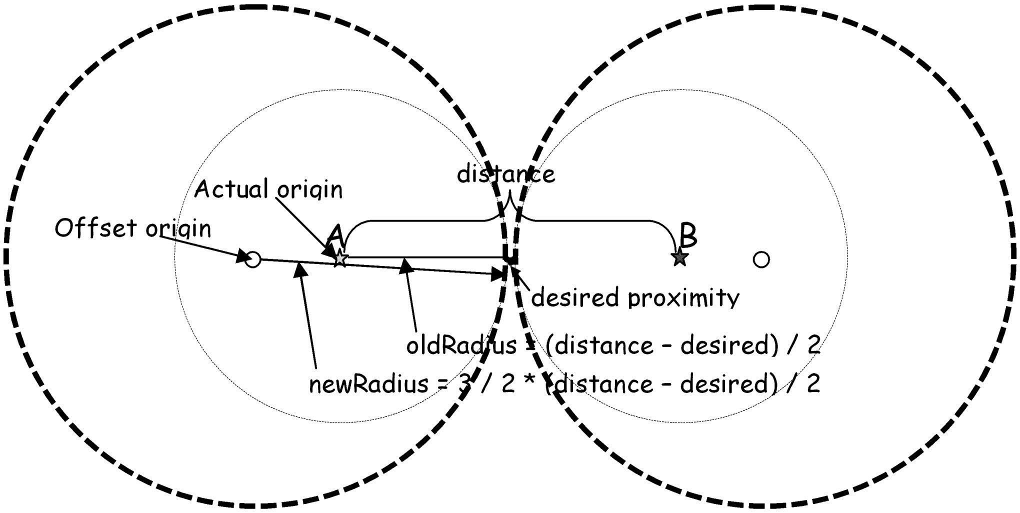

Actual origin
distance
Offset origin
A
B
desired proximity
oldRadius = (distance - desired) / 2
newRadius = 3 / 2 * (distance - desired) / 2

Figure 5-9 Offset Origin and Increase Range to Reduce Events

5.9.7.5.3 Quality of Service – Relationship Satisfaction Regions

Using a public infrastructure, the user and middleware do not control the sensors, the network organization, or the number of publishers, subscribers, advertisements, subscriptions and notifications; hence we cannot make guarantees about delivery. At this point we can merely assume timely and reliable delivery. What the user can control through the implementation strategy is the relationship evaluation and its selection of reporting accuracy, which may have implications in the number of events the system must process.

The reporting accuracy needs depend on the purpose of the relationship and the constraints imposed by the information availability (e.g., sensor precision and reporting rate). For example, buddy proximity for the purpose of an impromptu meeting need only inform us when a real possibility to meet occurs. If the buddies pass each other on the freeway, the sensor reporting rates may be such that the proximity is not detected. This is considered reasonable given that the desired meeting probably could not have occurred. Similarly, the proximity may be set to some distance D, but due to reporting rates or even implementation strategy, the proximity may not detected until the distance is something substantially less than D.

We therefore introduce a *relationship satisfaction region*, in which we allow the proximity goal to have both a hard and a soft goal. Then we can create a buffer, a relationship satisfaction region as shown in Figure 5-10.

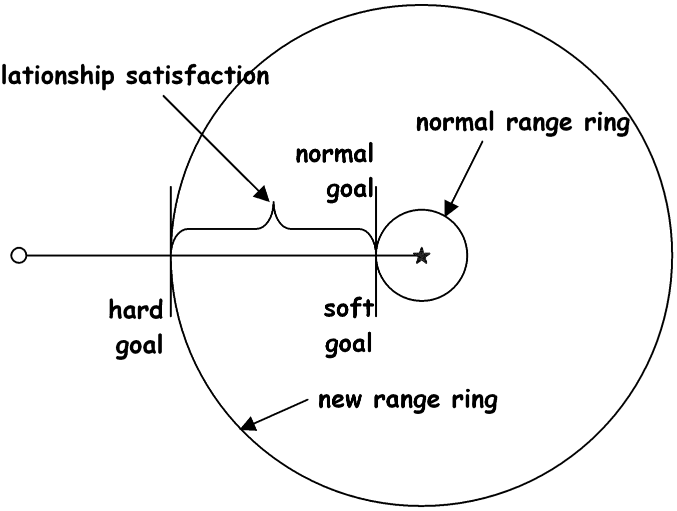


Figure 5-10 Relationship Satisfaction Region

This region allows us to provide a quality-of-service-like behavior where the relationship may be reported as early as reaching the soft goal, but will definitely be reported on reaching the hard goal. Thus, the implementation strategies create the range rings against the hard goal, but an update of any OTF checks for relationship satisfaction (e.g., proximity) against the soft goal. This allows us to generate fewer events as we approach satisfaction. Naturally, the size of the region used must be appropriate to the sensors and the task.

5.9.7.5.4 Implicit Knowledge

Although our original proximity algorithm was efficient, it was tightly bound; - each component was dependent on the known behavior of the other. Therefore, we choose to adopt the observer / mediator approach for the extensibility and scalability it provides. However, we do so at a cost of extra messages. The trait that we discover is that communications can be minimized if both entities (more if necessary) in a conversation run the same algorithm to determine follow-on behavior. In the proximity example, the only communication necessary is to report one user's new position and the algorithm on both sides can then computed the new range ring. This is facilitated by the common knowledge of the last reported positions. In the mediator case, we could give the OTFs the same algorithm as run by the mediator such that no reply is necessary for most updates. Although we still prefer the observer / mediator pattern, the point is that when knowledge and a need exist, we can use that knowledge to reduce event generation.

5.10 Summary

The "law of continuity", that to change from one state to another all intermediate states must be visited, serves as the foundation for the development of our implementation strategy components. First, we convert relationships to a distance along a continuous scale, then we define "half-way" points along relationship scale through which the data from individual sensors must pass in order to satisfy the relationship.

Thus we are able to set up bounding regions that enable independent filtering of each sensor. As the half-way points become smaller and smaller, the communicated data becomes more fine-grained.

We see by the complexity of the strategies developed that any distributed machine can be described that allows a substantial reduction in event traffic at the entry nodes. Combining this idea with the observer / mediator design pattern, we are able to develop a series of components from which we are can create hierarchical control structures as found in the traffic routing implementation strategy. We discovered that the more complex strategies required one-to-one conversations between the sensor proxies and the mediator, thus effectively reducing them to pair-wise proximity using a mediator. As a consequence, we discovered several reusable classes to make new relationship solutions easier.

6.0 Implementation of Fulcrum

Chapter 4 described the conceptual architecture of the Fulcrum approach to context-aware publish / subscribe and describes how it can be achieved as a straightforward generalization of CBPS. In part we extend messages to others than advertisements, subscriptions, and notifications; we separate content-addressability mechanisms from advertisements, subscriptions, and notifications to permit use for routing other payloads; and we modify event-brokers to accept remote code for local filtering.

Many important details, mostly regarding Fulcrum's implementation were not presented in Chapter 4. For those who might want to implement such a system, this chapter presents and discusses those details. Although Fulcrum is ultimately just an extension to CBPS, we present the entire system so that the parallels of the extensions, the patterns, are clear.

There are three main objectives in architecting and implementing Fulcrum. First, we want to leverage CBPS content-based routing and separation of concerns. Second, we want to extend those capabilities to increase the expressiveness and information availability. Third, we want to provide flexibility and to support the natural dimensions of growth. In particular, we want to allow the easy adoption of different communications protocols, support clients written in different languages, and easily develop new features.

6.1 Architectural Goals

Source code is available for SIENA [SIENA-2005]. It seems logical then to reuse it and make our modifications. However, the make versus buy/reuse decision found the implementation too tightly bound with usage assumptions and tradeoff beliefs that make it insufficiently extensible for the changes necessary for Fulcrum. Most notably, the I/O was tightly bound with the classes – leaving the serialization and transport embedded and difficult to extend. Additionally, the content-addressability features are tightly bound to the semantics of advertisements, subscriptions, and notifications. To achieve our goals, we instead leverage the basic patterns of CBPS systems, add our new expressiveness and information availability capabilities, and implement it all using a component-based approach and XML-based interfaces to facilitate future growth. The following sections detail the key design decisions.

6.1.1 Leveraging CBPS

The basic CBPS model is to provide an overlay network – a network of application-layer event-brokers, for the content-based routing of information. As with a traditional CBPS system, the efficient organization with respect to the physical network is presumed. Currently, we use a hub and spoke network model for the overlay network as shown in Figure 3-5.

6.1.2 Component-Based Development

Component-based development is used to maintain an appropriate separation of concerns and support evolution. Components in the system are independently created and then attached. For example, rather than providing a single controlling unit that is the main processor of information as well as the creator and owner of all related components, (e.g., sockets for communications) instead an assembler component, a factory, constructs the components, attaches them, and gets out of the way.

The component technique is used for several reasons. It facilitates development of individual modules to enhance the separation of concerns. This serves us when we wish to swap out one module for a better performing one or for one with different capabilities. This will show more clearly as we discuss the various system components. They also create a set of building blocks which makes different assemblies easy. This can be seen in the interfaces by which we create network connectivity or emulate a network on a single processor.

6.1.3 XML-Based Specification

Experience has shown that software developers prefer their own language, based either on task (e.g., SQL for database work) or comfort level (e.g., Visual Basic because it was the first language they learned). Consequently, although Fulcrum is built in Java, it is a middleware product and it needs to readily support user application development in any language. Therefore, a common interface language is necessary. And, as a research project, evolution is a given. Therefore, a flexible data representation is necessary. Naturally, these needs lead to our selection of XML as the common interface language.

To achieve maximum flexibility, we intentionally (at this point) avoid the brittleness created when a schema is declared.[50] General purpose processing is provided to ignore unexpected attributes and nodes. The system is intentionally not ‘bullet-proofed’ at this point. Additionally, the APIs currently developed have been selected to accept a DOM node that they may evolve without changes to the signature. This also facilitates multiple transports because they need not incorporate specific code modules, rather they must create the correct XML. And most languages have interfaces to convert

[50] A typical failing in using XML schemas is building them as a collection of type definitions instead of as a language. A full expression of this issue is beyond the scope of this dissertation.

a XML text stream into a document object model. A second set of APIs for direct manipulation without requiring a user to be XML literate have been developed and are described in parallel.

6.1.4 Transport / Protocol Mechanisms

All entities must be able to exchange information yet for various reasons the basic transport or protocol may not be identical. For example, the primary design supports sockets with a direct communication of XML data. However, HTTP-based, SOAP-based, or other communications styles may be necessary. Gigabit networks to low baud-rate serial lines may be employed. Therefore the transport / protocol mechanisms need to be isolated from the core processing. Then simple adaptors can be written for any given transport / protocol without affecting the core system. The adaptors become responsible for gathering a complete message before passing it along for processing.

6.1.5 Processing Isolation

We need to be able to isolate primary event processing from communications and from deployed code. Therefore we choose to let each processing component run in its own separate thread. This includes I/O components, event-brokers, and deployed user code. We then use synchronous queues to glue the various system components together.

6.2 Implementation Details

Fulcrum follows the pattern of other CBPS systems by providing an overlay network, a network of nodes at the application layer that lay over the physical network, for the content-based routing of information. As with traditional CBPS systems, the efficient organization with respect to the physical network is presumed. Currently, we use a hub and spoke network model for the overlay network as shown in Figure 3-5. On top of this we add the expressiveness and information availability capabilities called out in Section 4.

6.3 Overview

Before dropping into the implementation details, a quick scope of the effort may be provided by Table 6-1.

Table 6-1 Source Lines of Code Count

Purpose	Files	SLOC
Fulcrum core	147	10246
Agents	16	3073
Sensors	3	3002
Test	27	2613

Remember, this is just a CBPS system with extensions to support Internet-scale context-awareness. At the heart of the system is the event-broker. Because we choose to follow component-based software development principles, the I/O capabilities are separated from core processing.

A basic connection of two event-brokers is represented in Figure 6-1. Each event-broker is clustered with an input queue (IN) an output queue (OUT), and an input/output component (I/O) to which is attached a metadata adaptor – a modifier (MOD). Not shown is the database component provided for each event-broker. Each of these components, and the value they bring, will be discussed in detail below.

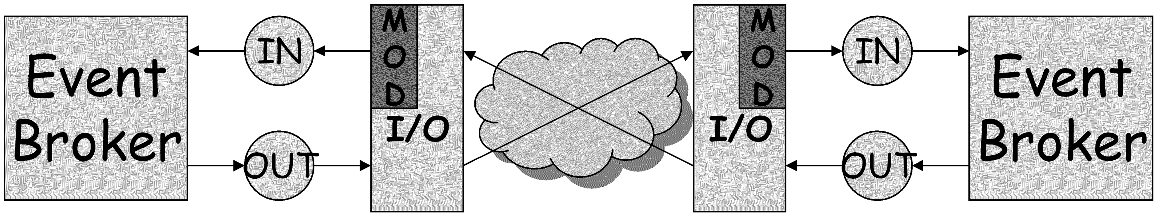


Figure 6-1 Event-Broker to Event-Broker Connection

A basic connection of a pub / sub client and an event-broker is represented in Figure 6-2. The pub / sub client looks very similar to the left-hand event-broker in the prior figure. This similarity will play a part in our design decisions as detailed below. We also and use the general term *processing element* to refer to either an event-broker or pub / sub client as appropriate. Noteworthy is that the pub / sub client may be either a publisher only, a subscriber only, or both publisher and subscriber. Each of these components, and the value they bring, will be discussed in detail below.

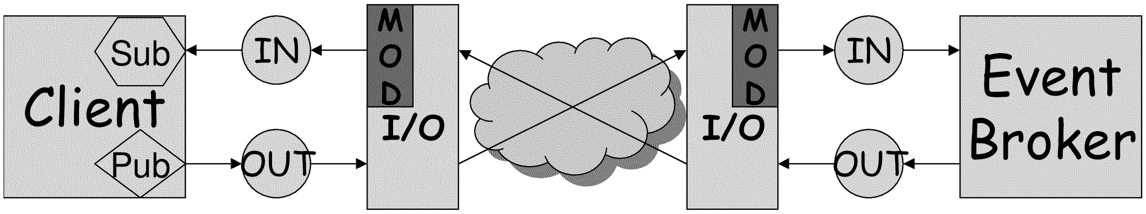


Figure 6-2 Pub / Sub Client to Event-Broker Connection

Figure 6-3 shows that each event-broker is actually connected to multiple other processing elements. Adding to our list of processing elements is an implementation strategy component, "wrapped" to isolate the event-broker from potential bad behavior of the deployed user code. The previously referenced input queue (IN) is actually a shared FIFO queue and serves as a multiplexer. The previously referenced output queue (OUT) is actually a collection of output queues, one for each connection, and a de-multiplexer to manage them.

The following two sections complete our high level overview with a description of the core system components. The remaining sections provide explicit detail.

6.3.1 Processing Elements

The primary objects of the system are the standard CBPS processing elements, identified as event-brokers, publishers, and subscribers.

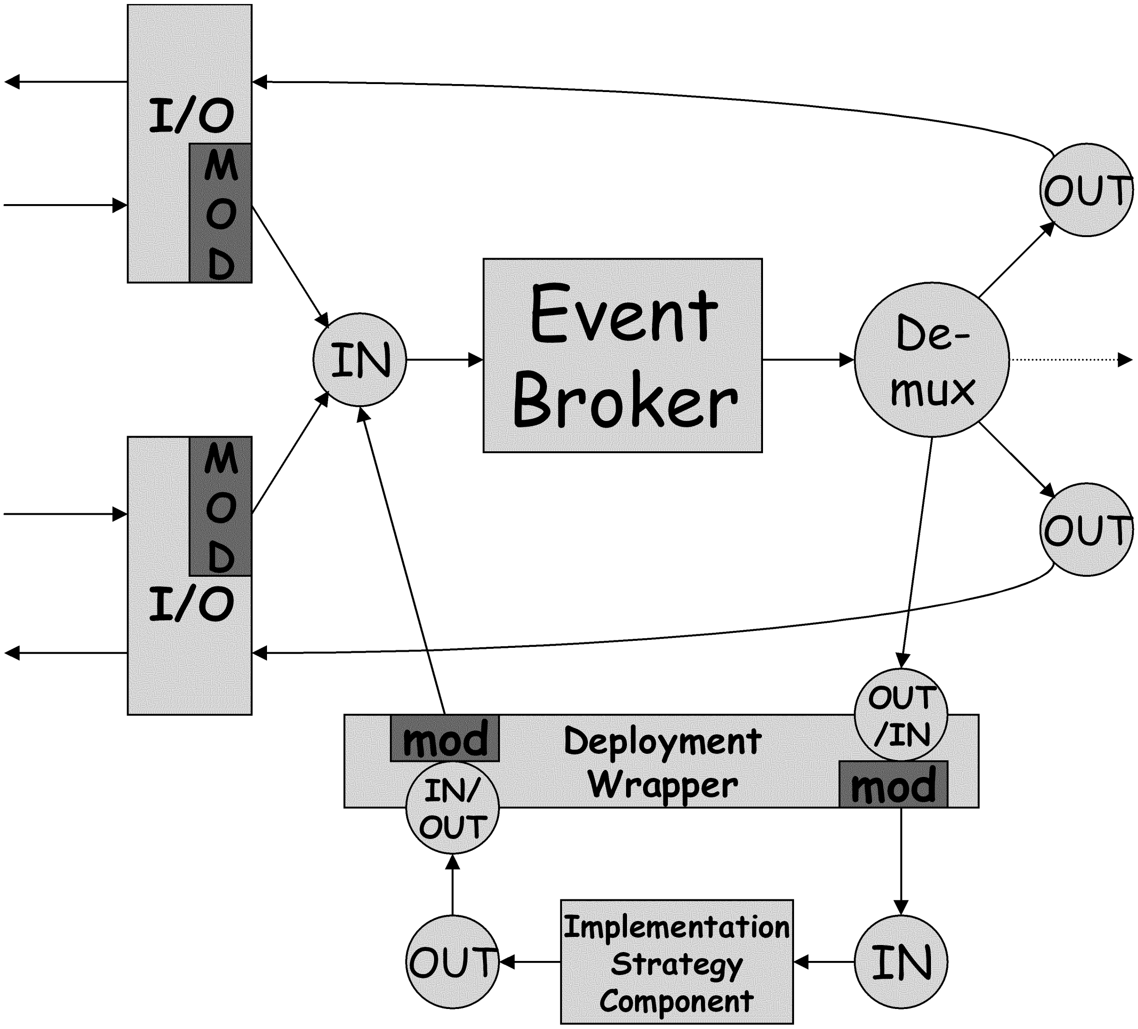


Figure 6-3 Event-Broker Input-Multiplexer, Output De-Multiplexer

Event-brokers lie at the heart of the system to perform the content-based routing. The event-brokers manage the system semantics. Event-brokers may connect to other event-brokers in order to form the overlay network.

Publishers advertise the base event-types they will publish and then they publish notifications of that base event-type. Their notifications may contain more fields than advertised.

Subscribers subscribe to event-types of interest and accept notifications that are passed on by the event-brokers. The notifications are expected to satisfy the event-types of prior subscriptions but may contain more fields than the base event-type.

Although we talk about three “types” of processing elements, they are really roles. That is, an event-broker is really both a publisher and a subscriber with some additional semantics thrown in. A pub / sub client may be either a publisher or subscriber or both. Therefore, these are implemented with a common message passing interface such that an object may take on any role. An architectural rule is imposed that an (initial) handshake is used to declare what data types a processing element is willing to receive and the agreement that it will not be sent other than what it has agreed to accept.

6.3.2 Secondary Objects

The secondary objects, with the exception of the forwarding database are specific to the Fulcrum implementation. They are the deployment wrappers, implementation strategy components, forwarding database, and queues as well as I/O and Threading components and messages.

Deployment wrappers are used to isolate implementation strategy instances from the remainder of the core system processing. This is the component responsible for securely instantiating deployed user code and ensuring appropriately constrained behavior.

Implementation strategy components are first-class pub / sub clients, agents, and may participate in any or all of the three primary roles of event-broker, publisher, or subscriber. It would be unusual for an implementation strategy to act as an event-broker; but, as an example, the ability receive advertisements, which are normally limited to event-brokers, might be valuable.

Forwarding databases are attached to each event-broker. The forwarding database holds the persistent data tables for advertisements, subscriptions, routing slips, and active subscription requests as well as all the matching algorithms.

Queues are glue that hold everything together. The queues serve as a separation of concerns technique to isolate the logical processing of information from the physical transport and protocol. Adaptors to convert data must push or pull their information to or from the queues. These work in conjunction with the I/O and threading capabilities. As shown in figures Figure 6-1 through Figure 6-3, queues sit between I/O components and the processing elements.

I/O processing handles the network communications. The baseline system provides for socket communications with straight XML. Jabber, HTTP, SOAP, or other communications transports or protocols could be easily added by plugging in different interfaces.

Threading is provided to enable processing to occur where possible and meaningful. They allow an independence of I/O, processing elements, and implementation strategy component behaviors.

Messages are XML-based and provide the common communications basis between system components. Both intra-system (internal) and inter-system (external) messages exist, where a “system” is any of our core processing elements or an implementation strategy component.

The nature of the component capabilities and significance makes it most useful to detail each of these in a bottom-up fashion as follows. First, the messages, queues, and threading are discussed.

6.4 Messages – Network and Internal

The messages are comprised of three groups, one is internal and two are network based. The internal messages (e.g., connection service) provide communications between elements where an ownership or even 'knows-about' relationships may not exist even though all elements are contained within the same executable. The external messages, (e.g., a configuration request (ISCAConfig) or a message bundle (ISCAMsg)) pass between the primary processing elements and nominally pass between different executables.[51]

6.4.1 Internal Messages

Internal messages are required because the system design does not directly attach the event-broker (or other processing elements) directly to the physical I/O. And, no additional APIs have been provided that would tightly bind the event-broker to any physical I/O. Thus, no physical I/O is actually necessary. However, when logical I/O, through the queues has been affected, the system must communicate the connection creation or disconnection.[52] A sample connection creation is:

```
<ConnectionService>
      <ConnectionId value=”xx”/>
      <Action value=”newConnection” direction=”toParent”/>
</ConnectionService>
```

A sample disconnection is signaled by:

```
<ConnectionService>
      <ConnectionId value=”xx”/>
      <Action value=”disconnection” reason=”run-exception”/>
</ConnectionService>
```

[51] Note: the entire system can be contained within a single executable.

[52] This is one downside to a component-based design.

As these messages are generated internally, there are no associated Java APIs. Because the processing elements, especially sensors, can be expected to work merely in terms of their raw data or demands, the adaptor may need to attach meta-information about the connection on which the data was received. In this case, we bind together the metadata with the message using a containing wrapper:

```
<Wrap> xxx yyy </Wrap>
```

to group multiple messages by wrapping them. Thus, a ConnectionService message may be attached to a regular ISCAMsg. This is done internally in the both ISCASocketReader and the DeploymentWrapper, so again there are no associated Java APIs.

6.4.2 External Messages

The external messages are comprised of two groups of messages, configuration and standard content. The configuration messages handle initialization information while the content messages provide routing slips, metadata, metadata filters, advertisements, subscriptions, notifications, and active subscriptions and payloads.

6.4.2.1 ISCAConfig

These messages coordinate and give initialization handshakes. In particular, because all connections 'look the same' in form, these messages are used to differentiate capabilities of the various processing roles. A system declares what it is capable of or desirable to receive (e.g., receive advertisements, subscriptions, and notifications).

```
<ISCAConfig>
   <Registration request="false" reply="false">
      <MessageContext/>
      <MessageContextFilter/>
      <RoutingSlip/>
      <Advertisement/>
      <Subscription/>
      <Notification/>
      <ActiveSubscription/>
      <Payload/>
   </Registration>
</ISCAConfig>
```

An event-broker would specify all. A publisher might specify none. A subscriber may specify that it will receive notifications only. But, a subscriber may wish to know the context associated with the notification, such as how long it has been in the system, say because no guarantee of time synchronization exits or because a timestamp is not provided with the data itself. In such a case, the subscriber can also specify that it will accept the message context. This is all part of increasing the information availability.

Upon receipt by an event-broker, several steps occur. 1) The connection is annotated with its capabilities. 2) If the request attribute is set true, then a response will be generated, with request set to false and reply set to true, describing what the event-broker is capable of. And, if the connection is capable of receiving advertisements, all of the advertisements held by the event-broker will be pushed down the connection.[53]

The attributes *request* and *reply* are not required and if omitted are treated as false values. Anything unrecognized as true is treated as false.

In order to bootstrap the system, if an event-broker provides a connection, it is assumed to be able to receive everything. However, it could, as a result of the initial handshake, reduce the list of information components it is willing to accept.

6.4.2.2 ISCAMsg

An ISCAMsg is the message bundle container described in Section 4 and is responsible for conveying content and controls throughout the system. It is essentially the wrapper that provides for the composition of the real content of advertisements,

[53] Mechanisms to control a mass of advertisements or flood-forwarded subscriptions is left to future work.

subscriptions, and notifications as well as routing slips, activeSubscriptions, message context, message context filters, and data payloads. An ISCAMsg is composed of some quantity of six different information components as follows:

```
<ISCAMsg>
   [ MessageContext ]
   [ MessageContextFilter (type=incoming or type=outgoing) ]*
   [ RoutingSlip (type=subscription or type=notification)]*
   [ Advertisement | Subscription | Notification ]*
   [ ActiveSubscription ]*
   [ Payload ]*
</ISCAMsg>
```

An <ISCAMsg> is an entity that travels between event-brokers or pub/sub clients as a whole. Its parts are used within the event-broker and may even be altered as detailed below, but the entire <ISCAMsg> container travels as a group. Although all components are identified as optional, normally one of the advertisement-subscription-notification group exists. They become optional when an active subscription is used to cause capability to be deployed to an event-entry edge of the network as will be discussed in more detail in the active subscription section. An empty message or one with no ability to be routed would be discarded.

Network Interface Samples (abbreviated format):

```
<ISCAMsg><Advertisement> … </Advertisement></ISCAMsg>
<ISCAMsg>
   <Subscription> … </Subscription>
   <ActiveSubscription> … </ActiveSubscription>
</ISCAMsg>
```

Application Programmer Interface Samples (abbreviated format):

```
ISCAMsg msg1 = new ISCAMsg() ;
msg1.add( new ISCAAdvertisement() ) ;

ISCAMsg msg2 = new ISCAMsg() ;
msg2.add( new ISCASubscription() ) ;
msg2.add( new ISCAActiveSubscription() ) ;
```

The base CBPS components and their supporting attribute-value and attribute-constraints are well documented in the CBPS literature. Here we will provide only brief descriptions that we may focus on the innovations. As the information components are merely conduits of information, the actual usage semantics will be discussed in the event-broker processing section. The description is purposely “out of order” to present the best known components first to show that the additional components and capabilities are simple extensions.

6.4.2.2.1 Attribute-Values

Each attribute-value consists of a name, type, and value. The current implementation follows the CBPS lead limiting the types to primitives only. Fulcrum currently supports double, long, and string. Inherently, the attribute-value is an equality statement (i.e., name = value). Strings are currently limited to equality, but as discussed in the database forwarding section, efficient techniques to support operators such as prefix and suffix are known. An attribute-value is identified by the <av> tag.

Network Interface Sample:

```
<av name=”name” type=”string” value=”Bob”/>
<av name=”X” type=”long” value=”123”/>
<av name=”Y” type=”long” value=”123”/>
```

Application Programmer Interface Samples:

```
ISCAGeneralPurposeAttributeConstraint nameAV, xAV, yAV ;

nameAV = new ISCAGeneralPurposeAttributeValue(“name”,“string”,“Bob”));
xAV = new ISCAGeneralPurposeAttributeValue( “X”, “long”, “123” ) ) ;
yAV = new ISCAGeneralPurposeAttributeValue( “Y”, “long”, “123” ) ) ;
```

The Java API is provided solely to hide the underlying XML.

6.4.2.2.2 Attribute-Constraints

Each attribute-constraint consists of a name, type, operator, and value. The operator list includes <, <=, =, >=, >, and ANY, where ANY serves as a wildcard to indicate that the named field must exist, but that ANY value is acceptable. An attribute-constraint is identified by the <ac> tag.

Network Interface Sample:

```
<ac name=”name” type=”string” op=”EQ” value=”Bob”/>
<ac name=”X” type=”long” op=”ANY”/>
<ac name=”Y” type=”long” op=”ANY”/>
```

Application Programmer Interface Samples:

```
ISCAGeneralPurposeAttributeConstraint nameAC, xAC, yAC ;

nameAC = new ISCAGeneralPurposeAttributeConstraint(
   “name”, “string”, “EQ”, “Bob” ) ) ;
xAC = new ISCAGeneralPurposeAttributeConstraint(
   “X”, “long”, “ANY”, null ) ) ;
yAC = new ISCAGeneralPurposeAttributeConstraint(
   “Y”, “long”, “ANY”, null ) ) ;
```

6.4.2.2.3 Advertisement Component

The advertisement component announces event-types to be published by defining the bounds in which publications will be provided as an attribute-constraint list. As noted previously, this is only a description of the required fields provided by notifications. This does not restrict a notification from providing other optional fields as well. This is specified using the <Advertisement> tag.

Network Interface Sample:

```
<Advertisement floodForward=”true”>
   <ac name=”name” type=”string” op=”EQ” value=”Bob”/>
   <ac name=”X” type=”long” op=”ANY”/>
   <ac name=”Y” type=”long” op=”ANY”/>
</Advertisement>
```

Application Programmer Interface Samples:

```
ISCAAdvertisement advertisement = new ISCAAdvertisement() ;
// advertisements default to floodForwarding - no need to set.
advertisement.add( new ISCAGeneralPurposeAttributeConstraint(
   “name”, “string”, “EQ”, “Bob” ) ) ;
advertisement.add( new ISCAGeneralPurposeAttributeConstraint(
   “X”, “long”, “ANY”, null ) ) ;
advertisement.add( new ISCAGeneralPurposeAttributeConstraint(
   “Y”, “long”, “ANY”, null ) ) ;
```

CBPS systems have previously been set to either flood forward advertisements or to flood forward subscriptions. Fulcrum defaults to flood forwarding advertisements. But, under our requirements for more direct control such that we can leverage the user's contextual knowledge, we provide the ability to turn it off. In such cases, to propagate the advertisement a routing slip or other carrier must be used as detailed below.

6.4.2.2.4 Subscription Component

The subscription component announces event-types of interest by defining the bounds in which notifications must be provided as an attribute-constraint list. As noted previously, this is only a description of the required fields provided in a notification. Other fields may be provided, but will be ignored. This is specified using the <Subscription> tag.

Network Interface Sample:

```
<Subscription floodForward=”true”>
   <ac name=”name” type=”string” op=”EQ” value=”Bob”/>
   <ac name=”X” type=”long” op=”ANY”/>
   <ac name=”Y” type=”long” op=”ANY”/>
</Subscription>
```

Application Programmer Interface Samples:

```
ISCASubscription subscription = new ISCASubscription() ;
subscription.setFloodForward( true ) ;
subscription.add( new ISCAGeneralPurposeAttributeConstraint(
   “name”, “string”, “EQ”, “Bob” ) ) ;
subscription.add( new ISCAGeneralPurposeAttributeConstraint(
   “X”, “long”, “ANY”, null ) ) ;
subscription.add( new ISCAGeneralPurposeAttributeConstraint(
   “Y”, “long”, “ANY”, null ) ) ;
```

As will be discussed in the advertisement and subscription sections, the subscription semantics are to trace back toward originating publishers providing advertisements that match the subscription. However, there are a few cases where flood forwarding the subscription may make sense and the user should be allowed to make that decision. In those cases, the Subscription tag can be extended with the control flag.

6.4.2.2.5 Notification Component

The notification component produces content, a collection of attribute-values, using the <Notification> tag.

Network Interface Sample:

```
<Notification floodForward=”true”>
   <av name=”name” type=”string” value=”Bob”/>
   <av name=”X” type=”long” value=”10”/>
   <ac name=”Y” type=”long” value=”20”/>
</Notification>
```

Application Programmer Interface Sample:

```
ISCANotification note = new ISCANotification() ;
note.add(
   new ISCAGeneralPurposeAttributeValue( “name”, “string”, “test” ) ) ;
note.add( new ISCAGeneralPurposeAttributeValue( “X”, “long”, “10” ) );
note.add( new ISCAGeneralPurposeAttributeValue(“Y”, “long”, “20” ) ) ;
```

To be consistent with advertisements and subscriptions, the flood forwarding flag is also allowed for notifications. This provides some added flexibility and would be useful as part of an emergency broadcast system, say a tsunami warning, for which a subscriber may not have even been aware to have subscribed. However, it does impose a burden on subscribers to validate that the data they receive is what they expect and to not crash on receiving such exception data.

6.4.2.2.6 Routing Slip Component

As described in Section 4, a routing slip is to be treated either as a subscription or a notification. Thus, it may be either an attribute-constraint list or an attribute-value list. The content is checked and data inconsistent with the declared type is discarded.

Network Interface Samples:

```
<RoutingSlip type=”subscription” persistent=”false”>
   <ac name=”name” type=”string” op=”EQ” value=”Bob”/>
   <ac name=”X” type=”long” op=”ANY”/>
   <ac name=”Y” type=”long” op=”ANY”/>
</Subscription>
```

```
<RoutingSlip type="notification" persistent="true">
   <av name="name" type="string" value="Bob"/>
   <av name="X" type="long" value="10"/>
   <ac name="Y" type="long" value="20"/>
</Notification>
```

Application Programmer Interface Samples:

```
ISCARoutingSlip routingSlip1 = new ISCARoutingSlip( "subscription" ) ;
routingSlip1.setPersistent( false ) ;
routingSlip1.add( new ISCAGeneralPurposeAttributeConstraint(
   "name", "string", "EQ", "Bob" ) ) ;
routingSlip1.add( new ISCAGeneralPurposeAttributeConstraint(
   "X", "long", "ANY", null ) ) ;
routingSlip1.add( new ISCAGeneralPurposeAttributeConstraint(
   "Y", "long", "ANY", null ) ) ;

ISCARoutingSlip routingSlip2 = new ISCARoutingSlip( "notification" ) ;
// routing slips default to persistent - no need to set.
routingSlip2.add(
   new ISCAGeneralPurposeAttributeValue( "name", "string", "test" ) ) ;
routingSlip2.add(
   new ISCAGeneralPurposeAttributeValue( "X", "long", "10" ) );
routingSlip2.add(
   new ISCAGeneralPurposeAttributeValue("Y", "long", "20" ) ) ;
```

The key reason for the persistence flag is to allow the routing slip to disappear once it has performed its duty. For example, the setup for a one-to-one connection might set the flag to false. Because advertisements, subscriptions, and notifications already have a flood forwarding flag, add one here would be meaningless.

6.4.2.2.7 Message Context Component

Context-aware computing is about dealing with information in a way that is relevant to its usage. In our case, there are a variety of situations where the information needs to be shared by or with the system in order to achieve specific aims, in particular, collaboration among the system components. But first we must increase information availability. Logically, the message context is a notification about the message and so uses the same attribute-value list as a notification. The message context component provides system information about the data using the <MessageContext> tag.

Network Interface Sample:

```
<MessageContext>
   <av name=”origin” type=”string” value=”abc”/>
   <av name=”sender” type=”string” value=”abc”/>
   <av name=”hopCnt” type=”long” value=”1”/>
   <av name=”arrivalTime” type=”long” value=”123”/>
   <av name=”travelTime” type=”long” value=”234”/>
</MessageContext>
```

Application Programmer Interface Sample:

```
ISCAMessageContext messageContext = new ISCAMessageContext() ;
messageContext.add(
   new ISCAGeneralPurposeAttributeValue( “origin”, “string”, “abc” ) );
messageContext.add(
   new ISCAGeneralPurposeAttributeValue( “sender”, “string”, “abc” ) );
messageContext.add(
   new ISCAGeneralPurposeAttributeValue(“hopCnt”, “long”, “1” ) ) ;
messageContext.add(
   new ISCAGeneralPurposeAttributeValue(“arrivalTime”, “long”, “123”));
messageContext.add(
   new ISCAGeneralPurposeAttributeValue(“travelTime”, “long”, “234” ));
```

Few users should ever directly construct a message context.

6.4.2.2.8 Message Context Filter Component

The message context filter component provides filtering control such that additional, non-data limitations can be imposed. For example, suppose an information provider wants to advertise a lunch-time special but doesn’t send the message until 11:30am. Then, he could limit the exposure of his *outgoing* information to only those people who are within three hops. Unfortunately, meaningful locality is not guaranteed when internet connections are involved. A consumer may similarly limit *incoming* information. Both forms use the <MessageContextFilter> tag.

Network Interface Sample:

```
<MessageContextFilter type=”outgoing”>
   <ac name=”hopCnt” type=”long” op=”LE” value=”3”/>
   <ac name=”travelTime” type=”long” op=”LE” value=”1000”/>
</MessageContextFilter>
```

```
<MessageContextFilter type="incoming">
   <ac name="hopCnt" type="long" op="LE" value="3"/>
   <ac name="travelTime" type="long" op="LE" value="1000"/>
</MessageContextFilter>
```

Application Programmer Interface Sample:

```
ISCAMessageContextFilter messageContextFilterOut ;
messageContextFilterOut = new ISCAMetaLocalFilter( "outgoing" ) ;
messageContextFilterOut.add( new ISCAGeneralPurposeAttributeConstraint(
   "hopCnt", "long", "LE", "3" ) ) ;
messageContextFilterOut.add( new ISCAGeneralPurposeAttributeConstraint(
   "travelTime", "long", "LE", "1000" ) ) ;

ISCAMessageContextFilter messageContextFilterIn ;
messageContextFilterIn = new ISCAMetaLocalFilter( "incoming" ) ;
messageContextFilterIn.add( new ISCAGeneralPurposeAttributeConstraint(
   "hopCnt", "long", "LE", "3" ) ) ;
messageContextFilterIn.add( new ISCAGeneralPurposeAttributeConstraint(
   "travelTime", "long", "LE", "1000" ) ) ;
```

6.4.2.2.9 Active Subscription Component

An active subscription is appended to a message with a subscription as shown below.[54] It is composed of several parts as follows:

1. Some implementation strategies are provided. Each composed of:

   a. deployment slip
   b. implementation class
   c. initialization parameters

2. the implementation code is provided.

Network Interface Sample:

```
<ISCAMsg>
  <Subscription>
    <ac name="name" type="string" op="EQ" value="BobBillProximity"/>
    <ac name="distance" type="double" op="LE" value="10.0"/>
  </Subscription>
  <ActiveSubscription>
     <ImplementationStrategy>
        <DeploymentSlip type="subscription">
          <ac name="name" type="string" op="EQ" value="Bob"/>
          <ac name="X" type="long" op="ANY"/>
          <ac name="Y" type="long" op="ANY"/>
        </DeploymentSlip>
        <ImplementationClass name="ISCA.Agent.ObserverAgent"/>
```

[54] Actually, the subscription itself is optional, allowing an active subscription to exist in its own right.

```
        <InitializationParameters>
          <uid value=”Bob”>
          <ctrluid value=”BobBillProximity”>
          <more class specific params/>
        </InitializationParameters>
     </ImplementationStrategy>
     <ImplementationCode> code here </ImplementationCode>
  </ActiveSubscription>
</ISCAMsg>
```

Application Programmer Interface Sample:

```
ISCAMsg msg = new ISCAMsg() ;
ISCASubscription sub = new ISCASubscription() ;
ISCAActiveSubscription actSub = new ISCAActiveSubscription() ;

msg.add( sub ) ;
msg.add( actSub ) ;

sub.add( new ISCAGeneralPurposeAttributeConstraint(
   “name”, “string”, “EQ”, “BobBillProximity” ) ) ;
sub.add( new ISCAGeneralPurposeAttributeConstraint(
   “distance”, “double”, “LE”, “10.0” ) ) ;

ISCAImplementationStrategy strategy = new ISCAImplementationStrategy();
actSub.add( strategy ) ;

ISCADeploymentSlip deploy = new ISCADeploymentSlip( “subscription” ) ;
deploy.add( new ISCAGeneralPurposeAttributeConstraint(
   “name”, “string”, “EQ”, “Bob” ) ) ;
deploy.add( new ISCAGeneralPurposeAttributeConstraint(
   “X”, “long”, “ANY”, null ) ) ;
deploy.add( new ISCAGeneralPurposeAttributeConstraint(
   “Y”, “long”, “ANY”, null ) ) ;
strategy.add(deploy ) ;

ISCAImpStrategyInstanceClass impStrategyInstanceClass =
   new ISCAImpStrategyInstanceClass( “ISCA.Agent.ObserverAgent” );
strategy.add( impStrategyInstanceClass ) ;

ISCAImpStrategyInstanceParams initParams =
   new ISCAImpStrategyInstanceParams() ;
initParams.add(“<uid value=\”Bob\”>” ) ;
initParams.add(“<ctrluid value=\”BobBillProximity\”>” ) ;
strategy.add( initParams ) ;

ISCAImplementationStrategyCode impCode =
   new ISCAImplementationStrategyCode(
      “ISCA.Agent.ObserverAgent”, “.:/home/rboyer”, true ) ;
actSub.add( impCode ) ;
```

The example shows only a single implementation strategy. A real case would also include a subscription to Bill's location with an observer agent and in the composite approach, a mediator / observer, call it BobBillProximity, which would be given an deployment slip indicating it should be co-located with either the Bob or Bill observer.

6.4.2.2.10 Payload Component

The payload component provides content of interest using the <Payload> tag. Its contents are completely freeform within the confines of legal XML. They are expected to be meaningful to the subscriber.

Network Interface Sample:

```
<Payload> meaningful stuff </Payload>
```

Application Programmer Interface Sample:

```
ISCAPayload payload = new ISCAPayload( meaningfulStuffObject ) ;
```

Naturally, the meaningful stuff object must satisfy the payload signature which only accepts an XMLCodecObject type (i.e., that supports toXML() and fromXML()). If a programmer wants the fromXML() to be invoked on his behalf, he must register a factory that is able to examine the XML and produce an appropriate object.

6.5 Queues

There are two primary types of queues that support the system.

A *FifoQueue* serves as the primary input to each processing element. This is a synchronized element that allows different threads to append and remove objects from the queue without race conditions of partial writes or reads. It frequently serves as a multiplexer, accepting input from multiple I/O component connections and with an event-broker pulling the data for processing.

A *MultiSendQueue* servers as a de-multiplexer used by event-brokers to communicate with their many related nodes while remaining ignorant of actual connection mechanisms. The output queue for each connection is added to the multi-queue by a logical name.

The result of this design is that a system may be run independent of transportation. That is, a small network could be run on a single machine, where the input queues to processing elements are directly attached to the MultiQueue of multiple other nodes. Similarly, some or all of the primary processing elements could be run on separate machines. With the right I/O service factories in place, we could also use different protocols or transportation mechanisms.

6.6 Threads

In order to maintain independence of processing elements, the system uses threads to allow the processing element to run concurrent with any I/O as well as concurrent with another event-broker where a system is contained on a single processor.

In order to provide consistent start and stop capabilities, an IThread interface is specified. This allows us to restore the deprecated `stop()` function of threads in a safe way.

```
public interface IThread
{
  public void ithread_start() ;
  public void ithread_stop() ;
}
```

Every run method provides a fairly trivial loop as shown below. The loop is controlled by an internal flag (i.e., terminateFlag) and an external flag (i.e., okToRun) in order to provide safe `stop()` functionality. The external flag is controlled by the ithread_stop() method.

```
public void run()
{
    boolean terminateFlag = false ;
    while( okToRun && !terminateFlag )
    {
      // Note: the remove is semi-blocking which allows us to return
      //       periodically to check to quit or to do other things.
      Object obj = inputQueue.remove( timeoutMs, timeoutNs ) ;
      if  ( null != obj )
         terminateFlag = process( obj ) ;
    }
}
```

6.7 I/O Elements

The I/O elements could be any of sockets, Jabber, HTTP, etc. In order to manage the physical transportation of information independently from the primary system, the queues and internal messages are used. The I/O components handle physical transports and protocols while the queues provide a staging area for the data prior to being processed by the main system. The I/O components can then discuss status, such as connections or disconnections using internal messages, passed through the queues. There are four classes in the ISCA.IO.ISCAService package. These are as follows.

6.7.1 ISCAIOServerService

The server service accepts the assignment of a port, an input queue, and a multi-output queue. It then creates a ServerSocket on the localhost using the given port. On failure, it returns false. On success, it returns true. The server service is a thread and must be started to activate its other capabilities. The run() method then sits in an infinite loop accepting connections from clients.

Upon connection, a new output fifo-queue is created. It is inserted into the multi-queue using the hashcode() of the client socket. It uses the socket and the output queue as the arguments to the creation of an ISCASocketWriter.

It next places into the input queue a <ConnectionService> message indicating an action of newConnection. Then it uses the socket and the input queue as the arguments to the creation of an ISCASocketReader. The main body is shown below.

```
while( !done )
try
{
   Socket clientSocket = null;
   try
   {
     clientSocket = serverSocket.accept() ;
   }
   catch (IOException e)
   {
     System.out.println("Accept failed: " + port ) ;
     continue ;
   }

   IQueueSynch clientQueue = new FifoSynch( name + ".SS" ) ;
   serverOutputQueue.setSendQueue(
    "" + clientSocket.hashCode(), clientQueue ) ;

   ISCASocketWriter sktWriter =
    new ISCASocketWriter( clientSocket, clientQueue ) ;
   sktWriter.start() ;

   serverInputQueue.append(
      "<ConnectionService>"
    + "<ConnectionId value=\"" + clientSocket.hashCode() + "\"/>"
    + "<Action value=\"newConnection\"/>"
    + "</ConnectionService>"
   ) ;

   ISCASocketReader sktReader =
      new ISCASocketReader( clientSocket, serverInputQueue ) ;
   sktReader.wrapData( true ) ;
   sktReader.start() ;

   yield() ;
}
catch( Exception e ) { e.printStackTrace() ; }
```

6.7.2 ISCAIOClientService

The client service accepts the assignment of host, port, and input and output queues. It then creates a socket and attempts to attach to the specified host and port. On failure, it returns false. On success, it uses the socket and the input queue as the arguments to the creation of an ISCASocketReader. It then uses the socket and the output queue as the arguments to the creation of an ISCASocketWriter. The work of the client service is now done. The main body is shown below:

```
Socket clientSocket = new Socket( host, port ) ;
clientSocketHashCode = "" + clientSocket.hashCode() ;

sktReader = new ISCASocketReader( clientSocket, clientInputQueue ) ;
sktReader.wrapData( true ) ;
sktReader.start() ;

sktWriter = new ISCASocketWriter( clientSocket, clientOutputQueue ) ;
sktWriter.start() ;
```

6.7.3 ISCASocketReader

The socket reader is provided a socket from which to read and a queue into which to place the results. Another flag, `wrapData`, may be set through a member function to have the reader wrap the actual data with connection management information.

When the `run()` method is executed, the socket's input stream is read and pushed through an XML pull parser for the sole purpose of clustering an XML sequence.[55] Each time an entire sequence is achieved, it is placed into the queue. When the `wrapData` flag is set, then the connection id is attached to the raw data, so that the processing unit on the other side of the queue, which may be receiving data from multiple sources, can uniquely identify the source of the data. The `run()` method is shown below.

[55] A more efficient bracket matching can later be developed to replace this code

```
public void run()
{
   BufferedReader in = null ;
   try
   {
      // Presumably this will be chunky and use blocking reads...
      in = new BufferedReader(
         new InputStreamReader( clientSocket.getInputStream() ) ) ;

      XmlPullParser parser = new KXmlParser() ;
      parser.setInput( in ) ;

      boolean done = false ;
      while( !done )
      {
         String result = traverse( parser, "" ) ;

         if  ( wrapFlag )
         {
           result = "<Wrap><ConnectionService><ConnectionId value=\""
            + clientSocket.hashCode() + "\"/></ConnectionService>"
            + result
            + "</Wrap>" ;
         }

         queue.append( result ) ;
      }
   }
   catch( Exception e )
   {
      queue.append(
        "<ConnectionService>"
         + "<ConnectionId value=\"" + clientSocket.hashCode() + "\"/>"
         + "<Action value=\"run-exception\"/>"
         + "</ConnectionService>"
      ) ;
   }

   if  ( null != in )
      in.close();
   clientSocket.close();
}
```

The `done` variable is currently never set and the connection runs until an exception occurs. This would be modified to control by a user interface

6.7.4 ISCASocketWriter

The socket writer is provided a socket to which to write and a queue from which to acquire the information to be written. When the `run()` method is executed, a loop, controlled by internal and external flags is use to iteratively take an element from the queue and write it to the socket's output stream followed by a `flush()`. If either the internal or external flag is set true, the run method exits.

```
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),true);

boolean terminateFlag = false ;
while( okToRun && !terminateFlag )
{
  // Note: the remove is semi-blocking which allows us to return
  // periodically to check to quit or to do other things
  Object obj = queue.remove( timeoutMs, timeoutNs ) ;
  if  ( null != obj )
  {
    // terminateFlag =
    out.print( obj ) ;
    out.flush() ;
  }

  yield() ;
}
```

6.8 Message Context Input Adaptor

As part of treating the event-brokers as part of an ecology of nodes that comprise the system, exposing and sharing system context is critical. This is partially handled through an I/O adaptor referred to as the message context input adaptor. This adaptor sits between the I/O component and the input queue, primarily to capture the identity of the sender and the time of arrival. In Figure 6-1, the adaptor, shown in blue, is labeled "mod" for modifier.

The adaptor takes an input <ISCAMsg> message and will extend it with system level contextual information when none is provided or will add or alter key values. The metadata so constructed is passed on throughout the system, providing the primary mechanism by which to further observe and control event traffic. The current information potentially carried is shown above in the Message Context Component section. The adaptor serves two critical purposes.

First, it assists in managing connections independent of the physical connection. It does this by providing a locally unique sender id that is used to identify the connection. This provides the mapping between the desired destination and the appropriate queue. This also provides an assurance of sufficiently unique names that, in a distributed system, cannot be guaranteed to be unique. For example, the <sender> will be altered to be the `hashcode()` of the socket on which it is received. Currently, no mechanism exists for a disconnection and reconnection with retention of existing advertisements, subscriptions, etc. This is left for future work.

Second, it provides system monitoring information that is manipulated by the various event-brokers through which it passes. This can be considered as piggy-backing additional contextual data about the message of interest. Currently there are three items that are used in this fashion, `<hopCnt>`, `<arrivalTime>` and `<travelTime>`. The hopCnt is incremented at each node through which is passes. The arrival time currently measures the time at which the adaptor receives the data. Upon forwarding the message, the system time is again computed, the arrival time is subtracted, and the travel time is updated accordingly. Future work intends to estimate time waiting on the socket before the data is processed.[56]

6.9 Publishers

Publishers represent any information producer (e.g., network enabled sensor). They connect to the system through one of the supported I/O mechanisms. After connecting,

[56] We would have done so already if Java provided FIONREAD to query data available on a socket.

the publisher performs three key actions.

First, the publisher sends a configuration message <ISCAConfig> to the event-broker to describe the capabilities of the publisher. The basic configuration for a publisher is to provide only its identity to the event-broker. This is only pro-forma and currently has no direct value. The publisher desires no other information from the event-broker to which it is connected.

Next, the publisher submits one advertisement for every event-type it will later produce as a notification. For some efficiency gain, these can all be packed in a single message bundle.

The event-type need only be a subset of the elements in the notifications. As discussed under the event-broker section, the advertisement will flood-forward to all elements capable of accepting advertisements unless otherwise annotated.

Then, at whatever reporting-rate the publisher (e.g., sensor) is configured to use, it will generate event notifications. By convention this information will always be within the confines of the previously reported advertisements.

A pub / sub client need not simply be a publisher, it might also be a subscriber as detailed below.

6.10 Subscribers

Subscribers represent any information consumer, whether agent or application under the direct control of a user. Subscribers connect to the system through one of the supported I/O mechanisms. After connecting, the subscriber performs three key actions.

First, the subscriber sends a configuration message <ISCAConfig> to the event-broker to describe the capabilities of the subscriber. The basic configuration for a subscriber is to provide its identity to the event-broker and that it is willing to accept notifications (in general). By convention, the subscriber will only be passed notifications that match the subscriber's subscriptions.

Next, the subscriber submits one subscription for every event-type it hopes to later receive as a notification. We do not bundle these. The event-type need only be a subset of the elements in the notifications. As discussed under the event-broker section, the subscription will trace back through the system to all elements providing matching advertising.

Then, when publishers generate matching notifications, they will be routed down every path where an originating, matching subscription can be found.

It is currently a design choice to allow the possibility of a subscriber to fail to configure itself to receive notifications yet still generate subscriptions. This is because the connections do not differentiate between event-brokers, publishers, and subscribers. It would probably be useful to require subscribers to accept notifications.

6.11 Event-Broker Forwarding Database

At the heart of the content-based routing mechanism is the database containing the advertisements and subscriptions and the associated content-based routing algorithms. The seminal event matching work comes from SIENA [Carzaniga-1998] where a *covering relationship* is described as a means of determining useful relationships between advertisements and advertisements, subscriptions and subscriptions, advertisements and subscriptions, and subscriptions and notifications.

The relationships can be more succinctly and clearly described in terms of sets and the matching of one attribute-constraint list to another or the matching of an attribute-constraint list to an attribute-value list. Additionally, the attribute-value list can be conceived of as an attribute-constraint list where every operator is set to "equal." In treating the relationships this way, we untangle them from the semantics of advertisements, subscriptions, and notifications, such we are able to reuse the "event matching" capabilities for deployment slips, routing slips, and express forwarding slips as well as message context filtering

There are two levels of set manipulations that need to occur simultaneously (or at least sequentially). The first is the set relationship between the ***fields*** that occur in the two lists. The second level is the set relationship between the ***content*** within the fields that match. We discuss these two levels in the following sections.

6.11.1 Field Relationships

Although we can divorce the computations from the semantics to gain reuse, we must still discuss the field relationships in the context of usage. Table 6-2 and Table 6-3 provide a high level overview of the field relationships as we attempt to match a row item against a column item.

Table 6-2 Field Relationship Table A

	Advertisement	Subscription	RoutingSlip(S)	RoutingSlip(N)
Advertisement	Subset/Superset	Superset	Superset	N/A
Subscription	Subset	Subset/Superset	N/A	Subset
RoutingSlip(S)	Subset	N/A	N/A	N/A
Notification	N/A	Superset	N/A	N/A
RoutingSlip(N)	N/A	Superset	N/A	N/A

The parenthetical (S) and (N) indicate the type of routing slip. Because deployment slips and express forwarding slips are essentially a form of routing slip, we do not add extra rows or columns but rather refer to the routing slip entries. The subset/superset combination indicates that either a subset or a superset relationship may apply.

The two places where subset and superset are both listed demonstrate efficiency measures. We prevent propagation of effectively redundant items (i.e., those that are subsumed by prior items of the same type) when we check to see if a new advertisement is subsumed by an existing advertisement or similarly when a subscription is subsumed by another subscription. The obverse of this efficiency concern requires order independence; we need to eliminate existing items that are subsumed by the new items to reduce future computational costs. In this case we need the superset perspective. Due to the matching algorithms discussed later, it is not merely a matter of using the same function and swapping the parameters, we actually need to support both subset and superset behaviors. These evaluations only exist with respect to items received from the same source.

All other field relationships follow in a straightforward way. Because no persistence for notifications exists, the entire column has been deleted. However, routing slips of type notification may be given persistence, in which case we will need to check them.

Table 6-3 Field Relationship Table B

	Msg Context	Msg Ctx Filter
Msg Context	N/A	Superset
Msg Ctx Filter	Subset	N/A

Omitted from the first table are the message context and message context filters because they are disjoint. Their relationships to each other are shown in Table 6-3.

6.11.2 Content Relationships

Given the appropriate set relationships between the fields, then we apply the content relationships as shown in Table 6-4 and Table 6-5.

Table 6-4 Content Relationship Table A

	Advertisement	Subscription	RoutingSlip(S)	RoutingSlip(N)
Advertisement	Superset/Subset	Intersection	Intersection	N/A
Subscription	Intersection	Subset/Superset	N/A	Superset
RoutingSlip(S)	Intersection	N/A	N/A	N/A
Notification	N/A	Subset	N/A	N/A
RoutingSlip(N)	N/A	Subset	N/A	N/A

Table 6-5 Content Relationship Table B

	Msg Context	Msg Ctx Filter
Msg Context	N/A	Subset
Msg Ctx Filter	Superset	N/A

Subscription content is an ***intersection*** with advertisement content when subscriptions are routed toward the publisher source of advertisements. This means that whenever an advertisement holds the possibility of a publisher providing a notification that matches a subscriptions, the subscription should be routed toward the publisher.

When eliminating redundant advertisements from the same source, then an advertisement whose content is a ***superset*** of another advertisement may eliminate the smaller advertisement as unnecessary. Checks must be performed in both directions.

When eliminating redundant subscriptions from the same source, then a subscription whose fields are a ***subset*** of another subscription may, but whose contents are a *superset*, can eliminate the subscription with more fields. Checks must be performed in both directions.

6.11.3 Set Relationship Summary

The relationships between a pair of attribute-constraint lists or between an attribute-constraint list and an attribute-value list can be summarized as the set relationships between ***fields*** X ***contents*** (of the matching fields) as follows:

{ Subset, Superset } x { Subset, Superset, Intersection }

Thus, the database requirements are to support these combinations with the natural insert and delete capabilities. These combinations must be applicable both between individual items (e.g., a specific message context with respect to a message context filter) as well as between an individual item and the entire database (e.g., a new subscription against all advertisements).

6.11.4 Forwarding Database Algorithms

The fast-forwarding algorithm [CW-2003] makes use of a counting algorithm for efficiently determining when a match has occurred. Using this technique, Fulcrum provides matching capabilities for any of the set relationship combinations. This technique, with our adaptations, is shown in Figure 6-4. On the left-hand side we have a hash-table (one for each interface – i.e., connection) holding references to each attribute ever referenced on that connection. In the second column, we represent all attribute constraints. These are organized by relational operators, <, <=, =, >=, >, and ANY, to allow us to quickly find the appropriate matches. These attribute constraints then point to their associated filters (e.g., subscription). The filter is annotated with a "size" to indicate the number of constraints to be matched and also holds an internal count of the number of constraints already matched. Finally, the filter is actually a message component of an

ISCAMsg, so it points to the ISCAMsg. In the case of an event matching a subscription, only the event is forwarded to the connection from which the subscription originated. However, in the case of a new advertisement matching a subscription, the entire ISCAMsg is passed back to the advertisement's connection.

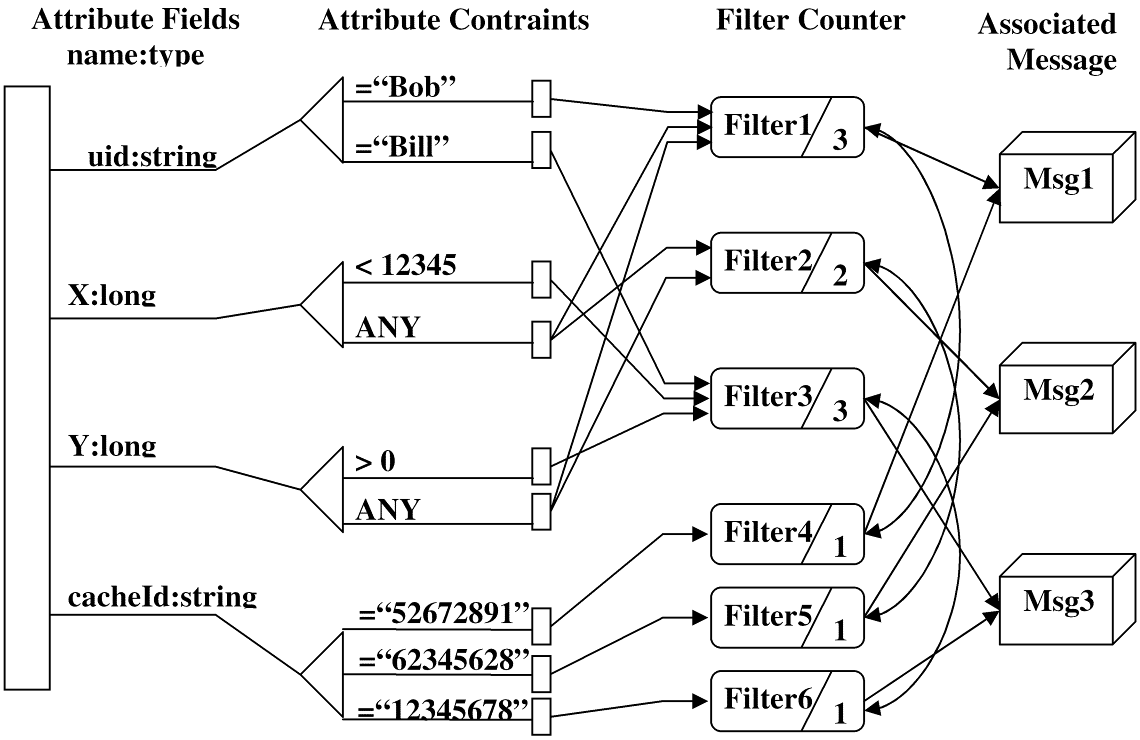


Figure 6-4 Fast Forwarding Algorithm Diagram

Also shown is the distributed memoization implemented as single-attribute filters, where the attribute name is "cacheId." At an edge-broker, the N-attribute matching first occurs. On success, a express forwarding slip is created an prepended to the ISCAMsg, such that at the internal nodes, only single-attribute matching is required. Pseudo code for our implementation of the fast forwarding algorithm is shown in Figure 6-5.

```
findMatches( messageComponent mc, callbackFunction cbf, clientData cd )
{
 pseudoTimeStamp++ ;
 foreach attribute (a) in messageComponent (mc)
 {
  foreach relationship list (<, <=, =, >=, >, ANY )
  {
   foreach constraint (c) matching attribute a
   {
    if  ( c.f.timestamp != pseudoTimeStamp )
    {
     c.f.timestamp = pseudoTimeStamp ;
     c.f.counter = 1;
    }
    else
     c.f.counter++ ;
    if  ( c.f.counter == c.f.size )
     if  ( cbf( mc, c.f, c.f.msg, cd ) )
      return( true ) ;
   }
  }
 }
}
```

Figure 6-5 Fast Forwarding Algorithm Pseudo Code

Our findMatches algorithm has been presented in a generalized form. Actual details depend on subset/superset matches for the field and subset/superset/intersection matches for the contents. It works as follows.

1. Rather than initializing additional sets to monitor the counting, at a cost of O(N), we instead maintain a pseudoTimeStamp, a monotonically increasing counter that is incremented on each call to the algorithm. Later to check current set membership, i.e., the filter is being counted, we check that the timestamps match.

2. We iterate through all the attribute fields of a message component (e.g., a routing slip). Remember, there may be multiple message components in a message. Iterating through the message components occurs outside of findMatches().

3. As matching constraints are discovered, they are counted against the size of the filter. When all attribute constraints of a filter have been matched, the callback function is

invoked to check whether or not additional limitations (e.g., message context filters) have also been passed. If so, the message is forwarded, then callback function returns true, and the findMatches algorithm terminates.

Although the published research joins together all attribute fields for all interfaces into a single large table, we have chosen to manage a separate table of fields for each interface. The benefit of our approach is that once a message has been forwarded on an interface, we need never again concern ourselves with additional entries for that interface. And, because we have developed the distributed memoization technique we may quickly determine that a given interface has been matched and avoid attempting to match against all the attribute constraints within that interface. Our added cost is that instead of a single iteration over all matching fields, we must iterate over all matching fields for each interface. Although we have not experimented to find the exact tipping point, we can see that a small database favors the single table clustering (the old way) while large databases favor iterating over each interface (our new way). To support Internet-scale, we must plan for the larger databases.

Due to the challenges with the Java container capabilities for providing functions that yield all elements within a range [x,y] – the challenge being that the functions only provide [x,y), two consequent implementation choices were made. First, the < and <= were separated into two lists, as were the >= and >. Second, a special comparator function was devised that accepts objects with an `inclusiveFlag` to perform an inclusive comparison.

At this point, string relationships are limited to equality checking. However, the fast forwarding algorithm research [CW-2003] describes how to provide string prefix, substring, and suffix matching. These will be added in the future when actual use cases are discovered.

6.11.4.1 Callback Function

A callback function is provided such that the event-broker with the knowledge of "what to do on match" can take appropriate action. A return code on the callback function determines whether or not the matching function should continue. This is useful because the nominal behavior is to send a message – advertisement, subscription, or notification – down a given connection only once. Thus, as soon as the 'forward' determination has been made by the callback function, the database can quit checking for certain channels.

6.11.5 Forwarding Database Summary

The event-broker's forwarding database encapsulates the data storage and algorithms for performing field and content matching between advertisements, subscriptions, notifications, and routing slips as well as for message context and message context filters. While the database executes the algorithms, it must pass back control to the event-broker, through a callback function, in order for the appropriate action to be taken when a match is found as well as for early detection of when to quit searching for matches.

6.12 Event-Brokers

The event-brokers form the overlay network and manage the system semantics. The overlay network provides an application layer over the Internet by which information is routed using the content-based routing semantics. The network connectivity should be mapped to approximate physical connectivity of the underlying nodes [Carzaniga-1998]. Continuing research is examining ways to make the connections more flexible based on content so as to gain the value of the covers relationship. The event-broker is the essential component of the system. Its primary relationships are shown in Figure 6-3.

At a high level, an event-broker only does two things. First, it accepts and manages connections from other event-brokers and possibly pub / sub clients. Second, it accepts and manages messages across these connections. This includes achieving all of the core system semantics. These are facilitated by the forwarding database previously discussed.

First, behavior with respect to connection management is described. Next, the distributed memoization / caching techniques are presented. Then we discuss the primary message semantics. And finally, we detail behaviors associated with active subscriptions and their implementation strategies. .

6.12.1 Connection Management

Each time a new connection is detected or removed, then a `<ConnectionService>` internal message is sent to the event-broker. On receipt of such a message, we add or remove internal knowledge of the connection. For each connection, all the messages received are grouped for searching as well as for removal when the connection is done.

6.12.2 Distributed Memoization / Caching Computations

One of the efficiency enhancements we make is to make the system an ecology such that computations performed in one event-broker can be memoized and shared with other event-brokers, such that they do not need to re-compute everything from scratch.

When a subscription is received at the subscriber's event-broker, it is given a cacheId by which it may later be referenced. This cacheId is placed in a single-attribute subscription, attached to the message, and entered into the database

Then, when a notification is received at the publisher's event-broker, all[57] subscriptions are checked to determine whether or not to forward the data. And, we know that if the data has been forwarded once, we should expect it to be forwarded all the way down the subscriber responsible for the matching subscription. Therefore when the notification is received and a match is found, we take the cacheId from the matched subscription, construct a single-attribute express forwarding slip, add it to the start of the message – before the regular notification, and send the whole message to the associated connection. This process is repeated for all connections.

Thus, when a notification is received at subsequent event-brokers, it is preceded by the express forwarding slip which is easily matched. However, this express forwarding slip may only be good for a single connection. Therefore we must repeat the process for each of the other connections as if we were receiving the data directly from a publisher. This technique is equally useful for subscriptions matching advertisements.

6.12.3 Message Semantics

Whenever an advertisement, subscription, or active subscription arrives at an event-broker, it is stored in an appropriate set of tables in the event-broker's local database. This is important in that it allows further system behavior to act independent of the order of advertisements or subscriptions, thus granting the intuitively obvious behavior of allowing a pub / sub client to plug-in and get the information it wants or get its information to the appropriate consumers. The semantics associated with each message is detailed in the following sections.

[57] Logically "all." In reality, the forwarding techniques avoid wasteful computation.

6.12.3.1 Advertisements

When a new advertisement is received from a given source, five activities take place.

First, we check the aggregation flag (default is true). Omission or any value other than false is considered true. If aggregation is set to false, then we bypass the next two aggregation steps.

Second, to determine whether the new advertisement is a subset of a prior advertisement we use the field-subset X content-subset for the database find function.

```
tableToSearch = "Advertisement"
controlFlags = { "Subset", "Subset" }
setup callback function
setup callback data
db.find( sender, tableToSearch, messageContext, domElm,
         currNode, ctrlFlags, cb, cbData )
```

If a match is found, then the advertisement is eliminated using basic CBPS practices.

Third, we check to determine whether the new advertisement is a superset of a prior advertisement. This is field-superset X content-superset for the database find function.

```
tableToSearch = "Advertisement"
controlFlags = { "Superset", "Superset" }
setup callback function
setup callback data
db.find( sender, tableToSearch, messageContext, domElm,
         currNode, ctrlFlags, cb, cbData )
```

If a match is found, then the lesser advertisement is eliminated using basic SIENA practices and if the flood forwarding flag is not false, then the current advertisement is set for forwarding. But, first, any outgoing message context filters are evaluated, and if they pass, the message is forwarded.

Fourth, to provide the expected behavior that any processing element can plug into the network at any time and get the information that is appropriate to them means making sure the system is order independent with respect to when advertisements or subscriptions arrive. To deal with this, every time a new advertisement is received, it is evaluated against all standing subscriptions such that the subscriptions can be forwarded toward the origination of the advertisement.

```
tableToSearch = "Subscription"
controlFlags = { "Superset", "Superset" }
setup callback function
setup callback data
db.find( sender, tableToSearch, messageContext, domElm,
         currNode, ctrlFlags, cb, cbData )
```

The same activity is performed with the tableToSearch = "RoutingSlipSubscription."

Fifth, we check the message context to determine if the hopCnt is one. If so, we set the tableToSearch = "DeploymentSlipImplementationStrategy." When matches are found, the associated implementation strategy is instantiated, as discussed in the implementation strategy section below.

Each time a new event-broker (or actually any processing element that claims ability to receive advertisements) is connected, all existing advertisements are shared. This allows separate networks to combine after they have been running for some time. However, there is some risk of over-burdening a connection. Some form of rate limiting should be included in future work.

6.12.4 Subscriptions

When a new subscription is received from a given source, four activities take place. First, we check the aggregation flag (default is true). Omission or any value other than false is considered true. If aggregation is set to false, then we bypass the next two aggregation steps.

Second, to determine whether the new subscription is a subset of a prior subscription we use the field-superset X content-subset for the database find function.

```
tableToSearch = "Subscription"
controlFlags = { "Superset", "Subset" }
setup callback function
setup callback data
db.find( sender, tableToSearch, messageContext, domElm,
         currNode, ctrlFlags, cb, cbData )
```

If a match is found, then the advertisement is eliminated using basic SIENA practices.

Third, we check to determine whether the new subscription is a superset of a prior subscription. This is field-subset X content-superset for the database find function. The apparent inversion is that for a subscription to be more encompassing, it has fewer constrained fields and the content limitations of those fields are broader in scope.

```
tableToSearch = "Subscription"
controlFlags = { "Subset", "Superset" }
setup callback function
setup callback data
db.find( sender, tableToSearch, messageContext, domElm,
         currNode, ctrlFlags, cb, cbData )
```

If a match is found, then the more constraining subscription is eliminated using basic CBPS practices.

Fourth, if the new subscription is broader in scope or is set for no aggregation then it is stored to promote order independence. If the flood forwarding flag is set, it is marked to be automatically forwarded over all connections (except the one on which it was received). Otherwise, it is checked against existing advertisements.

```
tableToSearch = "Advertisement"
controlFlags = { "Subset", "Superset" }
setup callback function
setup callback data
db.find( sender, tableToSearch, messageContext, domElm,
         currNode, ctrlFlags, cb, cbData )
```

On a successful match, the subscription is marked for forwarding. But, first, any outgoing message context filters are evaluated, and if they pass, the message is forwarded.

Each time a new event-broker (or actually any processing element that claims ability to receive subscriptions) is connected, all existing subscriptions with the flood forwarding flag set true are shared. This allows separate networks to combine after they have been running for some time. However, there is some risk of over-burdening a connection. Some form of rate limiting should be included in future work.

6.12.5 Notifications

When a notification is received from a given source, three activities take place.

First, we check the flood forward flag (default is false). Omission or any value other than true is considered false. If flood forwarding is set to true, then we forward the notification on all connections that have been configured to receive notifications.

Second, any outgoing message context filters are evaluated against the message context. If all fail, then the notification is discarded. Otherwise, the notification is checked against existing subscriptions to determine whether it should be forwarded.

```
tableToSearch = "subscription"
controlFlags = { "Superset", "Subset" }
setup callback function
setup callback data
db.find( sender, tableToSearch, metaLocal, domElm,
         currNode, ctrlFlags, cb, cbData )
```

When the notification satisfies a subscription, then the subscription's incoming message context filter is checked against the notification's message context filter. If successful, then the message will be forwarded. This is the point at which the cacheId of the subscription is passed over to the notification as its matchId. Despite the possibility of multiple matching subscriptions, only a single notification is passed down any given connection.

6.12.6 Routing Slips

A routing slip is a mechanism by which we can leverage the content-based routing of a message through the system without being locked into all the semantics of subscriptions or notifications.

Because the intent is to move some associated information components through the system, routing slips are inherently inert; they are set to non-aggregation and have no flood forwarding flag. Thus, the behavior of a routing slip of type subscription is effectively equivalent to a subscription with both aggregation and flood forwarding flags set false, except that it can never be matched by a notification. If the persistence flag is set false, then the routing slip is not stored for later reference. A routing slip of type notification behaves like a notification.

6.12.7 Active Subscriptions

Active Subscriptions are unique to Fulcrum.[58] As previously discussed, they are

[58] Hill and Knight use the term active subscription in a similar yet different way. The semantics and mechanics discussed are unique to Fulcrum.

logically an extension to a regular subscription. Consequently a regular subscription and an active subscription may reside in the same message bundle. As with the use of a routing slip, if any component of the message bundle fulfills forwarding requirements, then the entire bundle is forwarded. In this way, the subscription for the derived event – the actual event of interest – can setup a chain of subscriptions to the event-broker where the event will actually be generated without the cost of an extra advertisement. It should be noted that the active subscription can stand on its own; no subscription is actually required. Thus, the implementation strategies of an active subscription may be launched to where the appropriate data enters the network. The active subscription is treated as if it too is a message bundle, composed of a set of implementation strategies.

6.12.7.1 Implementation Strategy

An implementation strategy provides the "what to do" under a given set of conditions when data first enters the network. It is combined of three components, the deployment slip that dictates where to instantiate the implementation strategy, the instantiation class and the other initialization parameters.

6.12.7.1.1 Deployment Slip

The deployment slip in each implementation strategy is used to determine propagation. Just as the arrival of a new advertisement or subscription results in storing that component, so too will the arrival of an active subscription cause it to be stored. Because we really want to have the implementation strategy co-located with where sensor data enters the network, the deployment slip is normally of type subscription.

6.12.7.1.2 Instantiation Class

The primary class of each implementation strategy component is identified as the *instantiation class*. This class is required to implement the IISCAAgent interface.

```
public interface IISCAAgent extends IISCAClient, ISetQueues, IThread
{
  // Should an agent need to support a multi-I/O situation, they
  // can clearly identify their primary connection as the parent
  public void setParentUID( String uid ) ;
  public String getParentUID() ;

  public void setArgs( String [] args ) ;

  public void init() ;
}
```

In the case where an implementation strategy might serve as a bridge between multiple networks, then the parent id (then name of the event-broker) can be provided such that it can differentiate between its connections.

When the implementation strategy is instantiated, described below, the primary class is constructed, set with the parentUID, passed an array of initialization values, and finally init() is called to kick off the real processing.

6.12.7.1.3 Initialization Parameters

The initialization parameters provide the strings that feeds setArgs(). For evolutionary reasons, this is being provided as the string representation of the initialization parameters XML.

```
<uid value=”A”>
<partnerUid value=”B”>
<proximityRange value=”10”>
<reportingResponsibility value=”true”>
```

6.12.7.1.4 Instantiation

When an advertisement is found whose metadata claims it has arrived locally (i.e., message context hopCnt = 1) then the implementation class is invoked within a deployment wrapper. The deployment wrapper sets up the queues, instantiates the class, attaches the queues, and invokes init().

6.12.7.1.5 Behavior

An implementation strategy is a first-class pub / sub client. When instantiated, we call it an agent. Although connection to the network is handled by the event-broker during the deployment, the remainder of the behavior is up to each instance. In general however, we can say that an implementation strategy is a communications wrapper around a portion of a distributed algorithm. The communications wrapper handles all the publishing and subscribing while the algorithm does the real work. The typical behavior is as follows.

After instantiation and initialization, logically, the implementation strategy subscribes to the information of interest, subscribes to data to be produced by collaborating algorithms, and advertises the events it will generate. However, as discussed in Section 4, we wish to avoid the overhead associated with flooding the network with extra advertisements (or subscriptions). Hence, we employ three strategies.

First, the implementation strategy sets up a subscription to the same data to which it was routed. This subscription will be set with a message context filter that limits the hopCnt to one to prevent the subscription from being propagated and to prevent spurious matches from being accepted.

Second, the active subscription is normally paired with a subscription. The subscription gets carried along in the wake of the active subscription – all the way to the event-brokers at the edge of the network where the implementation strategies are instantiated. Thus, when the derived event is constructed, it can be passed back along the chain of subscriptions to the original subscriber. In this way the implementation strategy does not need to flood the network with an advertisement of the events that it derives.

Third, the implementation strategies setup one-to-one communication pathways with each other by using routing slips. These routing slips are set to be equivalent to the routing slip used to deploy the partner's strategy. That is, A is deployed using routing slip "a" and B is deployed using routing slip "b." Then, A sets up its connection to B using routing slip "b" and B sets up its connection to A using routing slip "a." The knowledge of an affiliate must be passed in as an initialization parameter.

Then, when data is received, it is parsed and the appropriate function of the algorithm is invoked. This is usually something like setValue(sensorReading). However, there are also control functions like setOriginAndRangeValue(newOrigin, newRange) that are invoked as part of the collaborative process.

When the algorithm determines that a semantically significant event has occurred and needs to be communicated, that information is passed back to the communications layer to make and send the message.

An important value of this separation of concerns is that the algorithm can be developed and tested independently of the communications.

6.12.7.2 Implementation Code

The implementation code is comprised of a collection of classes that are required to run the implementation strategy. This can, in the simplest terms, be considered a .jar file. However, it is not, for the simple reason of composition and long term extensibility. Instead it is a list of the classes (and their code) supporting the implementation strategy. Future work should consider dynamic generation of a .jar file.

6.12.8 Deployment Wrapper

The deployment wrapper serves two core purposes. It isolates the implementation strategy of an active subscription from the rest of the system (i.e., sandboxes it), so as to be able to provide security, resource utilization control, etc., which we haven't worried about yet. In the isolation, the wrapper behaves in the fashion of a communications transport. The implementation strategy need not be implemented within the event-broker (although it is now); it could be moved outside onto a secondary platform. The changes required to make this happen are localized to the deployment wrapper.

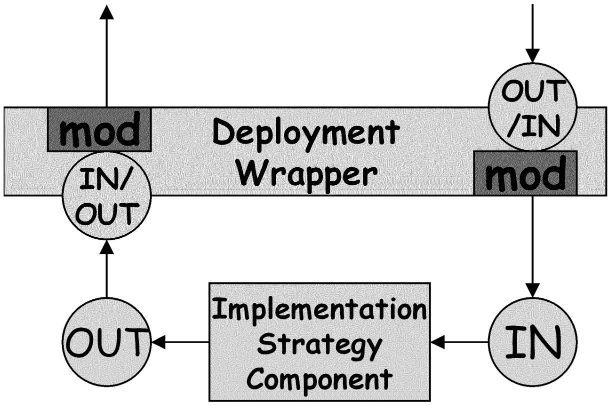


Figure 6-6 Deployment Wrapper Isolated Implementation Strategy

6.12.9 Client Mobility

In our proximity example, the entities move. Efficiency considerations suggest that a client (publisher or subscriber) should be close to its associated middleware node. There are two straightforward mechanisms by which to address client mobility.

6.12.9.1 Default CBPS Semantics

CBPS focuses on content rather than identity. The rehosting of either client is of no consequence to the other – just so long as the appropriate events are propagated through the network. A mobile publisher can abandon its access point, move to a new location, and re-register itself. Upon disconnection, the middleware will remove the active subscription and later a new one will be introduced when the publisher reconnects to the network. The active subscriptions then resynchronize as if starting from scratch. A similar scenario exists for moving the subscriber. In both cases, the publisher and subscriber must have the contextual awareness to know they are exiting and reentering the network. Also, the active subscription must be designed robustly to support such activities.

6.12.9.2 Mobility Service

If a mobility service were available, then the pub/sub client need not be aware of any disconnected state [CCW-2003]. Such services automatically handle the separation with moveOut/moveIn APIs [CDF-2001, Caporuscio-2002]. They use a proxy at the old access point, buffering any messages, until such time as the client reconnects [CDF-2001, CCW-2003, FGKZ-2003]. Active subscriptions do not present any additional mobility problems.

In either case, race conditions can arise. Most notably, if the publisher's state satisfies a subscription while disconnected, the event may not be pushed into the network. If events are buffered via a mobility service, then an event may get delivered more than once, requiring disambiguation at the subscriber.

6.13 Implementation Strategy Agents

An implementation strategy instance, an agent, follows a common pattern for which we are able to define an interface and an abstract base class. The interface, as we have seen previously, is the IISCAAgent. In the following sections, we show the code for the abstract agent and a few concrete instances.

6.13.1 Abstract Agent

The abstract agent provides a typical base class instantiation, purposefully omitting the three functions where agents differ. Those functions are setArgs(), init(), and process(). The general behavior is to run in an infinite loop, waiting for control terminating flags to be set. Inside the loop, it performs a semi-blocking read from the input queue. The block is for a limited time (e.g., 1000ms) at which point we are able to check the control flags.

```
package ISCA.Agent ;

import java.util.* ;

import ISCA.Agent.* ;
import ISCA.Queue.* ;

abstract class AbstractAgent
      extends Thread
      implements IISCAAgent
{
  String uid ;
  String parentUid ;
  IQueueSynch inputQueue ;
  IQueueSynch sendQueue ;

  public void setUID( String uid ) { this.uid = uid ; }
  public String getUID() { return( uid ) ; }

  public void setParentUID( String uid ) { this.parentUid = uid ; }
  public String getParentUID() { return( parentUid ) ; }

  public void setInputQueue( IQueueSynch queue ) {inputQueue = queue ;}
  public void setSendQueue( IQueueSynch queue )  { sendQueue = queue ;}

  boolean okToRun = true ;
  long timeoutMs = 1000 ;     // 1 second wake-up period
  int  timeoutNs = 0 ;

  public void ithread_start() { okToRun = true ; start() ; }
  public void ithread_stop() { okToRun = false ; }

  public void run()
  {
    boolean terminateFlag = false ;

    while( okToRun && !terminateFlag )
    {
      // Note: the remove is semi-blocking which allows us to return
      // periodically to check to quit or to do other things.
      Object obj = inputQueue.remove( timeoutMs, timeoutNs ) ;
      if  ( null != obj )
        terminateFlag = process( obj ) ;
    }
  }

  abstract public void setArgs( String [] args ) ;
  abstract public void init() ;
  abstract public boolean process( Object obj ) ;
}
```

6.13.2 Observer / Transformer / Filter Agent

A concrete instance of an implementation strategy typically serves three roles, that of observer, transformer, and filter (OTF). There are several steps that occur once an OTF is instantiated.

The first step is to receive initialization parameters through the setArgs() function. By convention, these are merely a list of XML strings as passed in an active subscription. For simplicity, they are concatenated together to form an XML string that may be converted into a document object model using available XML support functions. We then iterate through the results to set internal variables as necessary. Each OTF may define its own unique initialization parameters.

Then the init() function is invoked to start the core processing. This typically results in the same three main action we find in any pub / sub client. First, it registers itself, next it advertisement event-types, subscribes for event-types, or use routing slips to create one-to-one conversations with associated agents. Additionally, as shown in the example, other internal variables and objects are initialized (e.g., the world model).

The agent then sits dormant awaiting events. When an event is received originating from the sensor being observed, the world model is updated and typically the data is filtered out. On occasion, an update event will be generated. When an event is received containing control instructions, the appropriate actions are performed.

Sample code for a freeway sensor is shown below.

```
package ISCA.Agent ;

import java.util.* ;

import org.w3c.dom.*;
import ISCA.Agent.* ;
import ISCA.Queue.* ;
import ISCA.Tools.* ;
import ISCA.Balance.* ;


public class ObserverAgent
      extends AbstractAgent
      implements IBalanceMediator
{
  String uid = null ;
  String ctrluid = null ;
  double scalar = 1.0 ; // consider y = mx + b
  double offset = 0.0 ; // then scalar = m;   offset = b;
  boolean reciprocal = false ;  // to deal with cases where we need 1/x
  BalanceObserver balanceObserver = null ;


  public void updateItem( String id, double value )
  {
    System.err.println(
      "\n\nUpdate of: " + uid + " --value--> " + value + "\n\n" ) ;

    sendNotification( value ) ;
  }


  public void setArgs( String [] args )
  {
    String xmlArgs = "" ;
    for( int i = 0 ;   i < args.length ;   i++ )
    {
      System.err.println( "ObserverAgent args[ " + i + " ] = " +
args[i] ) ;
      xmlArgs += args[i] ;
    }

    Node node = DOMTools.XMLToNode( "<Args>" + xmlArgs + "</Args>" ) ;

    node = node.getFirstChild().getFirstChild() ;
    while( null != node )
    {
      if  ( node.getNodeName().equals( "uid" ) )
        uid = ((Element) node).getAttribute( "value" ) ;
      else if  ( node.getNodeName().equals( "ctrluid" ) )
        ctrluid = ((Element) node).getAttribute( "value" ) ;
      else if  ( node.getNodeName().equals( "scalar" ) )
        scalar = Double.parseDouble(
            ((Element) node).getAttribute( "value" ) ) ;
      else if  ( node.getNodeName().equals( "offset" ) )
        offset = Double.parseDouble(
            ((Element) node).getAttribute( "value" ) ) ;
      else if  ( node.getNodeName().equals( "reciprocal" ) )
        reciprocal =
            ((Element) node).getAttribute( "value" ).equals( "true" ) ;
    node = node.getNextSibling() ;
  }
}


public void init()
{
  timeoutMs = 4002 ;  // testing to identify usage
  setPriority( Thread.MIN_PRIORITY ) ;

  sendRegistration() ;
  sendAdvertisements() ;
  sendSubscriptions() ;

  balanceObserver = new BalanceObserver( uid ) ;
  balanceObserver.resetLastReportedValue( 0.0 ) ;
  balanceObserver.resetRangeValue( 0.0 ) ;
  balanceObserver.setMediator( this ) ;
}


public void sendRegistration()
{
  String msg ;

  msg = "<ISCAConfig><Registration>" ;
  msg += "<Notification/>" ;
  msg += "</Registration></ISCAConfig>" ;

  sendQueue.append( msg ) ;
}


public void sendSubscriptions()
{
  String msg = "<ISCAMsg><Subscription>" ;
  msg += "<ac name=\"uid\" type=\"string\" op=\"EQ\" value=\""
    + uid + "\"/>" ;
  msg += "<ac name=\"ctrluid\" type=\"string\" op=\"EQ\" value=\""
    + ctrluid + "\"/>" ;
  msg += "<ac name=\"action\" type=\"string\" op=\"ANY\"/>" ;
  msg += "</Subscription></ISCAMsg>" ;

  sendQueue.append( msg ) ;
}


public void sendAdvertisements()
{
  String msg = "<ISCAMsg><Advertisement>" ;
  msg += "<ac name=\"uid\" type=\"string\" op=\"EQ\" value=\""
    + uid + "\"/>" ;
  msg += "<ac name=\"ctrluid\" type=\"string\" op=\"EQ\" value=\""
    + ctrluid + "\"/>" ;
  msg += "<ac name=\"value\" type=\"double\" op=\"ANY\"/>" ;
```

```
    msg += "</Advertisement></ISCAMsg>" ;

    sendQueue.append( msg ) ;
  }


  public void sendNotification( double value )
  {
    String msg = "<ISCAMsg><Notification>" ;
    msg += "<av name=\"uid\" type=\"string\" value=\"" + uid + "\"/>" ;
    msg += "<av name=\"ctrluid\" type=\"string\" value=\""
      + ctrluid + "\"/>" ;
    msg += "<av name=\"value\" type=\"double\" value=\""
      + value + "\"/>" ;
    msg += "</Notification></ISCAMsg>" ;

    sendQueue.append( msg ) ;
  }


  public boolean process( Object obj )
  {
    // By the design of the active subscription, we will get data of
interest with speed
    // By addition of subscription to controlling actions, we can get
other data types
    // It would be nice to be able to verify the speed input came from
the sensor and not some liar.

    String speedStr = null ;
    String rangeStr = null ;
    String originStr = null ;
    String opIndexStr = null ;
    String actionStr = null ;

    Node node = DOMTools.XMLToNode( (String) obj ) ;
    node = DOMTools.getNextNode(
      node.getFirstChild().getFirstChild(), "Notification" ) ;
    node = node.getFirstChild() ;
    while( null != node )
    {
      if  ( node.getNodeName().equals( "av" ) )
      if  ( ((Element) node).getAttribute( "name" ).equals( "speed" ) )
        speedStr = ((Element) node).getAttribute( "value" ) ;
      else if  ( ((Element) node).getAttribute("name").equals("range"))
        rangeStr = ((Element) node).getAttribute( "value" ) ;
      else if  (((Element) node).getAttribute("name").equals("origin"))
        originStr = ((Element) node).getAttribute( "value" ) ;
      else if (((Element) node).getAttribute("name").equals("opIndex"))
        opIndexStr = ((Element) node).getAttribute( "value" ) ;
      else if  (((Element) node).getAttribute("name").equals("action"))
        actionStr = ((Element) node).getAttribute( "value" ) ;
      // else unknown or uninteresting value

      node = node.getNextSibling() ;
    }

    if  ( null != actionStr )
    {
      if  ( actionStr.equals( "resetRangeValue" )
            &&  ( null != rangeStr ) )
      {
         balanceObserver.resetRangeValue(Double.parseDouble(rangeStr));
         balanceObserver.resetCurrentValue(
            balanceObserver.getCurrentValue() ) ;
      }
      else if  ( actionStr.equals( "resetOriginRangeValue" )
            &&  ( null != rangeStr )  &&  ( null != originStr ) )
      {
         balanceObserver.resetRangeValue(Double.parseDouble(rangeStr));
         balanceObserver.resetLastReportedValue(
            Double.parseDouble( originStr ) ) ;
         balanceObserver.resetCurrentValue(
            balanceObserver.getCurrentValue() ) ;
      }
      else if  ( actionStr.equals( "resetOriginRangeOpValue" )
                 &&  ( null != rangeStr )
                 &&  ( null != originStr )
                 &&  ( null != opIndexStr )
      )
      {
         balanceObserver.resetRangeValue(Double.parseDouble(rangeStr));
         balanceObserver.resetLastReportedValue(
            Double.parseDouble( originStr ) ) ;
         balanceObserver.setOperator( Integer.parseInt( opIndexStr ) );
         balanceObserver.resetCurrentValue(
            balanceObserver.getCurrentValue() ) ;
      }

// else inadequately handled action...
    }
    else if  ( ( null != speedStr )  &&  ( 0 < speedStr.length() ) )
    {
      if  ( reciprocal )
        balanceObserver.resetCurrentValue(
            scalar / Double.parseDouble( speedStr ) + offset ) ;
      else balanceObserver.resetCurrentValue(
            scalar * Double.parseDouble( speedStr ) + offset ) ;
    }

    return( false ) ;
  }
}
```

6.13.3 CalTrans Loop Detector Subscriber

The following shows a sample subscriber

```
package ISCA.Client.Samples ;

import java.util.* ;

import ISCA.Client.* ;
import ISCA.Queue.* ;
import ISCA.Tools.* ;

public class CaltransListener
      extends Thread
      implements IISCAClient, IGetQueues, IThread
{
  boolean okToRun = true ;
  long timeoutMs = 5000 ;     // 5 second interval
  int  timeoutNs = 0 ;

  public void ithread_start() { okToRun = true ; start() ; }
  public void ithread_stop() { okToRun = false ; }


  String name ;
  String type ;
  String urlStr ;
  String barStr ;
  String sensorStr ;
  String routeStr ;
  String directionStr ;

  ISendQueue sendQueue ;
  IQueueSynch inputQueue ;
  IQueueSynch outputQueue ;


  public CaltransListener()
  {
    name = null ;
    type = null ;
    sensorStr = null ;
    routeStr = null ;
    directionStr = null ;

    inputQueue = null ;
    outputQueue = null ;
    sendQueue = null ;
  }

  public void setProperties( Properties props )
  {
    name = props.getProperty( "name" ) ;
    type = props.getProperty( "type" ) ;
    sensorStr = props.getProperty( "sensor" ) ;
    routeStr = props.getProperty( "route" ) ;
    directionStr = props.getProperty( "direction" ) ;

    inputQueue = new FifoSynch() ;
    outputQueue = new FifoSynch() ;
  }


  public String getUID() { return( name ) ; }
  public void setUID( String name ) { this.name = name ; }

  public IQueueSynch getInputQueue() { return( inputQueue ) ; }
  public IQueueSynch getOutputQueue() { return( outputQueue ) ; }
  public ISendQueue  getSendQueue() { return( sendQueue ) ; }


  public void subscribePublications()
  {
      String msg = "<ISCAMsg>" ;

      msg += "<Subscription>" ;
      msg += "<ac name=\"sensor\" type=\"string\" op=\"EQ\" value=\""
        + sensorStr + "\"/>" ;
      msg += "<ac name=\"route\" type=\"string\" op=\"EQ\" value=\""
        + routeStr + "\"/>" ;
      msg += "<ac name=\"direction\" type=\"string\" op=\"EQ\""
        + " value=\"" + directionStr + "\"/>" ;
      msg += "<ac name=\"speed\" type=\"long\" op=\"ANY\"/>" ;
      msg += "</Subscription>" ;

      msg += "</ISCAMsg>" ;

      outputQueue.append( msg ) ;
  }


  public void registerWithParent()
  {
      String msg = "<ISCAConfig>" ;
      msg += "<Registration
request=\"true\"><Notification/></Registration>" ;
      msg += "</ISCAConfig>" ;

      outputQueue.append( msg ) ;
  }


  public void connectedAction()
  {
      registerWithParent() ;
      subscribePublications() ;
  }
  public boolean process( Object obj )
  {
    if  ( null == obj )
    {
      return( false ) ;
    }
    else if ( obj instanceof String )
    {
      System.err.println( "Listening post - information:\n" + obj ) ;

      if  ( ((String) obj).equals( "<Connect/>" ) )
        connectedAction() ;

      return( false ) ;
    }
    else
    {
      System.err.println( "XmlProcessor :: Bad object: " + obj ) ;
      return( true ) ;
    }
  }


  public void run()
  {
    boolean terminateFlag = false ;

    while( okToRun && !terminateFlag )
    {
      Object obj = inputQueue.remove( timeoutMs, timeoutNs ) ;

      if  ( null != obj )
        terminateFlag = process( obj ) ;
    }
  }
}
```

6.14 Sample Sensor – Loop Detector

The most likely place from which information is to be acquired and placed in our Internet-scale context-aware infrastructure is information that is already available on the internet. As we will discuss in the future work section, currently too few people express their information in a readily digestible form for processing beyond their originally intended use and in fact often hide (unintentionally) their information behind Java applet human computer interfaces.

One set of information available today that fits our needs is from the CalTrans loop detectors in San Diego. This information derives from approximately 338 sensors spread across the I5, I8, I15, SR52, SR56, SR94, SR163, and I805 freeways. Although the San Diego information is updated only once a minute on the web site, the same kind of CalTrans loop detector information for the Irvine area, with 780 sensors, indicates a one second update rate is achievable.

The San Diego information provides a separate .php page for each freeway with javascript code to plot the information on the user's screen. Although it is not presented in a nice XML format, the information is amenable to parsing and extracting the information of interest (whereas the Irvine area is buried behind a java applet), which we may then inject into our ISCA environment. The following shows an excerpt from the 5.php file where it is easy to see the commonality.

```
<div ID="graph" STYLE="position:absolute; top:70px;">
<table cellpadding=0 border=0 cellspacing=0 width=763>
<tr>
<td valign=bottom height=100>
<table bgcolor=green width=15 border=0><tr><td
onmouseover="dispInfoNB('N/O 163', '64 mph', '5 NORTH')" height=5>
</td></tr></table>
<img src="../images/pixel-dblue.gif" width=100% height=1></td>
<td valign=bottom height=100>
<table bgcolor=green width=15 border=0><tr><td
onmouseover="dispInfoNB('Hawthorn St', '62 mph', '5 NORTH')" height=5>
</td></tr></table>
<img src="../images/pixel-dblue.gif" width=100% height=1></td>
<td valign=bottom height=100>
<table bgcolor=green width=15 border=0><tr><td
onmouseover="dispInfoNB('India St', '65 mph', '5 NORTH')" height=5>
</td></tr></table>
```

In the above snippet, one can clearly see the script function `dispInfoNB()` with three parameters being the sensor identified by street, the speed, and the route and direction combined as a single parameter. The function name then becomes the primary search criteria and the parameters become our data of interest. A similar function name `dispInfoSB()` is used for sensors on the same route in the other direction. The display of information shows two horizontal bar charts and one could interpret "NB" as north bar and "SB" as south bar.

The various web pages for each freeway can then be parsed and the information converted into our <ISCAMsg> format for distribution. The natural decomposition to freeways makes it convenient to use a different publisher for each freeway. Through the parsing phase, we can even use a different publisher for each direction on the freeway or even break down the information to a single publisher representing a single loop detector in a way that might represent the future of information provisioning.

Our loop detector sensor proxy periodically (conceivably based on the html meta refresh tag of 60 seconds), grabs the page, parses it, and places the new information on Fulcrum.

The design of our proxy is two part. The main class accepts initialization parameters to identify the sensor information for which it is to proxy; it connects to a Fulcrum event-broker, and publishes advertisements of the information it will publish, and controls the rate at which it will invoke the "screen-scraping" event generation code. The screen-scraping code takes the initialization parameters from the main class, and upon command

    String eventData = speedEventGen.buildEventData() ;

generates the event data which is then placed on the output queue.

```
package ISCA.URLToEventGenerator.CalTrans ;

import java.io.* ;
import java.net.* ;
```

```
public class CalTransWebEventGen
{
  String urlStr ;
  String barId ;
  String sensorStr ;
  String routeStr ;
  String directionStr ;  // The sensor id = (name, route, direction)

  public String lastReport = "" ;

  String reportedSpeed = null ;

  public CalTransWebEventGen( String urlStr, String barId,
      String sensor, String route, String direction )
  {
    this.urlStr = urlStr ;
    this.barId = barId ;
    this.sensorStr = sensor ;
    this.routeStr = route ;
    this.directionStr = direction ;
  }

  String contents = null ;
  public void reload()
  {
    contents = "" ;

    try {
      BufferedReader in ;
      if  ( 0 == urlStr.indexOf( "file://" ) )
      {
      in = new BufferedReader( new FileReader( urlStr.substring( 7 ) )
) ;
      }
      else
      {
      URL url = new URL( urlStr ) ;
      in = new BufferedReader( new InputStreamReader( (InputStream)
url.getContent() ) ) ;
      }

// We could make only a single pass over the data, but instead are
// going to intentionally just grab all the data into a single string
      String line ;
      while( null != ( line = in.readLine() ) )
      contents += line ;
    }
    catch( Exception e ) { contents = null ; }
  }

  public String buildEventData()
  {
    reload() ;
    return( buildEventData( barId, sensorStr, sensorStr, routeStr,
directionStr ) ) ;
```

```
 }

 public String buildEventData(
   String barId, String searchSensorStr, String sensorStr, String
routeStr, String directionStr )
 {
  String sensorIdInfo =
    "<av name=\"route\" type=\"string\" value=\""
   + routeStr + "\"/>"
   + "<av name=\"direction\" type=\"string\" value=\""
   + directionStr + "\"/>"
   + "<av name=\"sensor\" type=\"string\" value=\""
   + sensorStr + "\"/>" ;

  String dateTimeEventInfo = null ;
  String speedEventInfo = null ;
  if  ( null != contents )
  {
   dateTimeEventInfo = scrapeTime( contents ) ;
   speedEventInfo = scrapeSpeed( contents, barId,
      searchSensorStr, routeStr, directionStr ) ;

   if  ( null == speedEventInfo )
    return( null ) ;
  }
  else return( null ) ;

  String eventStr = sensorIdInfo
   + (null == dateTimeEventInfo ? "" : dateTimeEventInfo)
   + speedEventInfo ;

  return( eventStr ) ;
 }

 public String scrapeTime( String contents )
 {
  String searchDateStr = "<td class=submenu>" ;
  String searchDateEndStr = "</td>" ;

  int datePosnStart = contents.indexOf( searchDateStr ) ;

  if  ( 0 < datePosnStart )
  {
   int datePosnFinal = contents.indexOf(
      searchDateEndStr, datePosnStart ) ;

   if  ( 0 < datePosnFinal )
   {
   String dateStr = contents.substring(datePosnStart,datePosnFinal);

   int timePosnStart = contents.indexOf( searchDateStr,
datePosnFinal ) ;
   if  ( 0 < datePosnStart )
   {
        int timePosnFinal = contents.indexOf( searchDateEndStr,
timePosnStart ) ;

        if  ( 0 < timePosnFinal )
        {
          String timeStr = contents.substring( timePosnStart,
timePosnFinal ) ;

          return( "<av name=\"date\" type=\"string\" value=\"" +
dateStr.substring( searchDateStr.length() ) + "\"/>"
              + "<av name=\"time\" type=\"string\" value=\"" +
timeStr.substring( searchDateStr.length() ) + "\"/>" ) ;
        }
      }

      return( "<av name=\"date\" type=\"string\" value=\"" + dateStr +
"\"/>" ) ;
      }
    }
    return( null ) ;
  }

  public String scrapeSpeed( String contents, String barId,
      String sensorStr, String routeStr, String directionStr )
  {
    String searchSpeedStr = "onmouseover=\"" + barId + "('" + sensorStr
+ "', '" ;
    String searchRouteStr = " mph', '" + routeStr + " " + directionStr
;
    int speedPosnStart = contents.indexOf( searchSpeedStr ) ;


    if  ( 0 < speedPosnStart )
    {
      int routeDirPosnStart = contents.indexOf( searchRouteStr,
speedPosnStart ) ;

      if  ( speedPosnStart <= routeDirPosnStart )
      {

      if  ( routeDirPosnStart < speedPosnStart +
searchSpeedStr.length() + 25 )
      {

      // Found it.  Now extract the speed...
      reportedSpeed = contents.substring( speedPosnStart +
searchSpeedStr.length(), routeDirPosnStart ) ;
      return( "<av name=\"speed\" type=\"long\" value=\"" +
reportedSpeed + "\"/>" ) ;
      }
      else // must be going the wrong direction, so keep looking
            // forward, could be the case of "no data" on the web page
      {
      speedPosnStart = contents.indexOf( searchSpeedStr,
routeDirPosnStart + 25 ) ;
```

```
      if  ( 0 < speedPosnStart )
      {
        routeDirPosnStart = contents.indexOf( searchRouteStr,
speedPosnStart ) ;

        if  ( 0 < routeDirPosnStart )
        {
// System.err.println( "(2) Scraped speed..." ) ;

          reportedSpeed = contents.substring( speedPosnStart +
searchSpeedStr.length(), routeDirPosnStart ) ;
          return( "<av name=\"speed\" type=\"long\" value=\"" +
reportedSpeed + "\"/>" ) ;
        }
      }
      }
    }
    }

    return( null ) ;
  }
}
```

7.0 Performance Evaluation

Our thesis hypothesis is that the increased expressiveness, efficiency, and scalability requirements for Internet-scale context-awareness can be achieved by extending CBPS without violating its valuable separation of concerns.

The expressiveness claim was validated in two parts in chapters 4 and 5. Chapter 4 detailed the application of open implementation techniques to allow the user to directly apply domain knowledge by writing executable code and to deploy the code by leveraging content addressability. Chapter 5 provided details on algorithm expressiveness.

The efficiency claim was partially addressed in Chapter 4 when we showed how to leverage content-addressability to avoid flooding advertisements or subscriptions through the network when setting up one-to-one conversations. In this chapter we provide further evidence of efficiency improvements by presenting proximity and traffic-route monitoring experiments to empirically evaluate the reduction in hop count. We further validate the efficiency claim by presenting an experiment that evaluates the performance improvements achieved through the use of our distributed memoization technique.

Increasing scalability is partially validated through the improved efficiency and partially through the distribution of complex event-relationship processing such that we avoid centralized processing bottlenecks. The core concepts of distributing the processing were discussed in Chapter 4, while Chapter 5 described the use of the observer / mediator pattern to increase complexity of supportable algorithms. In this chapter we use the experimental data from the traffic-route monitoring test case to validate the scalability of the infrastructure and of problem complexity.

7.1 Proximity Experiment

This section describes an experiment of Fulcrum supporting a federated architecture for ActiveCampus and compares those results to the basic and enhanced CBPS approaches. We perform this evaluation in two parts. First, how much suppression of original events does Fulcrum achieve? Second, what overall efficiencies are achieved?

ActiveCampus user positions are automatically generated via triangulation based on 802.11b signal strength and automatically reported to the system. On ActiveCampus's "buddy page", the user display shows all buddies with an indicator of *nearby* or *far*, mirroring the proximity relationship that we have used in our examples. Consequently, we took a week of data from ActiveCampus and "replayed" it through Fulcrum, using a conservative broker configuration.

7.1.1 Setup

The ActiveCampus test data is comprised as shown in Table 7-1. The basic range-ring algorithm, as described in Section 5.2, was used as the implementation strategy deployed to the edge-brokers.

Table 7-1 Active Campus Experiment Setup

Quantity	Description
643	users (anonymized for privacy)
2,165	buddy relationships
604,800	seconds = 168 hours = 1 week
3.5	average number of events per logged-in user per minute
360,067	location reports
5	event-brokers in a crossbar configuration

The overlay network was configured with 5 event-broker nodes in a crossbar configuration with one internal "routing" node as shown in Figure 7-1. Such a configuration might be used geographic coverage (e.g., one node per city plus the crossbar) or to mediate different commercial providers (e.g., AIM, MSN, ICQ, and Yahoo connected through a crossbar). Different configurations would change the number of hops that an event has to travel. This configuration has a maximum hop count of 2, minimizing the penalty for an event not being suppressed.

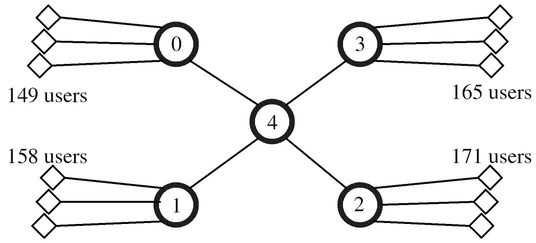


Figure 7-1 Crossbar Event-Broker Overlay Network

The 643 users were randomly distributed across the 4 edge brokers, resulting in 149, 158, 165, and 171 users at each respective node. Each user was configured as a publisher of its location as well as a subscriber to the proximity of each buddy.

This configuration resulted in the distribution of proximity subscriptions as shown in Table 7-2. Users attached to node 0 have proximity relationships with the number buddies on nodes 0 to 3 as 125, 155, 165, and 112, respectively. This resulted in 2,165 active subscriptions being deployed. Each was instantiated on two nodes, yielding 4330 instances.

Table 7-2 Proximity Relationship Distribution

Nodes	0	1	2	3
0	125	155	165	112
1	131	140	167	115
2	125	171	224	126
3	85	110	122	92

7.1.2 Results

Of the original 360,067 location reports, 11,955 were associated with users not participating in a buddy relationship, leaving 348,112 location reports that were subject to the proximity subscription.

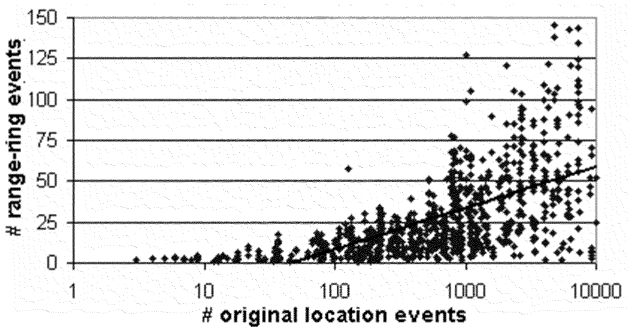


Figure 7-2 Range-Ring Events Generated Per Original Event.
A line is overlaid to show an expected lg2 movement.

Many users were not logged in to ActiveCampus at the same time as their buddies during the one-week period. Of the 2,165 active subscriptions, then, only 1,028 received data from both buddies. This means, effectively, that the 348,112 location reports were concentrated on half of the users. The range ring strategy resulted in 57,348 published range-ring events and 4,626 proximity relationship events (i.e., reports of a buddy moving into proximity) – 16.5% and 1.3% of the original event count, respectively, for a cumulative reduction of 82%. The breakdown of event reduction for each user's buddy is shown in Figure 7-2.

Comparing the number of original events to the number resulting from the active subscriptions yields an average event savings of 82% (a factor of 5.6). The average savings factor does not tell the whole story, however. Due to the centralized architecture of ActiveCampus at the time of data capture, user events were rate limited to 1 to 6 events per minute. In the wild, it is not unreasonable to expect 1-second (standard GPS reporting rate on naval ships) or better update rates – a 10-times increase. Because the number of range-ring events is based on distance, not the number of reports, the average event savings factor would then be 56. Also, user activity plays a significant role in event reduction as shown by a few sample data points displayed in Table 7-3.

Table 7-3 User activity affects event reduction

User Activity	# orig events	# range ring events
Stationary	22,657	6
Stationary	11,743	2
Mobile	1,008	127
Mobile	4,746	145

In many cases we find that users are relatively stationary. Their usage model is to go somewhere, turn their device on, remain in a confined space for a time, and finally to turn off their device and repeat the cycle. Thus, almost all events are suppressed at the source. Conversely, when people are in motion, there is usually much less suppression because buddies frequently exit their assigned range rings yet seldom walk directly toward each other.

7.1.3 Analysis Considerations

The analysis requires three perspectives. First, we consider a simplified, worst-case scenario where every relationship is independent – down to the sensor level. That is, no sharing of sensor data occurs. Second, we consider the effects of sharing sensor data across different relationships, as exemplified by multiple buddy relationships.

7.1.4 Analysis

Although we have achieved an 82% reduction in events, any efficiency gained must be evaluated from a systems perspective. In particular, the question is how much processing the whole system has to perform for all the events communicated.

Due to difficulties in running live Siena, Gryphon, and Solar CBPSs on the ActiveCampus data, we measured system efficiencies in terms of analytically accessible properties. The core costs are the processing of an event at an event-broker and any subsequent forwarding of the event through the broker network. We use the number of *hops* per original event, or *event hops*, as our common unit of measure to evaluate the efficacy of Fulcrum. We discuss the cost per hop in the next section.

To simplify the calculation of effectiveness we assume a uniform distribution of publishers, subscribers, and active subscriptions across all edge nodes. To verify that this simplification would not bias our results, we compared Fulcrum's empirical aggregate hop count based on the random distribution to the analytically derived one. The empirical aggregate count based on the configuration in the previous subsection is 94,334 hops. Analytically, it computes as 94,260, less than 0.1% smaller. We have confidence, then, that the analysis here is accurate.

Basic CBPS like Siena requires the subscribers to evaluate all relationships. Consequently, ¼ of the events coming into entry nodes and are immediately returned to subscribers attached to the same node, resulting in 1 event hop apiece (we only count processing and output, not the input). ¾ of the events will come into an entry node and get passed to the center node, on to one of the remaining three edge nodes, and finally to the subscriber, yielding 3 event hops. The subscriber itself evaluates the relationship, determining success 1.3% of the time and requiring 0 event hops. On average this is

$$\tfrac{1}{4} * 1 + \tfrac{3}{4} * (3 + 0.0133 * 0) = 2.5 \text{ event hops.}$$

This 2.5 event hops per original event computation uses the assumption that every relationship is independent. However, we know that CBPS derives its primary benefit through economies of scale, where sharing is part of the natural order. In our proximity experiment, each person has 3.37 buddies. Thus, for each event, 3.37 end-clients need to learn about the event and by our uniform distribution assumptions, 3.37 event brokers must receive the event. Thus, we almost always get one event going out to the hub where it then fans-out to reach the remaining subscribers. On average this is

$$\frac{1}{4} * 3.37 * 1 + (1-1/4^{3.37}) + \frac{3}{4} * 3.37 * (2 + 0.0133 * 0) = 6.89 \text{ event hops.}$$

But, these event hops service 3.37 relationships, so we have 2.04 hops per relationship.

When we consider the load factor on each node, we see that the 643 participants create an average of one event every 17 seconds. Thus, each edge-broker starts by processing 643 / 4 / 17 = 9.45 original events per second. Of these, (1-1/4$^{3.37}$) = 99% = 9.37 events get forwarded to the center node and are then evenly distributed to the edge-brokers at the other side of the network for 3.12 extra events apiece. Finally, the end-clients must perform an additional 3.37 relationship computations every 17 seconds for a total of 0.257 events per second. Consequently, each edge-broker must process 9.45 + 3.12 = 12.57 events per second while the center hub must process 4 * 9.37 = 37.4 events per second. We choose to assume event and relationship processing is equivalent that we may get a general sense of the overall load that can be compared with

Table 7-4 Basic CBPS - Proximity Load Factors

End-client	Edge-Broker	Central Hub
0.257 Events + 0.198 Computations	12.57 Events	37.4 Events
0.455 - Effort	12.57 - Effort	37.4 – Effort
Total system load = 643 * 0.455 + 4 * 12.57 + 37.4 = 380.24		

Although the number of events per second are small, we see a pattern that the event-brokers must perform exponentially more work at each layer deeper toward the center of the network that an event travels as was discussed in Chapter 3. This is largely due to the hierarchical nature of the network. A constant branching factor results in exponentially more nodes. Therefore, passing events in reverse, from the nodes toward the center, must result in exponentially more processing effort as the events combine toward the network core.

Aggregation-enhanced CBPS like Gryphon or Solar allows relationships to be computed at the first common node (1stCN). For the given broker configuration, ¼ of the events come into an entry node that also acts as a common node and the proximity relationship "success" event can be generated and sent to the subscriber in 1 event hop. ¾ of the events come in through their entry nodes and are passed to the center node that is consequently the first common node, yielding 1 event hop. Success events generated at the center node must be communicated with the subscriber for 2 more event hops. On average, this is

$$\tfrac{1}{4} * (0.0133 * 1) + \tfrac{3}{4} * (1 + 0.0133 * 2) = 0.773 \text{ event hops.}$$

This first computation is based on relationship independence. When we consider sharing, most events must pass to the network hub as they did with basic CBPS.

$$\tfrac{1}{4} * (0.0133 * 1) + (1-1/4^{3.37}) + \tfrac{3}{4} * (0.0133 * 2) = 1.014 \text{ event hops.}$$

But, these event hops service 3.37 relationships, so we have 0.3 hops per relationship.

The load factors follow a similar pattern as observed with basic CBPS. First, events are received and forwarded as discussed above. But now, the edge-brokers and the center node, in their roles as 1st CN, are required to perform additional computations, so that the end-clients do not. The average edge-broker must perform ¼ * 9.45 =2.36 computations. The central hub must compute multiple relationships for every event received calculated as: 37.4 * (¾ * 3.37) = 94.6. Finally, success events occur at a 1.3% rate and must get passed down, yielding an extra 0.12 events at the edge brokers and an extra 0.001 events at the end-clients.

Table 7-5 Aggregation Enhanced CBPS - Proximity Load Factors

End-client	Edge-Broker	Central Hub
0.06 Events	9.57 Events 2.36 Computations	37.4 Events 94.6 Computations
0.06 - Effort	11.93 - Effort	132 - Effort
Total system load = 643 * 0.06 + 4 * 11.93 + 132 = 218.3		

In the aggregation enhanced CBPS, events are reduced and relationship calculations eliminated at the end-clients. The edge-brokers also notice a slight reduction in the number of events to process. However, the tradeoff is that the central hub, as the primary 1st CN, takes on almost all responsibility for calculating the relationships.

Open-implementation Context-Aware Pub / Sub like Fulcrum's reduces original events into potentially significant events at the entry nodes. In this case, ¼ of the events come into an entry node that possesses both parts of an active subscription, yielding 1-hop success events. ¾ of the events come into an entry node hosting an active subscription that collaborates with a distant entry node. Empirical data from the previous subsection shows that 6.07 original events will become one collaboration event (range ring). These events travel through the center node and out to an associated active subscription. This yields 0.165 event hops for such events. Success events are then passed to the subscriber, which half the time will be local (1 hop) and the other half will be remote (3 hops) for an average of 2 hops. The net event hops per original event are

¼ * (0.0133*1) + ¾ (0.165 * 2 + 0.0133 * 2) = 0.27 event hops.

Active-subscription communications setup for each relationship adds an additional costs. ¼ of relationships reside on the same node for 0 external hops. ¾ of the relationships must create bi-directional paths at 2-hops each.

¼ * 0 + ¾ * (2 * 2) = 3 hops

This additional setup cost is non-recurring and amortize over time. For the 1028 relationships being evaluated, this results in an extra 3084 event hops for 0.00886 event-hops per original. The final Fulcrum cost is 0.0279 event-hops per original event.

This computation is based on relationship independence. However, as was detailed in Chapter 5, it did not make sense to share relationship events under most conditions of multiple proximity relationships. Hence, this 0.0279 is actually hops per relationship.

The load factors now shift toward the edge-brokers. As with the 1st CN approach, the end-clients only see original events and success events. But, the edge-brokers must each compute 3.37 relationships per event = 3.37 * 9.45 = 31.85 computations. From here, 0.165 * 9.45 = 1.56 collaboration events are generated and hence received by each edge-broker. These are then followed by 1.3% generation of success events of which ¾ are passed through the network ¾ * 0.0133 = 0.01. The central hub only sees the 1.56 collaboration events from each edge-broker for 4 * 1.56 = 6.24.

follow a similar pattern as observed with basic CBPS. First, events are received and forwarded as discussed above. The edge-brokers first receive the 9.45 events from the clients and then 0.165 *. These are then followed by

But now, the edge-brokers and the center node, in their roles as 1st CN, are required to perform additional computations, so that the

Table 7-6 Open-Implementation CAPS - Proximity Load Factors

End-client	Edge-Broker	Central Hub
0.06 Events	11.14 Events 31.85 Computations	6.24 Events
0.06 - Effort	42.99 - Effort	6.24 - Effort
Total system load = 643 * 0.06 + 4 * 42.99 + 6.24 = 216.78		

7.1.5 Discussion

We note that the total effort performed by the system is about 1.75 times greater for basic CBPS than for either aggregation enhanced or the open implementation approach. The most noticeable effects are where the work is performed. Basic CBPS requires the end-clients to be more directly involved at the cost of extra communications. It also leaves the central hub as the hot spot most susceptible to overloading as the network grows or as reporting rates increase. Aggregation enhanced CBPS eliminates extraneous communications with the end-clients but, even in this simple environment, places most of the processing burden on the central hub, leaving it susceptible to overloading as the network grows or as reporting rates increase. The open implementation approach also eliminates extraneous communications with the end-clients. The tradeoff is that the processing burden is placed on the edge brokers. If the network were to grow or the reporting rate increased, extra edge-brokers would be added to split the processing burden. Basic and aggregation enhanced CBPS have no simple mechanism to distribute processing load away from the central hub. As more end-clients are added, the central hub is directly affected.

When location sensor reporting is increased to 1 report per second, a 17-times increase, then basic CBPS will experience a 17-times increase in location events at 2.5 event hops apiece. Gryphon and Solar would experience a 17-times increase in location events sent to the 1st CN. However, this only reduces their event-hop count 0.751, because most of the effort lies in getting the original events to the 1st CN. In Fulcrum, because forwarded events are driven by user behavior, not event rate, the overlay network sees no increase in traffic, which yields a corresponding 17-times effective suppression, resulting in 0.0155 event hops per original event. Solar could achieve similar reductions by adding a transformation operator that rate-limits original location events, but would sacrifice the added accuracy of the increased reporting rate unless the rate-limiter made inferences over all the events.

7.2 Traffic Monitoring

This section describes an experiment of a "warning system" for traffic slowdowns and recoveries on preferred routes along San Diego freeways as shown in Figure 2-1 and Figure 2-2. Each route is composed of a series of CalTrans managed loop detectors (sensors) that report current traffic speeds. These speeds are converted to travel times that are then combined at a mediator to determine the route time. The San Diego region currently has 332 loop detectors (sensors) of which 323 are applicable. We did not consider HOV (high occupancy vehicle) lanes or non-reporting sensors.

For this experiment, each user's preferred route is monitored to determine when it is exceedingly slow or, after slowing, has resumed normal speed. Thus, a worker with a flexible schedule can choose to remain at work until the traffic on his specific route is considered tolerable. Each subscriber may have a different definition of "too slow." Without loss of generality, we choose to define slow as an increase in travel time greater than $\lg_2$(nominal route time). For example, a 16-minute travel time is considered slow when it exceeds 20-minutes; this could be a 16 mile freeway route where over 8 miles have speeds below 30 mph. We also note that although each user monitors a unique route, these routes overlap such that the same sensors are monitored for multiple users.

7.2.1 Setup

This test uses the traffic-route monitoring algorithm from Section 5.8.2 with the algorithm hierarchy shown as route1 in Figure 5-7.[59] The relationship manager – the observer / mediator at the top of the hierarchy – assigns to each sensor proxy the filter region. As the sensor data moves outside the bounding region, a collaboration event is published, the event is passed up to the manager, which then sends a filter update message back to the same proxy or may cause all proxies to update their filters. Each sensor proxy is provided with the ability to manage multiple filter regions – one for each relationship to which it belongs. The test data is comprised as shown in Table 7-7.

7.2.2 Results

This traffic-route monitoring experiment is substantially different from the proximity experiment for several reasons, which will affect our analysis. First, a route is a composition of a varying number of sensors. Second, there are a limited number of freeway sensors shared by many routes. Third, using a conservative overlay network, as

[59] The full hierarchy of Figure 5-7 shows the comparison of one route to another. The feasibility of such complex relationship hierarchies was tested with manually generated test data. Unfortunately, automatically generating competitive routes is beyond the scope of this research and hence not enough data can be generated on which to report repeatable results. However, we are still able to demonstrate the effectiveness of the algorithms in handling complex relationships using the single layer hierarchies represented by route1 and route2.

Table 7-7 Traffic-route Monitoring Experiment Setup

Quantity	Description
323	maximum loop-detectors (i.e., sensors)
2860	average maximum route
12.5	average maximum sensors per route
31555	average maximum observers
28.12	hours of data
1	average number of events per sensor per minute
545,041	sensor reports
10	network diameter – assume a hub-and-spoke model with uniform distribution
20	network branching factor

in the proximity example, is not reasonable considering the number of sensors and subscribers. Consequently, to evaluate event-hops, we need to normalize the number of events generated based on the size of the network and complexity of data relationships to effectively compare hop counts among the different CBPS styles. We then analyze the scalability achieved through the load balancing of our distributed processing by normalizing the hop count effort by the number of processing nodes. We build up these normalizations over a series of steps to clarify the considerations and decisions.

As before, we start by evaluating the number of event-hops per original event. However, we run into two complications. First, the size of the overlay network plays a significant part in the number of hops. For example, the conservative network for the proximity test, with a diameter of 4, cannot be directly compared with a metropolitan environment requiring a network diameter of 10. To eliminate this factor, we normalize by reporting event hops per original event *with respect to the diameter of the network* (i.e., traffic route monitoring divides the total event-hops by 10).[60]

[60] Although we attempt to normalize the reporting based on diameter, as the diameter shrinks, basic CBPS is better able to leverage economies of scale – it would use fewer event hops. But, to achieve the numbers of publishers and subscribers to support a city the size of San Diego, a network with a branching factor of 20 and a diameter of 10 is reasonable.

In the proximity example, it was reasonable to generate one hop count for each of the different CBPS methods, because all of the relationship structures were identical. That is, every pair-wise proximity relationship drew data from two sensors. However, each route may be composed of a different number of sensors and each sensor may have a different number subscribers, because road segments will be shared by different drivers. Consequently, the number of event-hops will vary. To capture this property, we define a new measure called *environment-complexity* that is the average number of sensors per route times the average number of listeners per sensor.[61] We now replace number of original events as the basis for the X-axis with *environment-complexity* per event.

With these external measures in place we now show the empirical results in Figure 7-3, zoomed-in in Figure 7-4, with the tabular data provided in Table 7-8. Each point on the graph represents the average of 30 test sequences, each with an average of 1314 routes and 13909 observers with 313 sensors.

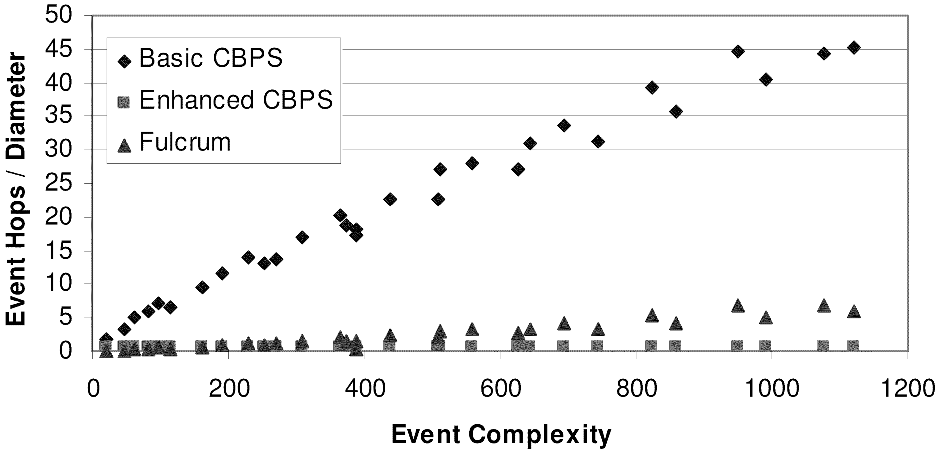


Figure 7-3 Traffic-Route Monitoring – Event Hops

[61] For the proximity example, there were two sensors per relationship an 3.37 observers per sensor, for an environment factor of 6.74.

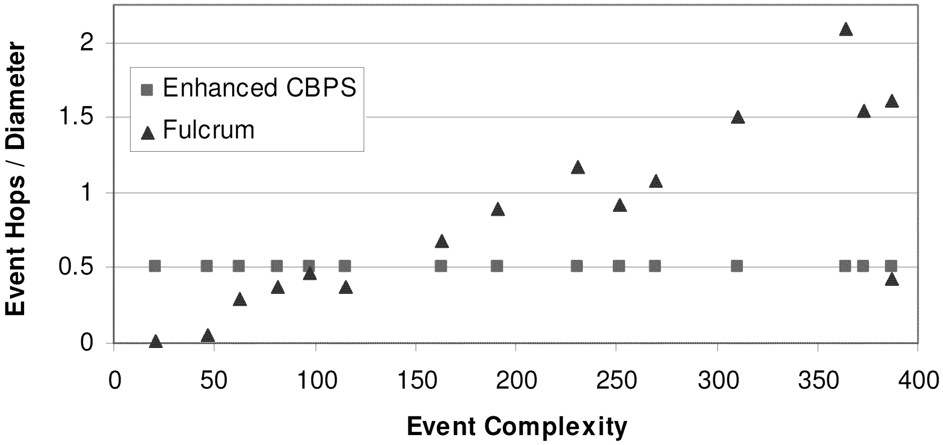


Figure 7-4 Traffic-Route Monitoring – Event Hops (zoomed-in)

For these experiments, the number of sensors per route has a minimum average of 7.1. With the branching factor of 20, there is only a $1/20^{7.1}$ chance of the 1st CN being other than the central node. This results in the constant 0.5 event hops / network diameter.

The test results for traffic routing show that increasing the environment-complexity significantly increases the event-hops used by a basic CBPS implementation. Although common (shared) subscriptions may be leveraged, a fan-out point is reached at the center of the network such that no further commonality exists and each event must travel to an individual destination. There are approximately 8 times more events passed using basic CBPS than using the Fulcrum technique. However, if the number of subscribers to each sensor report were to be further increased, the basic CBPS results would show a sub-linear growth. This re-enforces the observation that basic CBPS is best suited for large-scale broadcast of widely shared information.

The test results for traffic routing also show that an increase in event-hops in the Fulcrum environment. There is a cross-over between the number of event-hops for this

Table 7-8 Traffic-Route Monitoring Data

			Sensors	Observers		Event Hops / Diameter		
			per	per	Environment	Basic	Enhanced	
Sensors	Observers	Routes	Route	Sensor	Complexity	CBPS	CBPS	Fulcrum
68.93	159.60	17.33	9.21	2.32	21.32	1.66	0.50	0.008
291.97	1540.00	172.53	8.93	5.27	47.08	3.14	0.50	0.057
316.77	2818.40	396.93	7.10	8.90	63.18	4.95	0.50	0.291
321.47	3591.00	494.33	7.26	11.17	81.15	6.09	0.50	0.379
322.43	4256.00	578.47	7.36	13.20	97.11	7.10	0.50	0.467
322.20	3927.20	416.40	9.43	12.19	114.96	6.59	0.50	0.377
322.50	5736.40	625.47	9.17	17.79	163.13	9.39	0.50	0.674
323.00	7155.40	828.67	8.63	22.15	191.29	11.58	0.50	0.888
323.00	8634.93	1001.27	8.62	26.73	230.55	13.87	0.50	1.168
323.00	8080.80	802.13	10.07	25.02	252.03	13.01	0.50	0.914
323.00	8572.47	843.60	10.16	26.54	269.69	13.77	0.50	1.084
323.00	10658.07	1134.27	9.40	33.00	310.06	17.00	0.50	1.504
323.00	12821.60	1396.87	9.18	39.70	364.36	20.35	0.50	2.087
323.00	11840.40	1164.73	10.17	36.66	372.65	18.83	0.50	1.550
314.75	11672.00	1118.92	10.43	37.08	386.84	17.19	0.50	0.430
323.00	11455.40	1049.53	10.91	35.47	387.10	18.23	0.50	1.609
323.00	14323.33	1446.20	9.90	44.34	439.19	22.67	0.50	2.301
323.00	14370.87	1259.73	11.41	44.49	507.56	22.75	0.50	2.177
323.00	17200.20	1788.80	9.62	53.25	512.04	27.13	0.50	3.062
323.00	17805.67	1752.67	10.16	55.13	560.03	28.06	0.50	3.182
323.00	17177.87	1461.40	11.75	53.18	625.12	27.09	0.50	2.695
323.00	19591.62	1848.15	10.60	60.66	642.98	30.83	0.50	3.281
323.00	21465.47	2055.67	10.44	66.46	693.95	33.73	0.50	4.243
323.00	19955.33	1656.73	12.04	61.78	744.15	31.39	0.50	3.292
323.00	25069.80	2365.33	10.60	77.62	822.63	39.31	0.50	5.474
323.00	22798.40	1871.27	12.18	70.58	859.94	35.79	0.50	4.040
323.00	28467.47	2639.87	10.78	88.13	950.41	44.57	0.50	6.779
323.00	25753.73	2072.27	12.43	79.73	990.91	40.37	0.50	5.013
323.00	31555.00	2860.08	11.03	97.69	1077.84	44.46	0.50	6.907
323.00	28826.00	2297.27	12.55	89.24	1119.84	45.12	0.50	5.830

algorithm and an aggregation-enhanced CBPS system when the environment-complexity reaches the 100-150 region. There are three important considerations. First, Fulcrum is not intended to supplant aggregation-enhanced CBPS or even basic CBPS where their performance is better. However, with open implementation, a long-term goal is to have the system automatically meta-evaluate the subscription and determine which style of processing is best suited and adapt appropriately. Second, perhaps better algorithms could be written; if so, they could swapped-in without any change to the system. Third, the number of event-hops do not tell the scalability story because they do not consider the bottlenecks of centralized processing or the load balancing of distributed processing.

This last issue is the subject of our next set of evaluations. In the following figures we show both the bottleneck characteristic of aggregation enhanced CBPS as well as the load balancing characteristics of basic CBPS and Fulcrum. To evaluate the load balancing, we normalize the results by dividing the number of event-hops by the number of event-brokers (or end-clients) performing the relationship calculations, as shown in Figure 7-5, Figure 7-6, and Figure 7-7 with the tabular data provided in Table 7-9.

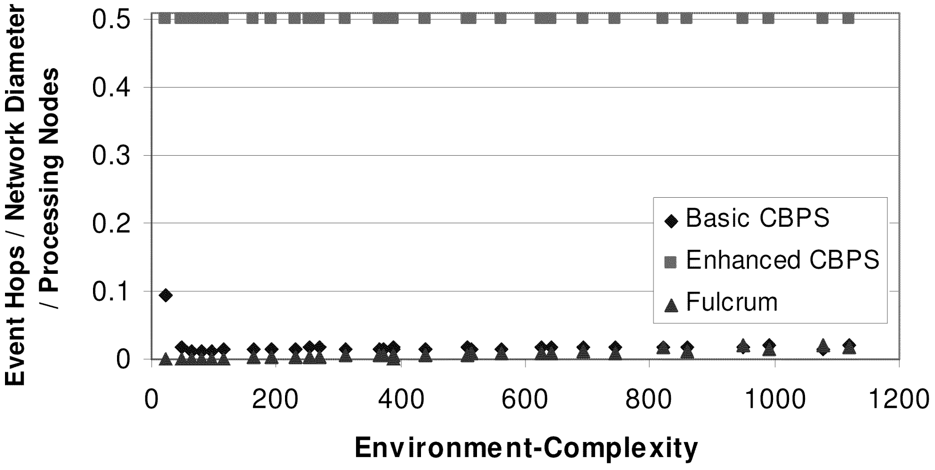


Figure 7-5 Traffic-Route Monitoring – Load Balancing

This normalization assumes that the costs of computing relationships are equal regardless of whether the processing occurs at the event-entry, 1st CN, or end-client. Although we expect the end-clients to be less capable and the costs effectively greater (e.g., because they are cell-phones), the equal-cost assumption is adequate and helps to provide a more accurate picture of comparative performance.

The tests show the value of distributed processing. As we observed earlier, all

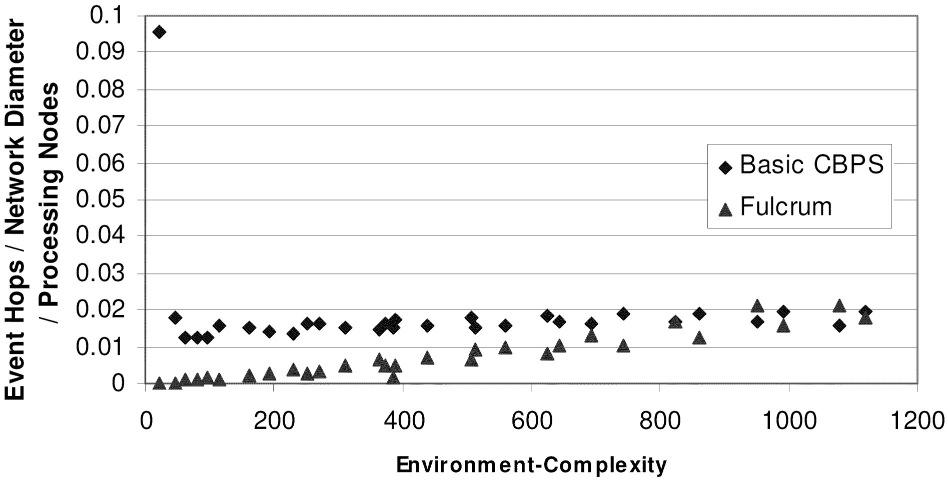


Figure 7-6 Traffic-Route Monitoring - Load Balancing (zoomed-in #1).

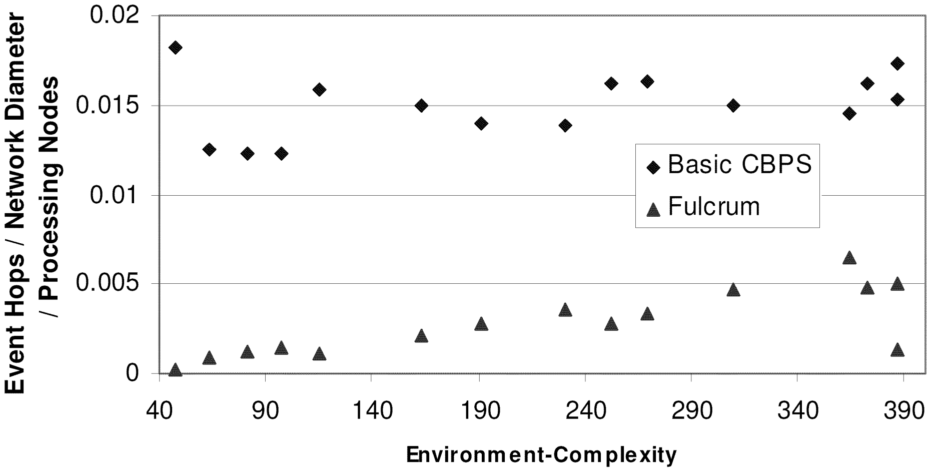


Figure 7-7 Traffic-Route Monitoring - Load Balancing (zoomed-in #2).

relationships have the central hub as their 1st CN. Thus, there is only a single node that performs all the relationship computations, resulting in a stable value of 0.5 event hops / network diameter / processing nodes. So, although this mechanism achieves the lowest event hop count, it does so at the cost of creating a bottleneck in the network core. Thus, each event causes the 1st CN a load factor proportional to the environment complexity. This centralizing nature of the aggregation-enhanced CBPS inhibits its scalability.

Table 7-9 Traffic-Route Monitoring Data – Load Balancing

	Event Hops / Diameter / Processing Nodes		
Environment	Basic	Enhanced	
Complexity	CBPS	CBPS	Fulcrum
21.32	0.0956	0.5000	0.00012
47.08	0.0182	0.5000	0.00020
63.18	0.0125	0.5000	0.00092
81.15	0.0123	0.5000	0.00118
97.11	0.0123	0.5000	0.00145
114.96	0.0158	0.5000	0.00117
163.13	0.0150	0.5000	0.00209
191.29	0.0140	0.5000	0.00275
230.55	0.0138	0.5000	0.00362
252.03	0.0162	0.5000	0.00283
269.69	0.0163	0.5000	0.00336
310.06	0.0150	0.5000	0.00466
364.36	0.0146	0.5000	0.00646
372.65	0.0162	0.5000	0.00480
386.84	0.0154	0.5000	0.00137
387.10	0.0174	0.5000	0.00498
439.19	0.0157	0.5000	0.00712
507.56	0.0181	0.5000	0.00674
512.04	0.0152	0.5000	0.00948
560.03	0.0160	0.5000	0.00985
625.12	0.0185	0.5000	0.00835
642.98	0.0167	0.5000	0.01016
693.95	0.0164	0.5000	0.01314
744.15	0.0189	0.5000	0.01019
822.63	0.0166	0.5000	0.01695
859.94	0.0191	0.5000	0.01251
950.41	0.0169	0.5000	0.02099
990.91	0.0195	0.5000	0.01552
1077.84	0.0155	0.5000	0.02138
1119.84	0.0196	0.5000	0.01805

Basic CBPS spreads out the relationship computations across the less powerful, destination clients. While each new route to be monitored increases the number of observers per sensor, it also introduces a new processing node. The results fluctuate around 0.016 event hops / diameter / processing nodes.

Fulcrum spreads out the relationship computations across the event-entry edge-nodes. The near-constant sensor quantity while the number of routes increases results in the slow growth, slope = 0.00002, and cross-over with basic CBPS in the environment-complexity region of 1000-1200. The performance ratio with respect to distributed processing of Fulcrum to basic CBPS is shown in Figure 7-8. It starts at 798:1 (not shown when environment-complexity = 21.3) and dwindles to about 1:1 as the environment-complexity reaches about 1000. On average, over the tested environment-complexity region, aggregation-enhanced CBPS performs 358 times worse, with respect to distributed processing.

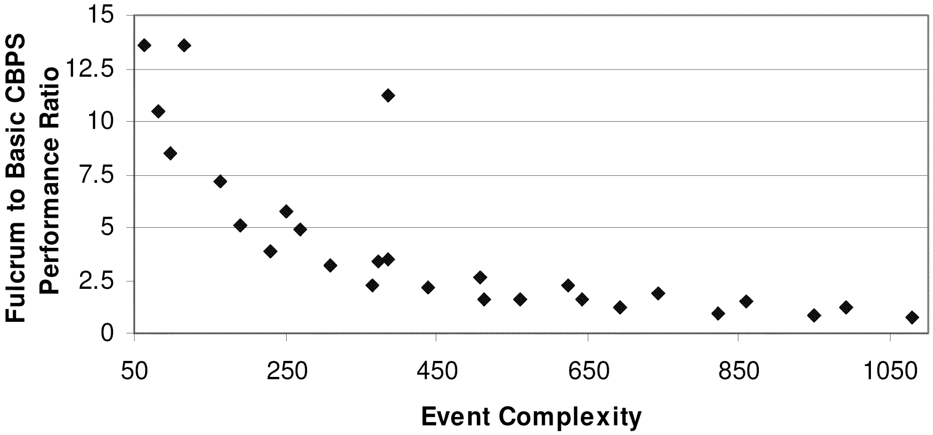


Figure 7-8 Fulcrum to Basic CBPS Distributed Processing Performance Ratio

7.3 Distributed Memoization

This section describes an experiment to evaluate the performance of the distributed memoization mechanisms, as described in Section 4.1.8, with respect to the fast forwarding algorithm [CW-2003] we have adopted as described in Section 6.11.4.

We know that the matching algorithm must iterate through all possible matches (P_{Ai}) for each attribute (A_i) in a notification (N) for a base cost of O(|N| * $|P_{Ai}|$). We know attribute lists may be found through a hash table lookup for a cost O(1). We know the values associated with each attribute are stored in a TreeMap such that the cost to find a potential match is O(lg $|A_i|$). This yields a total cost of O($|N|$ * ($|P_{Ai}|$ + lg($|A_i|$))).

Although the growth of a system may appear random, the world is not. There are a limited number of attributes – $|A_i|$ grows with the number of subscribers. Although the relationships of interest are different, the sensors being observed are of interest to multiple subscribers, often with similar values – $|P_{Ai}|$ also grows. As a CBPS-based system grows, we end up with lots of overlap. We generate the effects of a large system by creating overlapping data, such that many near-matches exist. This can be done with a collection of subscriptions (advertisements) where the first N attribute-constraints are identical for all subscriptions. We cause the subscriptions to differ only in the last attribute of the notification.

7.3.1 Setup

1. The overlay network was set up in a “ping/pong” configuration as follows:

Pub/Sub client Event-broker(1) Event-broker(2) Event-broker(3) Pub/Sub Client

2. Each pub/sub client, designated “ping” and “pong”, generated from 250 to 2000 near-identical advertisements and subscriptions as follows:

Id=A, Source=ping (or pong), Count =#

This advertisement setup means that we have complete overlap on the first two attributes (id and source). Consequently, any subscription with an id value of A will match all N advertisements on the id attribute. Similarly with the source field.

3. Notifications repeat the pattern, using the same set of values with the same matching on the first two attributes. 2000 notifications were used for each test.

4. To run the test, each pub/sub client creates a set of advertisements, then creates a set of subscriptions, and finally generates a series of notifications.

5. All pub/sub clients and event-brokers shared the same Sparc 10 running SunOS 5.8. Consequently, context-switching and other contention issues adversely affect the performance.

7.3.2 Results

Time-to-match metrics were recorded for each of: subscriptions matched in the regular way, subscriptions matched using the memoization technique, notifications matched in the regular way, and notifications matched using the memoization technique. The results are shown in Figure 7-9 and Figure 7-10 with tabular data in Table 7-10.

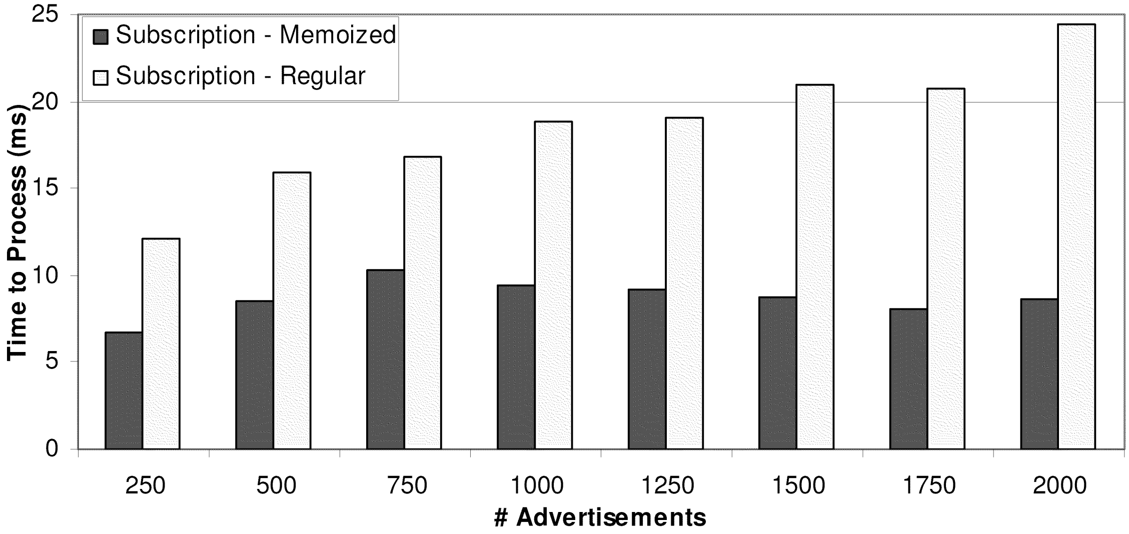


Figure 7-9 Distributed Memoization – Subscription Matching Performance.

These results show two behaviors. First, as a system becomes larger, the cost to match notifications to subscriptions and subscriptions to advertisements the regular way increases O($|P_{Ai}|$ * lg($|A_i|$)) as shown by the trend lines. At 2000 elements with two matching attributes each, we see just about a 2.75**:**1 ratio and growing for both subscriptions (24.45 **:** 8.62) and for notifications (17.09 **:** 6.31).

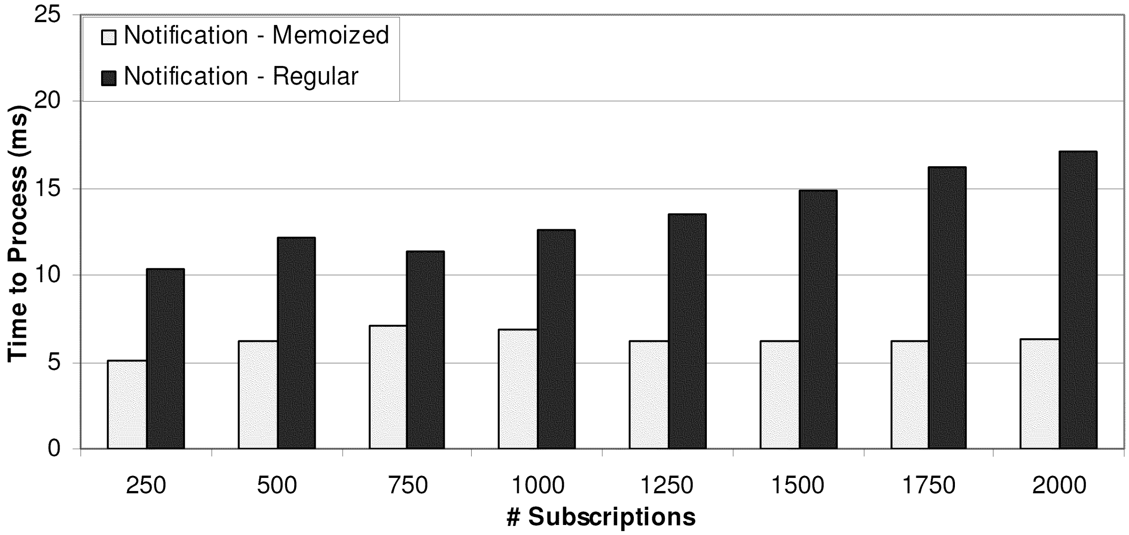


Figure 7-10 Distributed Memoization – Notification Matching Performance.

Table 7-10 Distributed Memoization Performance Data

	Subscription Memoized	Subscription Regular	Notification Memoized	Notification Regular
250	6.6712	12.0681	5.0213	10.3241
500	8.5405	15.9138	6.2142	12.1889
750	10.2872	16.8107	7.0954	11.4287
1000	9.4177	18.8297	6.8324	12.5978
1250	9.1486	19.1029	6.2126	13.4933
1500	8.7941	20.9224	6.1761	14.8638
1750	8.0798	20.7350	6.2330	16.2038
2000	8.6184	24.4546	6.3105	17.0936

We note that although all tests were run with only three attributes, approximately the same performance would occur if the number of potential matches were increased or decreased in inverse proportion to the number of attributes to be evaluated. We also note that the amount of work involved in matching, storing and forwarding a subscription are about 35% greater than matching and forwarding a notification.

Second, if we apply the distributed memoization technique, we can achieve an O(1) matching cost, making Fulcrum from 2 to 2.75 times faster with respect to the test data.

Constant time matching improves performance by reducing matching costs in the middle of the network and reduces the time for an event to travel from one side of the network to the other.

We note that the cost paid to perform regular event matching at the edge of the network is smaller because there will be fewer subscriptions and hence fewer attribute-values against which to match the information (i.e., only data specifically related to the edge-node's clients). Thus, we pay a low cost to find the cache identity and then get to reuse it where matching costs are normally higher. What savings do we really achieve?

The savings depend on the environment in which we deploy the technique. For example, if the distributed memoization were deployed into a CBPS environment where multiple consumers are possible for a given event, then we must account for the need to evaluate an event against every connection. We know that the memoization savings are only on a per connection basis. So, we first check all connections for the memoized value before going back to the more costly event matching. Thus, events are sped through the network using the express forwarding slips. As a result, the other connections, where we expect the matching to fail, perform this matching in parallel. We calculate the actual savings using Amdahl's Law.

Let us assume that only one connection matches with a 50% savings. By design, we know to never attempt matching for the connection over which the event was received. This leaves B-1 potential evaluations, where B is the branching factor. Then the overall savings is

$$1 - (0.5 / (B-1) + (B-2) / (B-1)) = 1 - (2(B-2) + 1) / 2(B-1) = 1 - (2B - 3) / (2B - 2)$$

To put this into context, if the branching factor were 11, the savings would be

$1 - (2B - 3) / (2B - 2) = 1 - 19/20 = 5\%$

These savings are inversely proportional with the branching factor.

However, if a pure ISCA environment were used, with a guarantee that only one consumer will be interested in any given event, then we can use a single lookup that covers all connections. In this case, we would achieve a savings of

$1 - 0.5 / B$

Using the same branching factor of 11, the savings would be

$1 - 0.5 / B = 1 - 0.5/11 = 95.45\%$

These savings grow with the branching factor.

7.4 Summary

The Fulcrum open implementation approach enables users to apply domain knowledge and evaluate context-aware relationships at entry nodes, while suppressing useless events. The (simple) pair-wise proximity test shows a reduction in the hop count from O(|events|) to an expectation of about 4ln(|events|), while the (complex) traffic-route monitoring test shows about an 8:1 reduction in the hop count with respect to basic CBPS. Efficiency improvements are also demonstrated in the distributed memoization testing which show an N-attribute notification can be effectively wrapped in a single-attribute express forwarding slip where O(1) event matching can be performed. With the test data, this yields from 2 to 2.75 times improvement over regular notification matching as CBPS-based systems such as ours grow. In a CBPS environment, this savings is reduced by the need to perform matching against each connection separately, yielding only a 5% savings when the branching factor is 11. However, in a pure ISCA environment, memoization achieves a 95% savings with the same branching factor.

Naturally the scalability of the system improves in proportion to the improved efficiency. Most importantly, the scalability of the Fulcrum approach is increased through load balancing that avoids bottlenecks that occur when the relationship computations are centralized. This is demonstrated in the evaluation of the traffic route monitoring test where the event hops / network diameter / number of processing nodes shows an average 358:1 improvement over aggregation-enhanced CBPS.

8.0 Discussion and Future Work

Achieving reductions in event traffic requires several tradeoffs. First, more of the computational burden is placed on the event-entry edge-brokers and consequently, a slight delay is experienced when semantically significant events are received. Usually, the event reduction benefit outweighs these costs. Fulcrum is more complex; it requires a programmer to write efficient distributed algorithm and make them available to the average user. When necessary, users can always use the basic CBPS approach. We believe we gain the best of both worlds by leveraging CBPS as a core infrastructure for the commonality while providing efficiency mechanisms for the individuality of context-awareness.

Yet, like basic CBPS, there are several outstanding issues which have yet to be addressed. In our case, we inherit several issues from CBPS itself and then add a few new concerns with our ability to introduce user code into the system. In the following sections, we discuss a variety of minor issues and then two significant concerns caused by not owning the entire problem. That is, the ability to perform actions exceeds the control of any single authority, including that of the event-brokers. Most notably, these include the problems of badly behaved clients, or in our case agents, and data looping problems.

There are three primary thrusts to future research directions. The first touches on productization and usability and includes topics like the development of implementation strategy repositories, an automatic expression parser and algorithm generator, development of quality user interfaces, and increasing deployment control. The second explores additional efficiency measures such as providing for meta-subscriptions, dynamically adjusting content-volatility, and query optimization techniques. The third considers issues associated with agents, including improved control and evolving the infrastructure into an agent framework.

8.1 Relationship Subscription Quantity

One concern is the large number of subscriptions that might be deployed as Java code. For our proximity experiment, there were 2,165 relationship subscriptions for 643 users with 3.37 average buddy requests. There are four components that mitigate this concern.

First, in the proximity experiment, there are 643 pub / sub clients and the overlay network can distribute the processing load across the event-entry edge-brokers. As the number of end-clients is increased, the overlay network can grow proportionately. In the traffic route monitoring experiment, the number of listeners is disproportionate to the number of sensors and hence event-entry edge-brokers. In this case, processor clustering may occur to spread the computational burden. However, this area requires more research.

Second, there is intrinsic sharing. The Java class loader only loads unique classes and the event-brokers only store and transmit a single copy of each class. Thus, only the first such subscription at a node results in transmitting code and configuring the class.

Third, open implementation allows the subscriber to achieve higher-levels of sharing by plugging in a more collaborative strategy. For example, a user could attach a strategy that tracks multiple buddies for a user, resulting in a sharing of range rings and event subscriptions behind the scenes. In fact, this was our approach for the traffic routing monitoring; only one sensor proxy existed per sensor and each proxy served as a monitor for all interested observers. To achieve this greater sharing requires a knowledge that such sharing will be desirable and coordination to make use of it.

8.2 Using a Different CBPS System

Another question is whether subscribers could, with an existing CBPS like Solar or Gryphon, write collaborative subscriptions like those realized through our open implementation approach. Indeed, for our examples, a subscriber could put out subscriptions for range ring events on each publisher. The subscriber itself would have to act as the clearinghouse that receives the events. Upon doing so, it would create new range ring subscriptions and retract the old ones. There are three problems.

For one, this added level of indirection would increase the hop count unless the subscriber was attached to the 1st CN. This is similar to our added costs when introducing a mediator.

Two, there is not a clear separation between the relational property of interest and its efficient implementation, increasing complexity and reducing the opportunities for reuse.

Three, there is a race condition, at least with this straightforward adaptation. Our algorithms avoid the race condition by updating the existing subscription on the other entry node, and before doing so checks that the other user has not already moved outside the new range ring. This kind of atomic check is not possible in normal CBPS when placing a new subscription; a subscription filter simply waits for the arrival of the next event.

8.3 Sharing the Processing Burden

Our Fulcrum approach places more computational burden on the event-entry edge-brokers. This seems appropriate, as these brokers only need to handle data directly associated with the local sensors and hence have little data they are required to process.

The alternative is to use the aggregation-enhanced CBPS approach. Under the assumptions of a hub-and-spoke model, constant branching factor (B), uniform distribution, and every-client-is-also-a-publisher, then, as was detailed in Chapter 7, each hop towards the network center results in the next event-broker being responsible for B times more events than the prior level, resulting in the 1st CN being responsible to process exponentially more events than the edges. This begins to show in the hot-spot and load-balancing concerns in our test cases. The test cases were too small to demonstrate the full magnitude of this issue.

8.4 Delays

Although hop counts are reduced by our approach, a slight delay is added at the edge-broker when the event is first received and passed to the implementation strategy agent. The agent evaluates the event with respect to the filter region before either discarding the event or passing semantically significant event data to associated algorithm components. Solar and Gryphon share this problem.

When the implementation strategy manages multiple relationships, as discussed for multiple buddies and as implemented for traffic monitoring, then efficient structuring of evaluations, (e.g., exploiting overlaps, like $X > 10$ is subsumed by $X > 5$) is again possible. This delay is acceptable for the savings in event traffic it provides.

Mitigating these delays is the distributed memoization. Once it is decided that an event needs to be pushed through the system, it is assigned an express forwarding slip and takes less time to traverse the internal nodes than regular notification matching.

If the delays are not acceptable to a client, that client could apply the basic CBPS paradigm. It should be noted however, that reducing the hop count through open implementation is scalable with respect to increasing sensor reporting rate, whereas in basic CBPS and aggregation-enhanced CBPS it is not.

8.5 Complexity and Reusable Components

Developing appropriate distributed algorithms is a more complex task than simply subscribing to all sensor data. We have begun to reduce the complexity of creating such algorithms through the creation of reusable components. Although these components make it easier to support new algorithms, a skilled programmer is still required. Future work, as discussed below, is needed to make Fulcrum capabilities broadly accessible.

8.6 Badly Behaved Clients

The normal behavior for a client in a CBPS environment is to make a connection, advertise or subscribe to event-types, and generate or receive notifications. A client that violates the basic behavioral pattern, say by having an infinite loop that generates slightly varying advertisements, or by creating a subscription that matches everything can effectively disable the middleware. We have not attempted to defend against this problem.

In accepting such risks, we have a precedent for equivalent risks inherent with deploying user code into the network. The Java code of an active subscription could be inefficient, buggy, or malicious. We have used the deployment wrapper, as described in

Section 6.12.8, to provide sandboxing by using a separate instance of a secure class loader to keep the agents separate. However, at this point, it is little more than a placeholder to support future work, as discussed below.

8.7 Duplicate Data

Duplicate data results when the same data enters or is repeated through the network multiple times. It is essentially "bad system behavior" as compared to "bad client behavior."

This may occur because there are multiple publishers of the same sensor data. For example, a satelite broadcasts its sensor data which is picked-up at multiple receivers which then independently connect and publish their data. Such redundancy is often intentional to ensure critical data is received. In some cases, the end-clients are designed to deal with duplicate data. This becomes difficult when the receivers first modify the data before publishing it to the common system.

Also, duplicate data may occur if a client republishes data it has received, say amended with additional content. Sometimes it is enough to change the attribute names, say from "X" to "myX" and to modify the end-clients accordingly. However, in our global environment we are not likely to have ownership control over these end-clients and so could not change them to look for a modified attribute name.

8.8 Unintended Feedback Loops

Unintended feedback loops (aka data looping aka data ringing) are similar. Instead of re-publishing the same data, two (or more) different clients feed off of each other's data, creating a cycle. The worst case situation occurs when no stable state is achieved and an endless loop ensues.

Normally, these kinds of looping condition are difficult to handle even after they have been detected. This is primarily due to not controlling all the pieces and not being allowed to break anything that is currently working. With the introduction of message context and the message context filter, we now have enough information to control receipt of messages to those that have the provenance information of a specific provider – providing of course we can alter the end-client to look for this contextual data.

Data looping conditions are often difficult to detect and track down when humans are directly observing system behaviors. With deployed code, it will be more difficult to detect such situations.

8.9 Why Cooperate

Another question remains why the infrastructure should run code for a subscriber? First, We have shown that for context-aware applications, the naïve way to request information for use in context-aware relationships can be highly inefficient and is not scalable. However, the demand for context-aware services is increasing. Therefore, to achieve some of the most basic scalability requirements of any such system, these kinds of techniques are required.

Second, we know that people are clever and constantly devise new kinds of relationships that are useful to them. The ability to build enough capability into the infrastructure to satisfy most users would be a continuing challenge. Yet, the competitive advantage in the quest for market share is often defined by providing more services at lower costs. In providing the equivalent of "self-service" to the subscribers, this burden can be reduced.

However, in either case, the security and agent control issues must be solved. As discussed below, code certification techniques can be used to deal with such issues.

8.10 Future Research Directions

There are three primary thrusts to future research directions that are reasonably unique to Fulcrum. The first addresses productization and usability. The second explores additional efficiency measures. The third considers issues associated with agent behaviors. Issues like security and peer-to-peer networking we leave to the experts in other fields, for now.

8.10.1 Implementation Strategy Repositories

The open implementation paradigm identifies four layers of interface as described in Section 4.1.3.1. We currently have insufficient experience to have developed a library of common algorithms that a user could freely reference at layer 3 nor to describe at layer 2. However, we recognize a commonality among the implementation strategies we have developed that would allow us to build up a useful parameterized set of algorithms. Future work should expand the infrastructure to provide a repository of algorithms and a 'language' appropriate to declaratively specifying characteristics of the environment such that a user could interact with the system using the different layers.

8.10.2 Automatic Expression Parser and Algorithm Generator

Many relationships may be expressed as $A + B + c < D + E + F$, where c is a constant and A, B, D, E, and F are independent variables. For example, the comparison of two traffic-routes uses this kind of relationship hierarchy, as shown in Figure 5-7. More generally, to find the best of N routes, we might write minimum((A+B), (D+E+F), (G+H+I+J), …, threshold = 2 minutes). It would be useful therefore to provide a language and an expression parser that dynamically generates the implementation strategy, based on the repository of implementation strategies described above.

Each independent variable would need to uniquely identify the sensor source of the data. Such a parser would allow users to avoid the messy details of the implementation strategies. It would, in effect, take the user back to open implementation interface layer 1, and in so doing increase the usability of the system. However, the more complex the relationship (i.e., equation), the greater the need for domain-specific knowledge. Therefore, it would be useful to provide some mechanism to incorporate the content-volatility into the expression (e.g., language annotation similar to a coefficient), in effect, bringing us open implementation's layer 2 interface.

It is easy to envision a parser that deals solely with the composition of independent variables. Although the simple composition expression parser described in the prior paragraph would be useful, we need a parser capable of detecting the patterns in higher-order equations in order to reduce them to independent filters. For example, the proximity equation, "$(A.x – B.x)^2 + (A.y – B.y)^2 < c$" does not have an straight-forward reduction because we compare attributes between different events. It would be useful to automatically create the same kind of range-ring solutions automatically as we did by hand because we knew the properties of Cartesian distance.

8.10.3 User Interface

The last two enhancements have focused on providing ways to make the system easier to use. However, even with these enhancements, Fulcrum is still completely middleware with only network interface and API definitions. It requires all users be programmers. To achieve broad acceptance requires a user interface simple enough for an average consumer to use. Such research falls in the Human Computer Interface (HCI) realm. The design of the user interface will naturally depend on the form-factor. A cell-phone based interface will be distinctly different from a desktop interface where an omnipresent ‘dashboard’ experience might be presented. The significant challenge will be developing a general purpose interface that may be extended easily, say with a ‘wizard’, to incorporate the various domain-specific differences. The user interface should, as a minimum, support input and control that map to the capabilities of the proposed expression parser.

8.10.4 Increased Deployment Control

Presently, the deployment slip is intended to reach the event-broker of a specific publisher. It would increase the overall flexibility of the system to provide a mechanism whereby the deployment slip represents a composition of event-types such that the 1st CN could be selected as the deployment destination. We would define a *composition deployment slip* that would contain multiple deployment slips as a conjunction. It would then traverse the network up to the point where all the deployment slips are matched on the same connection. When they can no longer be matched on the same connection, then the current event-broker must be the 1st CN. However, as demonstrated in Chapter 7, this could increase bottlenecks within the network core.

It might also be useful to provide support for user code that could be instantiated and then directly query the event-broker's database and performance metrics to determine whether or not algorithmic code should be instantiated at the given event-broker.

A third possibility is to allow the deployment slip to be treated as a hint, instead of as a directive. The middleware would need to possess the logic to determine how best to handle the active subscriptions. Then, this would behave as an open implementation layer 2 interface. However, if the active subscriptions are not instantiated where expected, then it might create havoc with the routing slips used to set-up one-to-one conversations between the distributed algorithm components.

8.10.5 Meta-Subscriptions

One piece of expressiveness that is still missing is meta-subscriptions. For example, there is no ability to subscribe to subscriptions. A 'smart' sensor might conserve battery power by not reporting information unless there were a listener. However, currently it has no way of knowing if such a listener exists. The sensor needs to subscribe to subscriptions to its data. The same sensor would also need to know when the subscription is deleted.

In anticipation of such capabilities, we have provided the payload concept. Without it, reporting the satisfaction of the subscription to a subscription would look like a subscription.

8.10.6 Dynamically Adjusting Content-Volatility Factor

In Section 5.4, we described content-volatility as a form of metadata that allows us to adjust bounding regions based on knowledge about the data sources. Although this is part of efficiently handling dynamic data, the ability to specify the content-volatility is limited to static input or subject to control signals from collaborative processes – just as we might handle currency exchange fluctuations. However, this requires cooperative processes.

It might be useful to allow content-volatility information to be automatically adjusted based on current observations. For example, each time sensor data moves outside its bounding region, the observer / mediator component of the algorithm could increase the content-volatility factor associated with that sensor, say by 10%, while reducing associated partners proportionately.

8.10.7 Dynamically Adjusting Content Matching Order

Our distributed memoization shows the value in matching fewer attributes. Once a match is successful and the forwarding decision made, then no further matching for the given connection is attempted. Because CBPS is, in essence, a streaming database, we might improve performance by developing the CBPS analogy of appropriate query optimization techniques. For example, we might take a message bundle and first attempt to match subscriptions with fewer attributes. Or, we might first check the ones with high frequency fields.

To match a notification, the forwarding algorithm we are using takes each attribute, in the order specified, and cuts across all subscriptions with that attribute. We might be able to design a fitness function for matching costs and minimize the average costs. For example, we might be able to assign a cost to each attribute based on its frequency, with higher frequencies resulting in higher costs. We could then adjust the cost by the number of attributes in the subscription, where fewer attributes to match cost less. Then, when a notification comes in, we could pre-sort the attribute matching order based according to cost. Some historical weighting based on notifications received might further adjust the weights. The cost values would be unique to each event-broker.

8.10.8 Improved Control Over Agent Behavior

We have acknowledged there are a variety of potential issues with respect to badly behaved clients and agents. The resources available to “greedy” subscriptions can be reduced by sandboxing the computation in a separate thread and using mechanisms akin to Jabber’s *karma* [JSF-2004].

Separately, security research is ongoing to [FZBKM-2004, Khurana-2005].

More fine grained threading controls would be useful. As one possible defense, it is conceivable that writing active subscriptions be remanded to expert programmers who must certify their code in the same ways that a programmer for a cell phone application must get the program certified before it can be placed into general usage.

8.10.9 Agent Framework

Publish / subscribe has been considered as a potential communications genre for mobile agents [PLZ-2003]. Through our use of open implementation techniques, we have created the possibility of using Fulcrum itself as an agent framework. When an agent wants to move, it identifies the appropriate location and submits an active subscription, sending itself and a serialized representation of its core information to some other location to perform its work. It may require little additional work, other than writing the appropriate active subscriptions, to completely develop an agent framework.

9.0 Conclusions and Contributions

The commoditization of networked sensors is fueling the emergence of internet-based context-aware applications. Relationships tend to be highly personalized, consist of dynamic data from multiple information publishers, and result in few composite relationships satisfying end-user criteria proportionally to the raw data rate.

Content-based publish / subscribe systems are a natural substrate for supporting context-aware application development because they provide for separation of publishers and subscribers, efficient event distribution, extensibility, and scalability. However, Internet-scale context-aware computing requires that attribute-to-attribute relationships be evaluated across event boundaries, which CBPS does not support. This results in a subscriber being forced to request all sensor data. Pushing all the raw sensor events across the network is neither efficient nor scalable; it burdens the network and leads to device saturation. We must reduce the number of events that pass through the system; complex processing must be performed within the network.

Recent advances in aggregation-enhanced CBPS permit subscribing to these relationships, but the relationships are computed at intervening nodes in the CBPS middleware, resulting in centralizing the computations and creating a system bottleneck. Processing data at the 1st CN, near the center of the network, is non-scalable; we need to push the processing out to the sensor-edges of the network and perform distributed computation.

The underlying problem is that the separation of concerns afforded by the middleware precludes publishers' event-brokers from collaborating with subscribers and each other to implement algorithmically efficient application-specific context-aware inferences. Despite the apparent shortcomings of CBPS, the increased expressiveness, efficiency, and scalability requirements for ISCA can be achieved by extending CBPS. We do this through a combination of three techniques.

First, subscribers are permitted to share their domain knowledge through the use of open implementation techniques to dramatically reduce the number of events processed and forwarded through the middleware. Open implementation provides for separation of subscriptions and implementation strategies, allowing for separate, modular development and reuse of relationship subscriptions and implementation strategies.

Fulcrum supports open-implementation context-aware publish / subscribe. Fulcrum employs *active subscriptions* in the form of Java applets. They are first-class entities that both subscribe to events and publish new events, enabling event-based collaboration amongst copies of the strategy deployed where raw sensor events enter the system.

Second, by using a distributed algorithm idiom based on the "law of continuity" and the use of the observer / mediator pattern, these implementation strategies provide efficient filtering of complex relationships. Each distributed algorithm component acts autonomously; it sets up a filter region, monitors sensor data, and discards all events within the identified region. When the sensor data falls outside the filter region, the components communicate with each other to set up new filter regions. By following separation of concerns principles and using the observer / mediator pattern, we are able to extend our earlier implementation strategies to support a wider class of problems, to support larger numbers of publishers within a relationship, and to support complex relationships.

Third, we leverage the content-based routing mechanisms at three levels. First, *deployment slips* allow the subscriber to specify destinations where the implementation strategy components should be deployed. Thus, we are routing content-processing as well as content. Second, *routing slips* enable the implementation strategy components to efficiently set up one-to-one conversations without burdening the network with a flood of advertisements or subscriptions. Third, *express forwarding slips* allow the middleware to convert a costly N-attribute notification into a single-attribute notification with an O(1) matching cost. Using this content-based approach to address the performance problems, we preserve and leverage the CBPS ideas without exposing network details.

We evaluated the Fulcrum approach with both trivial and more complex context-relationships to show how we can create and deploy a hierarchy of filters that leverage the communications properties of the CBPS infrastructure to create bi-directional data and control flows. We demonstrated the efficiency characteristics in terms of reduction in event hops and reduced matching costs. We also analyzed the value of distributing the relationship computations across multiple event-brokers instead of centralizing the computations as done in aggregation enhanced CBPS systems.

The pair-wise proximity test shows a reduction in the hop count from O(|events|) to an expectation of about 4ln(|events|). This shows an 82% reduction over basic CBPS and 64% with respect to aggregation enhanced CBPS. The traffic-route monitoring test shows about an 8:1 reduction in the hop count with respect to basic CBPS and a cross-over in the performance with aggregation enhanced CBPS, depending on the complexity of the relationship and the number of relationships to evaluate. The most significant observation is that the Fulcrum approach scales with technology improvements that lead to increased sensor reporting rates because Fulcrum performs independent filtering at the event-entry edges of the network based on actual data movement whereas other CBPS forms require all events to be propagated.

Efficiency improvements are also demonstrated in the distributed memoization testing which show how an N-attribute notification can be effectively wrapped in a single-attribute express forwarding slip where O(1) event matching can be performed. With the test data, this yields from 2 to 2.75 times improvement over regular notification matching on one of B connections to yield an overall savings of about 5% when applied to a CBPS environment and over 95% if applied to a true ISCA environment.

These efficiency improvements naturally enhance the scalability of the system. The key measure of Fulcrum's scalability occurs when we examine the load balancing that avoids bottlenecks that occur when the relationship computations are centralized. This is most clearly demonstrated in the evaluation of the traffic route monitoring test where the event hops / network diameter / number of processing nodes shows an average 358:1 performance improvement over aggregation-enhanced CBPS.

By combining open implementation, distributed processing, content-addressability, and distributed memoization, we are able to satisfy the required increases in expressiveness, efficiency, and scalability necessary to achieve our Internet-scale context-awareness vision.

9.1 Thesis Contributions

This thesis makes seven contributions as follows.

9.1.1 Internet-Scale Context Awareness as Emerging Challenge

In Chapter 2, it identifies an emerging problem and defines an idea called Internet-scale context-awareness. Our problem starts with the proliferation of networked devices world-wide producing continuous streams of data at high data rates. Easily half of these billion plus devices are also information consumers that each want to be informed when data relationships of interest occur. However, the relationships are highly individualized and are derived from high frequency dynamic data from multiple publishers yet yield low frequency composite events which may be complex. A lifecycle model for perishable data is created.

9.1.2 Open Implementation Approach to CBPS

In Chapter 4, it introduces open implementation mechanisms into the CBPS environment to avoid the information glut and device saturation that would normally ensue in trying to detect such data relationships. The "user" then is allowed to inject domain-specific knowledge into the network in the form of first-class publish / subscribe agents, as Java applets. The user-provided agents are fulfilling the open implementation interface layer 4, where mechanisms are provided for the user to provide procedural code satisfying a well defined interface. These agents are distributed algorithms that collaborate to eliminate useless event traffic at the sensor-edges of the network. The deployment of the agents are performed declaratively using content-addressability. We are effectively programming the network.

9.1.3 Generalization of Content-Addressability

In Chapter 4, in leveraging CBPS, it demonstrates the power of content-addressability in three new ways. First, it introduces a *deployment slip* to allow the user to control where to deploy code into the network and enables the creation of complex information hierarchies. Second, it introduces a *routing slip* to deploy advertisements, subscriptions or notifications into the network, primarily for the purpose of creating one-to-one conversations without subjecting the network to a flood of advertisements or subscriptions. Third, it creates a distributed memoization mechanism and introduces a *express forwarding slip* to reduce multi-attribute event matching to single attribute event matching and hence more rapidly routes data through the network. This changes the overlay network from a collection of independent event-brokers into a collaborative ecology.

9.1.4 "Continuity-Based" Style of Context-Aware Distributed Algorithms

In Chapter 5, it develops a style of distributed algorithms and a small collection of related techniques that are effective in minimizing event traffic. The algorithms leverage the "law of continuity" to define bounding regions around sensor data, where relationship satisfaction is impossible, such that new sensor report may be filtered independently from other sensors. Efficiency techniques include the application of *content-volatility* contextual information, creation of *relationship satisfaction regions*, and buffering based on the direction from which event data approaches a target midpoint.

Additionally, it incorporates the observer / mediator pattern into the distributed algorithms to enable a hierarchy of complex behaviors. This is facilitated by two key features used to create a critical buffering service. First, mediators manage individual conversations with each of the components under their management. Second, they manage a buffer region. Thus, a critical update of one set of sensor data need not affect other managed sensors.

9.1.5 Develops Six Distributed Algorithms

It introduced six distributed algorithms to efficiently determine context-aware relationships. Two algorithms are created for pair-wise proximity, two algorithms for group proximity, one algorithm is developed for traffic-route performance monitoring and another algorithm to dynamically notify a user when a selected traffic-route is a better choice.

9.1.6 Incorporates Explicit Contextual Filtering

It creates a mechanism for attaching system context to a message and provides users with explicit controls to limit propagation or receipt of messages based on contextual filters. As one example, a deployed agent may be designed to receive data only from immediately adjacent publishers.

9.1.7 Fulcrum - A Reference Implementation for Context-Aware Publish / Subscribe

In Chapter 6, it develops a reference implementation called Fulcrum. In Chapter 7, it provides an experimental evaluation of the system – more specifically of the algorithms created. These performance metrics covering both efficiency and scalability improvements are summarized in the conclusion.

10.0 References

[AAHLKP-1997] Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M., Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks* **3**(5): pp. 421-433. October 1997, [Online] Available: http://portal.acm.org/citation.cfm?id=272199 [2/15/05]

[ACMTN-2004] Emotional Computing, ACM Technews, Online Available: http://www.acm.org/technews/articles/2004-6/1112f.html#item5 [11/2004]

[AFP-2004] Sept. 2, 2004, Mobile phone sales surge at record pace in second quarter: Gartner, [Online] Available: http://story.news.yahoo.com/news?tmpl=story2&u=/afp/20040902/tc_afp/world_telecom_mobile [9/3/4]

[AHRI-2004] 2004, Aware Home Research Initiative [Online]. Available: http://www.cc.gatech.edu/fce/ahri/projects/

[AOL-2004] 2004, America On-Line [Online]. Available: http://www.aol.com/ [8/10/4]

[aura-2002] Project Aura, Carnegie Mellon University, 12/17/2002, [Online]. Available: http://www-2.cs.cmu.edu/~aura/ [11/13/4]

[BBC-2004] BBC News, A 'prison without bars', 7/19/04 [Online]. Available: http://news.bbc.co.uk/1/hi/uk/3906625.stm [8/1/04]

[Bellis-2004] Bellis, Mary. 2004. Johannes Gutenberg – Printing Press [Online]. Available: http://inventors.about.com/library/inventors/blJohannesGutenberg.htm [7/3/4]

[Berger-2004] 2004 Berger, Sandy, Internet Information Doubles, [Online] Available: http://www.compukiss.com/populartopics/research_infohtm/article892.htm [8/10/4]

[BG-2002] Burrell, J., Gay, G., E-graffiti: Evaluating Real-World Use of a Context-Aware System. Interacting with Computers 14(4) pp 301-312, 2002.

[BG-2005] Boyer, Robert T. , Griswold, William G., Fulcrum: An Open Implementation Approach to Context-Aware Publish / Subscribre, Proceedings 38th Hawaii International Conference on System Sciences, January, 2005.

[Botts-2003] Botts, Mike, A Sensor Model Language: Moving Sensor Data onto the Internet, Magazine, April, 2003, [Online]. Available: http://www.sensorsmag.com/articles/0403/30/main.shtml [8/29/4]

[Botts-2003a] Botts, Mike, Sensor Web Enablement, Open GIS Consortium, May 12, 2003

[Botts-2003b] Botts, Mike, Sensor Model Language (SensorML): XML-Based Language for In-situ and Remote Sensors, Open GIS Consortium, NIST Workshop on Data Exchange Standards at the Jobsite, May 29, 2003

[BP-2000] Bahl, Paramvir, Padmanabhan, Venkata N., RADAR, An In-Building RF-based User Location and Tracking System, IEEE INFOCOM 2000, 2000

[caag-2004] Megan's Law [Online]. Available: http://caag.state.ca.us/megan [8/15/4]

[CABB-2004] Cilia, M., Antollini, M., Bornhovd, C., Buchmann, A., Dealing with Heterogeneous Data in Pub/Sub Systems: The Concept-Based Approach, Proceedings Distributed Event Based Systems, pp 26-31, 2004

[CalTrans-2003] Caltrans, State Highway Congestion Monitoring Program Annual Report, 2002 HICOMP Report, November 2003, pg 2-11.

[Caporuscio-2002] Caporuscio, M., Mobility Support in the SIENA Publish / Subscribe Middleware, [Online]. Available: http://www-serl.cs.colorado.edu/downloads/serl-talks/2002.03.12-Mobility_and_Siena.ppt [4/25/5]

[Carzaniga-1998] Carzaniga, A., Architectures for an Event Notification Service Scalable to Wide-area Networks, PhD. Thesis, Computer Science, University of Colorado, Boulder, 1998

[CCR-2003] Chen, X., Chen, Y., Rao, F., An efficient spatial publish/subscribe system for intelligent location-based services, *Proceedings of the 2nd international workshop on Distributed event-based systems*, 2003

[CCW-2003] Caporuscio, M., Carzaniga, A., Wolf, A., Design and Evaluation of a Support Service for Mobile, Wireless Publish / Subscribe Applications, University Colorado Technical Report CU-CS-944-03, January, 2003

[CDF-2001] Cugola, G., Di Nitto, E., Fuggetta, A., The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Transactions on Software Engineering, 27(9), 2001.

[CD-2003] Unknown, Fingers do the talking, China Daily, December 29, 2003, [Online] Available: http://www.chinadaily.com.cn/en/doc/2003-12/29/content_294080.htm [9/11/4]

[CFHZB-PAST-2003] Cilla, M., et al, Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware, Proceedings Distributed Event Based Systems '03, ACM, 2003

[CiteSeer-2004] 2004, Computer and Information Science Papers CiteSeer Publications ResearchIndex [Online]. Available: http://citeseer.ist.psu.edu/ [8/10/2004]

[CK-2000] Chen, Guanling and Kotz, David, "*A Survey of Context-Aware Mobile Computing Research*", Department of Computer Science Dartmouth College, Technical Report TR2000-381, 2000

[CK-Solar-2002a] Chen, Guanling and Kotz, David, SOLAR: Toward a Flexible and Scalable Data-Fusion Infrastructure for Ubiquitous Computing, Dartmouth Tech Report, ?2002 [also UbiComp 2001 – UbiTools Workshop 2001]

[CK-Solar-2002b] Chen, Guanling and Kotz, David, SOLAR: A Pervasive-Computing Infrastructure for Context-Aware Mobile Applications, Computing, Dartmouth Computer Science Technical Report TR2002-421, February 28, 2002

[Collins-2003] Collins, Jonathan, The Cost of Wal-Mart's RFID Edict, RFID Journal, 10/23/3, [Online]. Available: http://216.121.131.129/article/articleprint/572/-1/1 [12/22/4]

[Cooltown-2004] 2004, Cooltown / Making cooltown real, HP [Online]. Available: http://www.cooltown.com/cooltown/index.asp [8/10/4]

[Coursey-2003] Coursey, David, Broadband from the electric company? No thanks, October, 13, 2003, [Online]. Available: http://reviews-zdnet.com.com/4520-7297_16-5089730.html [7/24/4]

[CPTWY-2000] Cohen, N., Purakayastha, A., Turek, J., Wong, L., Yeh, D., Challenges in Flexible Aggregation of Pervasive Data, IBM T.J. Watson Research Center, Tech Report RC21942, 2000

[CRW-1999] Carzaniga, Antonio, Rosenblum, David S., and Wolf, Alexander L., Challenges for Distributed Event Services: Scalability vs. Expressiveness, May 1999.

[CRW-2000] Carzaniga. A., Rosenblum, D. S., and Wolf, A. L., Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service, *19th ACM Symposium on Principles of Distributed Computing*, pp 219-227, 2000

[CW-2003] Carzaniga, A., Wolf A., Forwarding in a Content-Based Network, SIGCOMM '03

[DAS-2001] Dey, A., Abowd, G., and Salber, D., A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications, Human-Computer Interaction, 16 (2-4), 97-166

[Dey-2000] Dey, Anind K., Providing Architectural Suppport for Building Context-Aware Applications, PhD Thesis, November 2000.

[DGPW-EMASMA-2003] Dance, Sandy et al, An Evolving Multi Agent System for Meteorological Alerts, AAMAS '03

[Dist11-2004] San Diego Speeds [Online]. Available http://www.dot.ca.gov/dist11/d11tmc/sdmap/speeds/ [12/18/4]

[Einhorn-2004] Einhorn, Bruce, China.Net, BusinessWeek Online, March 15, 2004 [Online] Available: http://www.businessweek.com/magazine/content/04_11/b3874012.htm [9/11/4]

[eMRG-2004] eMarket Research Group, Popup-ad [11/20/2004]

[FD-2004a] Metcalfe's Law, [Online] Available: http://encyclopedia.thefreedictionary.com/Metcalfe's%20law [9/7/4]

[FD-2004b] Reed's Law, [Online] Available: http://encyclopedia.thefreedictionary.com/Reed%27s%20law, [9/7/4]

[FGKZ-2003] Freige, L., Gartner, F., Kasten, O., Zeidler, A., Supporting Mobility in Content-based Publish / Subscribe Middleware, Middleware 2003, LNCS 2672, pp. 103-122, 2003.

[FZBKM-2004] Freige, L., Zeidler, A., Buchmann, A., Kilian-Kerr, R., Muhl, G., Security Aspects in Publish / Subscribe Systems, Proceedings Distributed Event Based Systems, 2004.

[Forrest-2003] Forrest, Ian, Elecrric Co Broadband up and running, October 13, 2003, [Online]. Available: http://reviews-zdnet.com.com/5208-6118-0.html?forumID=1&threadID=41&messageID=647&start=-1 [7/24/4]

[Fulford-2002] Fulford, Benjamin, Sensors Gone Wild, Forbes, 10/28/2, [Online]. Available:

http://www.forbes.com/global/2002/1028/076.html [8/10/4]

[Gaudin-2001] Gaudin, Sharon, Cell phone facts and statistics, Network World, 7/2/1 [Online] Available: http://www.nwfusion.com/research/2001/0702featside.html [6/24/4]

[GBBT-2003] Griswold, W., Boyer, R., Brown, S., and Truong, T., A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure, *2003 International Conference on Software Engineering*, May 2003

[GBBTBJS-2002] Griswold, W., Boyer, R., Brown, S., Truong, T., Bhasker, E., Jay, G., Shapiro, R., ActiveCampus – Sustaining Educational Communities through Mobile Technology, Technical Report CS2002-0714, Computer Science and Engineering, UC San Diego, July 2002

[GHJV-1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software* Reading, MA, Addison-Wesley, 1995

[Gilder-1993] Gilder, George, Metcalfe's Law and Legacy, Forbes, September 13, 1993, [Online] Available: http://www.seas.upenn.edu/~gaj1/metgg.html [9/7/4]

[Goldhaber-04] Goldhaber, Michael H., 2004. The Attention Economy and the Net, First Monday, [Online]. Available: http://xoomer.virgilio.it/macanigg/english.htm, [7/21/4]

[Google-2004] 2004, Google [Online]. Available: http://www.google.com/ [8/10/4]

[Gray-2004] 2004, Elisha Gray, [Online] Available: http://www.oberlin.edu/external/EOG/OYTT-images/ElishaGray.html [9/4/4]

[GreenHouse-2004] April 14, 2004, Green House Project [Online]. Available: http://thegreenhouseproject.com/news/news12.htm [8/10/4]

[GSBBRST-2003] Griswold, W., Shanahan, P., Brown, S., Boyer, R., Ratto, M., Shapiro, R., and Truong, T., ActiveCampus – Experiments in Community-Oriented Ubiquitous Computing, Technical Report CS2003-0750, UC San Diego, June 2003

[Hall-2003] Hall, William, Working towards biologically based theories of organizations and knowledge to understand how organizations work best. Tenix Defence Systems, 9/2003 [Online] Available: http://nzkm.sites.totallydigital.co.nz/assets/NZKMNetGovis(1).pdf [2/15/05]

[Harbor-2002a] The "Always On" Pervasive Internet, Harbor Research, January 2002

[Harbor-2002b] Catalytic Strategy, Harbor Research, January 2002

[Harbor-2002c] Evaluating Devices to be Networked, Harbor Research, July/August 2002

[Harbor-2003a] Core Networks:Can They Escape the Commoditization Spiral, Harbor Research, June 2003

[Harbor-2003b] Let the Circle Be Unbroken, Harbor Research, July 2003

[Harbor-2003c] The Pervasive Internet Opportunity, Harbor Research, December 2003

[Harrsch-2003] 2003, Pervasive Internet,Mary Harrsch, RSS: The Next Killer App For Education, [Online] Available: http://ts.mivu.org/default.asp?show=article&id=2010 [9/3/4]

[HG-2001] Huang, Y., and Garcia-Molina, H., Publish / Subscribe in a mobile environment. Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE01), Santa Barbara, CA, May 2001.

[HK-2003] Hill, Jonathan C., Knight, John C., Selective Notification: Combining Forms of Decoupled Addressing for Internet-Scale Command and Alert Dissemination, 2003

[HKCH-2002] Hill, Jonathan C., Knight, John C., Crickenberger, Aaron M., Honhard, Richard, Publish and Subscribe with Reply, Tech Report: CS-2002-32, Dept of Computer Science, University of Virginia

[HL-2001] Hong, J. and Landay, A., An Infrastructure Approach to Context-Aware Computing, *Human-Computer Interaction*, 16(2-4) 287-303, 2001

[HNV-2004] Hoffman, Donna L., Novak, Thomas P., and Venkatesh, Alladi, Has the Internet Become Indispensible?, Communications of the ACM, Vol 47, No. 7, July 2004, pp 37-42.

[HomePlug-2005] 2005, HomePlug Powerline Appliance [Online]. Available: http://www.homeplug.com/en/index.asp [4/25/5]

[Hong-2002] Hong, J., The Context Fabric: An Infrastructure for Context-Aware Computing, Proceedings CHI 2002, April 20-25, 2002

[House_n-2004] MIT, The MIT Home of the Future Consortium [Online]. Available: http://architecture.mit.edu/house_n [8/10/4]

[HP-2004] Krazits, Tom, HP announces new GSM/GPRS-enabled iPaq, July, 26, 2004 [Online] Available: http://www.thestandard.com/article.php?story=2004072614423020&mode=print [8/10/4]

[IBM-2001] IBM Research, Linux Watch, 10/11/2001, [Online]. Available: http://www.research.ibm.com/trl/projects/ngm/wp10_e.htm [11/13/4]

[ierp-2004] Microsoft, Intelligent Environments Resource Page [Online]. Available: http://www.research.microsoft.com/ierp/ [8/10/4]

[iicsc-2004] 2004, Information on Inexpensive Cellular Service in China, [Online] Available: http://www.chinatour.com/ad/cellphonechina.htm [9/3/2004]

[JS-2003] Jin, Y. and Strom, R., Relational Subscription Middleware for Internet-Scale Publish-Subscribe, *Proceedings Second International Workshop on Distributed Event-Based Systems*, 2003

[JSF-2004] Jabber Software Foundation. 2004. Jabber: Open Instant Messaging and a Whole Lot More, Powered by XMPP. [Online] Available: http://www.jabber.org/ [2/22/04]

[JXMPP-2004] JabberStudio. 2003. J-XMPP Project Information, [Online] Available: http://www.jabberstudio.org/projects/j-xmpp/project/view.php [2/22/04]

[Kiczales-96] Kiczales, Gregor, Beyond the Black Box: Open Implementation, *IEEE Software*, January 1996

[KLLMMM-97] Kiczales, G., Lamping, J., Lopes, C.V., Maeda, C., Mendhekar, A., Murphy, G., Open Implementation Design Guidelines, *1997 International Conference on Software Engineering (ICSE)*, May 1997

[KnowNow-2004] KnowNow, [Online] Available: http://www.knownow.com/forms/kn-summit-reprint-form.shtml [11/26/4]

[Khurana-2005] Khurana, Himanshu, Scalable Security and Accounting Services for Content-based Publish / Subscribe Systems, Proceedings Symposium on Applied Computing, March 13-17, 2005

[Kulik-2003] Kulik, J., Fast and Flexible Forwarding for Internet Subscription Systems, Proceedings Distributed Event-Based Systems, 2003

[Liard-2003a] Liard, Michael, The Global Markets and Applications for Radio Frequency Identification and Contactless Smartcard Systems, 4th Edition, Volume I, January 2003, [Online]. Available: http://www.vdc-corp.com/autoid/white/03/03rfid.pdf

[Liard-2003b] Liard, Michael, The Global Markets and Applications for Radio Frequency Identification and Contactless Smartcard Systems, 4th Edition, Volume II: RFID Transponder ICs, February 2003, [Online]. Available: http://www.vdc-corp.com/autoid/white/03/03rfidvol2.pdf

[Liard-2003c] Liard, Michael, RFID in the Supply Chain: The Wal-Mart Factor Part One: VDC's Firsthand Impressions of the November Mandate, 2003 [Online]. Available: http://www.vdc-corp.com/autoid/press/03/pr03-78.html [12/22/4]

[Liard-2004] Liard, Michael, RFID in the Supply Chain: The Wal-Mart Factor Part Two: Impact on Near-Term Market Growth, March 2004, [Online]. Available: http://www.vdc-corp.com/autoid/white/04/walfactor2.pdf [12/22/4]

[LNK-2005] Lee, G., Naganuma, T., Kurakake, S., Efficient Matching in a Context-Aware Event Notification System for Mobile Users, Proceedings Distributed Event-Based Systems, 2005

[Matias-2004] February 17, 2004, Nathan Matias, Get Off Your RSS!, [Online] Available: http://www.sitepoint.com/article/get-off-your-rss [6/26/4]

[Microsoft-2005] 2005, Microsoft Office, [Online]. Available: http://office.microsoft.com [4/25/5]i[Mone-2004] Mone, Jim, Fliers sign up for biometric ID project, San Diego North County Times, June 29, 2004 pg D-1.

[Moore-65] Moore, Gordon, Cramming more components onto integrated circuits, Electronics, vol. 38, number 8, April 19, 1965

[Muhl-2002] Muhl, G., Large-Scale Content-Based Publish / Subscribe Systems, PhD Thesis, Darmstadt University of Technology, 2002.

[MV-Mobi-2003] McNett, Marvin, and Voelker, Geoffrey M., Access and Mobility of Wireless PDA Users, UCSD MobiCom 2003 Poster, 2003

[Netlingo-2004] 2004, Gilder's Law, [Online] Available: http://www.netlingo.com/lookup.cfm?term=Gilder's%20Lawhttp://www.netlingo, [9/7/4]

[Newell-76] Newell, Allen, Fairy Tales, Speech at Carnegie Mellon University, 1976

[NH-2004] 2004, Novell & Hyperion, e-mail: Novell exteNd secure enterprise dashboard solution, [Online] Available: http://www.novell.com/collateral/4613355/4613355.html [8/10/4]

[Nokia-2004] Nokia, The Anytime Anyplace World, [Online] Available: http://now.eloqua.com/web/Nokia/Archetypes_White_Paper_US.pdf [6/24/4]

[ntia-2001] 2001, [Online] Available: http://www.ntia.doc.gov/otiahome/top/conferenceworkshops/2001_outreach_workshops/three_laws.html [9/7/4]

[Palm-2004] Palm One, Wireless Solutions, [Online] Available: http://www.palmone.com/us/wireless/ [8/6/4]

[Papadopoulos-2004] Papadopoulos, Greg, The Net Effect, How increased bandwidth is driving innovation in business and technology, Sun Microsystems, 2004

[PB-P2POBN-2003] Pietzuch, Peter R. and Bacon Jean, Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware, U.Cambridge, 2003?

[Philips-2004] Unknown, Whitepaper MP3RUN, digital audio player, Philips, June, 2004 [Online] Available: http://www.press.ce.philips.com/apps/c_dir/e3379701.nsf/0/AAAC14A1CD824621C1256EBB002F34BE/$File/Whitepaper%20MP3RUN.pdf [9/11/4]

[PHS-2004] 2004, PROMPT REPORT on the number of subscribers of cellular telephone and PHS(Personal Handy-phone System) at the last day of January 2003 in Japan, [Online] Available: http://www.soumu.go.jp/joho_tsusin/eng/Statistics/MobilePhone/handy-phone-imm-e0301.html [9/3/2004]

[Pinto-2002] 2002, Pinto, Jim, The Pervasive Internet and Its Effect on Industrial Automation, [Online] Available: http://www.automationtechies.com/sitepages/pid1020.php [8/9/4]

[Pinto-2004a] 2004, Pinto, Jim, The 3 Technology Laws, [Online] Available: http://www.automationtechies.com/sitepages/pid1010.php [8/9/4]

[Pinto-2004c] 2004, Pinto, Jim, Intelligent Connected Appliances, [Online] Available: http://www.automationtechies.com/sitepages/pid1028.php [8/9/4]

[Pinto-2004d] 2004, Pinto, Jim, Sensors Everywhere, Computers Invisible, [Online] Available: http://www.jimpinto.com/enews/aug2-2004.html [8/10/4]

[Pitre-1998] Pitre, Thomas, The Life of Information, 10/8/1998, [Online]. Available: http://www.suite101.com/article.cfm/electronic_information/11366 [8/8/4]

[PlaceLab-2004] 2004, Place Lab, Intel [Online]. Available: http://www.placelab.org/ [8/10/4]

[PLZ-2003] Padovitz, A., Loke, S.W., Zaslavsky, A., Using the Publish-Subscribe Communications Genre for Mobile Agents, MATES 2003, LNAI 2831, pp 180-191, Springer-Verlag, 2003

[Radicati-2004] Radicati Group, Inc., Enterprise Wireless Email Market Trends, 2003-2007, October 2003

[RDJ-2002] Rjaibi, W., Dittrich, K., Jaepel, D., Event Matching in Symmetric Subscription Systems, 2002, [Online]. Available:

[Reuters-2004] Microsoft, Swatch debut wireless watches, 10/20/2004 [Online]. Available: http://www.msnbc.msn.com/id/6290978/ [11/13/4]

[Rey-2004] Rey, Guido. 2004. Innocenzo Manzetti [Online]. Available: http://xoomer.virgilio.it/macanigg/english.htm [7/3/4]

[Rosenthal-2004] Rosenthal, Avi, Crunching Cell Phone Numbers, ElectronicHouse.com, 4/20/4 [Online] Available: http://www.electronichouse.com/default.asp?nodeid=2000 [6/25/4]

[RWC-1998] Rosenblum, David S., Wolf, Alexander L., and Carzaniga, Antonio, Critical Considerations and Designs for Internet-Scale, Event-Based Compositional Architectures, January 1998.

[Scuka-2004] Scuka, Daniel, Cell Phones That Surf for News, Japan Media Review, 6/25/4, [Online] Available: http://www.japanmediareview.com/japan/wireless/1046564940p.php [6/25/4]

[SDR-YANCEES-2003] Silva Filho, R. S., De Souza, C. R. B., Redmiles, D.F., The Design of a Configurable, Extensible, and Dynamic Notification Service, Proceedings Second International Workshop on Distributed Event-Based Systems (DEBS '03), 2003

[SE-MAPSEC-2000] Sahingoz, Ozgur Koray and Erdogan, Nadia, MAPSEC: Mobile-Agent Based Publish / Subscribe Platform for Electornic Commerce, Turkey, 2003?

[SER-2001] 2001, "Smart Environment" Research, [Online]. Available: http://www.equator.ac.uk/PublicationStore/Smart_Environments_booklet.pdf

[SIENA-2005] 2005, SIENA [Online]. Available http://serl.cs.colorado.edu/~carzanig/siena/software/index.html [4/25/5]

[Smith-2004] Smith, David, Web Services Enable Service Oriented and Event-Driven Architectures, Business Integration Journal, May 2004

[Symbol-2004] MC9000: Best-Practice Power Management Techniques Deliver Maximum Battery Efficiency, May 2004 Online Available: url in gmail [11/26/4]

[Tam-2004] Tam, Danny, The Net Effect:Looking Forward: Services on Demand, Sun Microsystems, 2004

[Tarasewich-2003] Tarasewich, P., Toward a Comprehensive Model of Context for Mobile and Wireless Computing, *Proceedings of America's Conference on Information Systems (AMCIS) 2003*, 2003

[TBFZB-P2P-2003] Terpstra, Wesley W., et al, A Peer-to-Peer Approach to Content-Based Publish/Subscribe, Proceedings Distributed Event Based Systems, 2003

[Textually-2004a] 3/16/4, Don't leave home without your keitai, [Online] Available: http://www.textually.org/textually/archives/000083.htmhttp://www.textually, [9/3/4]

[TIK-2004] 2004, Wireless @ TIK – Geolocation Resources, [Online] Available: http://www.tik.ee.ethz.ch/~beutel/positioning.html [8/10/4]

[ts-2004] http://www.trade-system.com/bmaster1.html

[TSSWM-1997] Tennenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., Minden, G., A Survey of Active Network Research, IEEE Communications, pp 80-86, January 1997.

[Tuomi-02] Tuomi, Ilkka, The Lives and Death of Moore's Law, First Monday, vol 7., number 11, November 2002

[ULib-2004] 2004, Universal Library, [Online].Available: http://www.ulib.org/html/index.html [8/10/2004]

[UPnP-2005] 2005, Welcome to the UPnP Forum! [Online]. Available: http://www.upnp.org [4/25/5]

[Warren-2001] Warren, Trevor, IBM's Linux Wristwatch, 3/15/1, [Online]. Available: http://www.freeos.com/articles/3800/ [11/13/4]

[Weiser-91] Weiser, Mark, The Computer for the 21st Century, Scientific American, pp94-104, September, 1991.

[Weiser-93] Weiser, Mark, Some Computer Science Issues in Ubiquitous Computing, CACM, July 1993

[WHFG-1992] Want, R., Hopper, A., Falcao, V., Gibbons, J., The Active Badge Location System, ACM Transactions on Information Systems, Vol. 40, No. 1, 91-102, January, 1992

[Wikipedia-2004] 2004, Moore's Law, Wikipedia, 12/5/2004 [Online]. Available: http://en.wikipedia.org/wiki/Moore's_Law [12/24/4]

[Winograd-2001] Winograd, T., Architectures for Context, *Human-Computer Interaction*, 16(2-4) 401-419, 2001

[WJH-97] Ward, Andy, Jones, Alan, Hopper, Andy,. A New Location Technique for the Active Office. *IEEE Personal Communications*, Vol. 4, No. 5, October 1997, pp. 42-47

[WSAGPEGW-1995] Want, R., Schilit, B., Adams, N., Gold, R., Petersen, K., Ellis, J., Goldberg, D., Weiser, M., The ParcTab Ubiquitous Computing Experiment, Technical Report CSL-95-1, Xerox Palo Alto Research Center, March, 1995

[X10KB-2004] 2004, X-10 Home Automation Knowledge Base [Online]. Available: http://www.geocities.com/ido_bartana/ [8/10/4]

[X10-2004] 2004, X10.com [Online]. Available: http://www.x10.com/homepage.htm [8/10/4]

[Yahoo-2004] 2004, Yahoo [Online]. Available: http://www.yahoo.com/ [8/10/4]

[ZigBee-2005] 2005, ZigBee Alliance [Online]. Available: http://www.zigbee.org/en/index.asp [4/25/5]

[CDMP-2000] Cheverst, K., Davies, N., Mitchell, K., Friday, A., Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project, Proceedings MobiCom, 2000, [Online]. Available: http://portal.acm.org/citation.cfm?id=345910.345916