Herald: Achieving a Global Event Notification Service

Luis Felipe Cabrera, Michael B. Jones, Marvin Theimer

Microsoft Research, Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

{cabrera, mbj, theimer}@microsoft.com
http://research.microsoft.com/~mbj/, http://research.microsoft.com/~theimer/

Abstract

This paper presents the design philosophy and initial design decisions of Herald: a highly scalable global event notification system that is being designed and built at Microsoft Research. Herald is a distributed system designed to transparently scale in all respects, including numbers of subscribers and publishers, numbers of event subscription points, and event delivery rates. Event delivery can occur within a single machine, within a local network or Intranet, and throughout the Internet.

Herald tries to take into account the lessons learned from the successes of both the Internet and the Web. Most notably, Herald is being designed, like the Internet, to operate correctly in the presence of numerous broken and disconnected components. The Herald service will be constructed as a set of protocols governing a federation of machines within cooperating but mutually suspicious domains of trust. Like the Web, Herald will try to avoid, to the extent possible, the maintenance of globally consistent state and will make failures part of the client-visible interface.

1. Introduction

The function of event notification systems is to deliver information sent by event publishers to clients who have subscribed to that information. Event notification is a primary capability for building distributed applications. It underlies such now-popular applications as "instant messenger" systems, "friends on line", stock price tracking, and many others [7, 3, 13, 16, 10, 15].

Until recently, most event notification systems were intended to be used as part of specific applications or in localized settings, such as a single machine, a building, or a campus. With the advent of generalized eCommerce frameworks, interest has developed in the provision of global event notification systems that can interconnect dynamically changing sets of clients and services, as well as to enable the construction of Internet-scale distributed applications. Such global event notification systems will need to scale to millions and eventually billions of users.

To date, general event notification middleware implementations are only able to scale to relatively small numbers of clients. For instance, Talarian, one of the more-scalable systems, according to published documentation, only claims that its "SmartSockets system was designed to scale to thousands of users (or processes)" [14, p. 17].

While scalability to millions of users has been demonstrated by centralized systems, such as the MSN and AOL Instant Messenger systems, we do not believe that a truly global general-purpose system can be achieved by means of such centralized solutions, if for no other reason than that there are already multiple competing systems in use. We believe that the only realistic approach to providing a global event notification system is via a federated approach, in which multiple, mutually suspicious parties existing in different domains of trust interoperate with each other.

A federated approach, in turn, implies that the defining aspect of the design will be the interaction protocols between federated peers rather than the specific architectures of client and server nodes. In particular, one can imagine scenarios where one node in the federation is someone's private PC, serving primarily its owner's needs, and another node is a mega-service, such as the MSN Instant Messenger service, which serves millions of subscribers within its confines.

The primary goal of the Herald project is to explore the scalability issues involved with building a global event notification system. The rest of this paper describes our design criteria and philosophy in Section 2, provides an overview of our initial design decisions in Section 3, discusses some of the research issues we are exploring in Section 4, presents related work in Section 5, and concludes in Section 6.

2. Goals, Non-Goals, and Design Strategy

The topic of event notification systems is a broad one, covering everything from basic message delivery issues to questions about the semantic richness of client subscription interests. Our focus is on the scalability of the basic message delivery and distributed state management capabilities that must underlie any distributed event notification system. We assume, at least until proven otherwise, that an event notification system can be decomposed into a highly-scalable base layer that has relatively simple se-

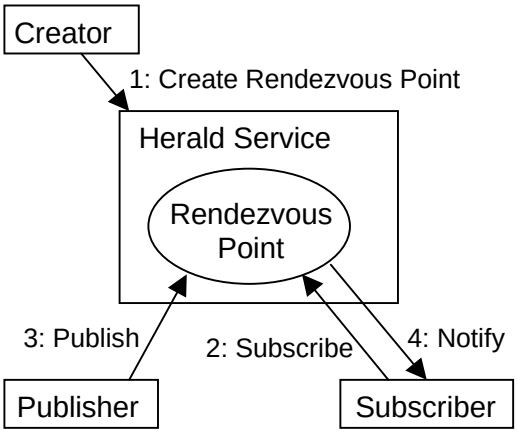

Figure 1: Herald Event Notification Model

mantics and multiple higher-level layers whose primary purposes are to provide richer semantics and functionality.

Consequently, we are starting with a very basic event notification model, as illustrated in Figure 1. In Herald, the term ***Event*** refers to a set of data items provided at a particular point in time by a *publisher* for a set of *subscribers*. Each subscriber receives a private copy of the data items by means of a *notification* message. Herald does not interpret the contents of the event data.

A ***Rendezvous Point*** is a Herald abstraction to which event publications are sent and to which clients subscribe in order to request that they be notified when events are published to the Rendezvous Point. An illustrative sequence of operations is: (1) a Herald client creates a new Rendezvous Point, (2) a client subscribes to the Rendezvous Point, (3) another client publishes an event to the Rendezvous Point, (4) Herald sends the subscriber the event received from the publisher in step 3.

This model remains the same in both the local and the distributed case. Figure 1 could be entirely upon a single machine, each of the communicating entities could be on separate machines, or each of them could even have distributed implementations, with presence on multiple machines.

2.1 Herald Design Criteria

Even with this simple model, there are still a variety of design criteria we consider important to try to meet:

- **Heterogeneous Federation:** Herald will be constructed as a federation of machines within cooperating but mutually suspicious domains of trust. We think it important to allow the coexistence of both small and large domains, containing both impoverished small device nodes and large mega-services.
- **Scalability:** The implementation should scale along all dimensions, including numbers of subscribers and publishers, numbers of event subscription points, rates of event delivery, and number of federated domains.
- **Resilience:** Herald should operate correctly in the presence of numerous broken and disconnected components. It should also be able to survive the presence of malicious or corrupted participants.
- **Self-Administration:** The system itself should make decisions about where data will be placed and how information should be propagated from publishers to subscribers. Herald should dynamically adapt to changing load patterns and resource availability, requiring no manual tuning or system administration.
- **Timeliness:** Events should be delivered to connected clients in a timely enough manner to support human-to-human interactions.
- **Support for Disconnection:** Herald should support event delivery to clients that are sometimes disconnected, queuing events for disconnected clients until they reconnect.
- **Partitioned operation:** In the presence of network partitions, publishers and subscribers within each partition should be able to continue communicating, with events being delivered to subscribers in previously separated partitions upon reconnection.
- **Security:** It should be possible to restrict the use of each Herald operation via access control to authenticated authorized parties.

2.2 Herald Non-Goals

As important as deciding what a system *will* do is deciding what it *will not* do. Until the need is proven, Herald will avoid placing separable functionality into its base model. Excluded functionality includes:

- **Naming:** Services for naming and locating Rendezvous Points are not part of Herald. Instead, client programs are free to choose any appropriate methods for determining which Rendezvous Points to use and how to locate one or more specific Herald nodes hosting those Rendezvous Points. Of course, Herald will need to export means by which one or more external name services can learn about the existence of Rendezvous Points, and interact with them.
- **Filtering:** Herald will not allow subscribers to request delivery of only some of the events sent to a Rendezvous Point. A service that filters events, for instance, by leveraging existing regular expression or query language tools, such as SQL or Quilt engines, and only delivering those matching some specified criteria, could be built as a separate service, but will not be directly supported by Herald.
- **Complex Subscription Queries: Herald** has no notion of supporting notification to clients interested in complex event conditions. Instead, we assume that complex subscription queries can be built by deploying agent processes that subscribe to the relevant Rendezvous Points for simple events and then publish

an event to a Rendezvous Point corresponding to the complex event when the relevant conditions over the simple event inputs become true.

- **In-Order Delivery:** Because Herald allows delivery during network partitions—a normal condition for a globally scaled system—different subscribers may observe events being delivered in different orders.

2.3 Applying the Lessons of the Internet and the Web

Distributed systems seem to fall into one of two categories—those that become more brittle with the addition of each new component and those that become more resilient. All too many systems are built assuming that component failure or corruption is unusual and therefore a special case—often poorly handled. The result is brittle behavior as the number of components in the system becomes large. In contrast, the Internet was designed assuming many of its components would be down at any given time. Therefore its core algorithms had to be tolerant of this normal state of affairs. As an old adage states: "The best way to build a reliable system is out of presumed-to-be-broken parts." We believe this to be a crucial design methodology for building any large system.

Another design methodology of great interest to us is derived from the Web, wherein failures are basically thrown up to users to be handled, be they dangling URL references or failed retrieval requests. Stated somewhat flippantly: "If it's broken then don't bother trying to fix it." This minimalist approach allows the basic Web operations to be very simple—and hence scalable—making it easy for arbitrary clients and servers to participate, even if they reside on resource-impoverished hardware.

Applied to Herald, these two design methodologies have led us to the following decisions:

- *Herald peers treat each other with mutual suspicion and do not depend on the correct behavior of any given, single peer*. Rather, they depend on replication and the presence of sufficiently many well-behaved peers to achieve their distributed systems goals.
- *All distributed state is maintained in a weakly consistent soft-state manner and is aged, so that everything will eventually be reclaimed unless explicitly refreshed by clients*. We plan to explore the implications of making clients responsible for dealing with weakly consistent semantics and with refreshing the distributed state that is pertinent to them.
- *All distributed state is incomplete and is often inaccurate*. We plan to explore how far the use of partial, sometimes inaccurate information can take us. This is in contrast to employing more accurate, but also more expensive, approaches to distributed state management.

Another area of Internet experience that we plan to exploit is the use of overlay networks for content delivery [5, 6]. The success of these systems implies that overlay networks are an effective means for distributing content to large numbers of interested parties. We plan to explore the use of dynamically generated overlay networks among Herald nodes to distribute events from publishers to subscribers.

3. Design Overview

This section describes the basic mechanisms that we are planning to use to build Herald. These include replication, overlay networks, ageing of soft state via time contracts, limited event histories, and use of administrative Rendezvous Points for maintenance of system meta-state. While none of these are new in isolation, we believe that their combination in the manner employed by Herald is both novel, and useful for building a scalable event notification system with the desired properties. We hypothesize that these mechanisms will enable us to build the rest of Herald as a set of distributed policy modules.

3.1 Replication for Scaling

When a Rendezvous Point starts causing too much traffic at a particular machine, Herald's response is to move some or all of the work for that Rendezvous Point to another machine, when possible. Figure 2 shows a possible state of three Herald server machines at locations L1, L2, and L3, that maintain state about two Rendezvous Points, RP1 and RP2. Subscriptions to the Rendezvous Points are shown as Sub*n* and publishers to the Rendezvous Points are shown as Pub*n*.

The implementation of RP1 has been distributed among all three server locations. The Herald design allows potential clients (both publishers and subscribers) to interact with any of the replicas of a Rendezvous Point for any operations, since the replicas are intended to be functionally equivalent. However, we expect that clients will typically interact with the same replica repeatedly, unless directed to change locations.

3.2 Replication for Fault-Tolerance

Individual replicas do not contain state about all clients. In Figure 2, for instance, Sub5's subscription is recorded only by RP1@L3 and Pub2's right to publish is recorded only by RP2@L1. This means that event notifications to these subscriptions would be disrupted should the Herald servers on these machines (or the machines themselves) become unavailable.

For some applications this is perfectly acceptable, while for others additional replication of state will be necessary. For example, both RP1@L1 and RP1@L2 record knowledge of Sub2's subscription to RP1, providing a degree of fault-tolerance that allows it to continue receiving notifications should one of those servers become unavailable.

Since RP1 has a replica on each machine, it is tolerant of faults caused by network partitions. Suppose L3

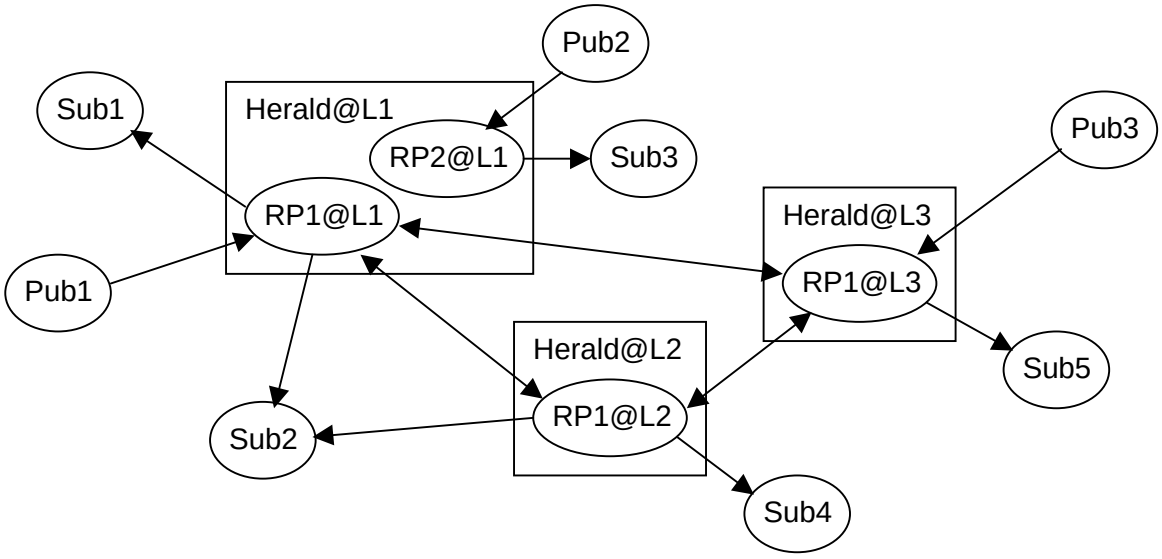

Figure 2: Replicated Rendezvous Point RP1 and Fault-Tolerant Subscription Sub2

became partitioned from L1 and L2. In this case, Pub1 could continue publishing events to Sub1, Sub2, and Sub4 and Pub3 could continue publishing to Sub5. Upon reconnection, these events would be sent across the partition to the subscribers that hadn't yet seen them.

Finally, note that since it isn't (yet) replicated, should Herald@L1 go down, then RP2 will cease to function, in contrast to RP1, which will continue to function at locations L2 and L3.

3.3 Overlay Distribution Networks

Delivery of an event notification message to many different subscribers must avoid repeated transmission of the same message over a given network link if it is to be scalable. Herald implements event notification by means of multicast-style overlay distribution networks.

The distribution network for a given Rendezvous Point consists of all the Herald servers that maintain state about publishers and/or subscribers of the Rendezvous Point. Unicast communications are used to forward event notification messages among these Herald servers in much the same way that content delivery networks do among their interior nodes. However, unlike most content delivery networks, Herald expects to allow multiple geographically distributed publishers. Delivery of an event notification message to the subscribers known to a Herald server is done with either unicast or local reliable multicast communications, depending on which is available and more efficient.

In order to implement fault tolerant subscriptions, subsets of the Herald servers implementing a Rendezvous Point will need to coordinate with each other so as to avoid delivering redundant event notifications to subscribers. Because state can be replicated or migrated between servers, the distribution network for a Rendezvous Point can grow or shrink dynamically in response to changing system state.

3.4 Time Contracts

When distributed state is being maintained on behalf of a remote party, we associate a time contract with the state, whose duration is specified by the remote party on whose behalf the state is being maintained. If that party does not explicitly refresh the time contract, the data associated with it is reclaimed. Thus, knowledge of and state about subscribers, publishers, Rendezvous Point replicas, and even the existence of Rendezvous Points themselves, is maintained in a soft-state manner and disappears when not explicitly refreshed.

Soft state may, however, be maintained in a persistent manner by Herald servers in order to survive machine crashes and reboots. Such soft state will persist at a server until it is reclaimed at the expiration of its associated time contract.

3.5 Event History

Herald allows subscribers to request that a history of published events be kept in case they have been disconnected. Subscribers can indicate how much history they want kept and Herald servers are free to either accept or reject requests.

History support imposes a storage burden upon Herald servers, which we bound in two ways. First, the creator of a Rendezvous Point can inform Herald of the maximum amount of history storage that may be allocated at creation time. As with subscriptions, servers are free to reject creation requests requiring more storage than their policies or resources allow.

Second, because clients and servers both maintain only ageing soft state about one another, event history

information kept for dead or long-unreachable subscribers will eventually be reclaimed.

While we recognize that some clients might need only a synopsis or summary of the event history upon reconnection, we leave any such filtering to a layer that can be built over the basic Herald system, in keeping with our Internet-style philosophy of providing primitives on which other services are built. Of course, if the last event sent will suffice for a summary, Herald directly supports that.

3.6 Administrative Rendezvous Points

One consequence of name services being outside Herald is that when Herald changes the locations at which a Rendezvous Point is hosted, it will need to inform the relevant name servers of the changes. In general, there may be a variety of parties that are interested in learning about changes occurring to Rendezvous Points and the replicas that implement them.

Herald notifies interested parties about changes to a Rendezvous Point by means of an *administrative* Rendezvous Point that is associated with it. By this means we plan to employ a single, uniform mechanism for all client-server and server-server notifications.

Administrative Rendezvous Points do not themselves have other Administrative Rendezvous Points associated with them. Information about their status is communicated via themselves.

4. Research Issues

In order to successfully build Herald using the mechanisms described above, we will have to tackle a number of research issues. We list a few of the most notable ones below.

The primary research problem we face will be to develop effective policies for deciding when and how much Rendezvous Point state information to move or replicate between servers, and to which servers. These policies will need to take into account load balancing and fault-tolerance concerns, as well as network topology considerations, for both message delivery and avoidance of unwanted partitioning situations. Some of the specific topics we expect to address are:

- determining when to dynamically add or delete servers from the list of those maintaining a given Rendezvous Point,
- dynamic placement of Rendezvous Point state—especially event histories—to minimize the effects of potential network partitions,
- dynamically reconfiguring distributed Rendezvous Point state in response to global system state changes,
- dealing with "sudden fame", where an Internet-based application's popularity may increase by several orders of magnitude literally overnight, implying that our algorithms must stand up to rapid changes in load.

Since we plan to rely heavily on partial, weakly consistent, sometimes inaccurate, distributed state information, a key challenge will be to explore how well one can manage a global service with such state. Equally important will be to understand what the cost of disseminating information in this fashion is.

It is an open question exactly how scalable a reliable multicast-style delivery system can be, especially when multiple geographically dispersed event publishers are allowed and when the aggregate behavior of large numbers of dynamically changing Rendezvous Points is considered. In addition, Herald requires that event notifications continue to be delivered to reachable parties during partitions of the system and be delivered "after the fact" to subscribers who have been "disconnected" from one or more event publication sources. To our knowledge, operation of delivery systems under these circumstances has not yet been studied in any detail.

Herald's model of a federated world in which foreign servers are not necessarily trustworthy implies that information exchange between servers may need to be secured by means such as Byzantine communication protocols or statistical methods that rely on obtaining redundant information from multiple sources. Event notification messages may need to be secured by means such as digital signatures and "message chains", as described, for example, in [12].

Another scaling issue is how to deal with access control for large numbers of clients to a Rendezvous Point. For example, consider the problem of allowing all 280 million U.S. citizens access to a particular Rendezvous Point, but no one else in the world.

Finally, Herald pushes a number of things often provided by event notification systems, such as event ordering and filtering, to higher layers. It is an open question how well that will work in practice.

5. Related Work

The Netnews distribution system [8] has a number of attributes in common with Herald. Both must operate at Internet scale. Both propagate information through a sparsely connected graph of distribution servers. The biggest difference is that for Netnews, human beings design and maintain the interconnection topology, whereas for Herald, a primary research goal is to have the system automatically generate and maintain the interconnection topology. The time scales are quite different as well. Netnews propagates over time scales of hours to weeks, whereas Herald events are intended to be delivered nearly instantaneously to connected clients.

A number of peer-to-peer computing systems, such as Gnutella [2], have emerged recently. Like Herald, they are intended to be entirely self-organizing, utilizing resources on federated client computers to collectively provide a global service. A difference between these services and

Herald is that the former typically use non-scalable algorithms, including broadcasts. Unlike Herald, with the exception of Farsite [1], these services also typically ignore security issues and are ill prepared to handle malicious participants.

Using overlay networks for routing content over the underlying Internet has proven to be an effective methodology. Examples include the MBONE [11] for multicast, the 6BONE [4] for IPv6 traffic, plus content distribution networks such as Overcast [6] and Inktomi's broadcast overlay network [5]. They have demonstrated the same load-reducing benefits for information dissemination to large numbers of clients needed for Herald. However, most work has focused on single-sender dissemination networks. Furthermore, they have not investigated mechanisms and appropriate semantics for continued operation during partitions.

The OceanStore [9] project is building a global-scale storage system using many of the same principles and techniques planned for Herald. Both systems are built using unreliable servers, and provide reliability through replication and caching. Both intend to be self-monitoring and self-tuning.

6. Conclusions

Global event notification is emerging as a key technology underlying numerous distributed applications. With the requirements imposed by use of these applications at Internet scale, the need for a highly scalable event notification system is clear.

We have presented the requirements and design overview for Herald, a new event notification system designed to fill this need. We are currently implementing the mechanisms described in this paper and are planning to then experiment with a variety of different algorithms and policies to explore the research issues we have identified.

Acknowledgments

The authors would like to thank Yi-Min Wang, Dan Ling, Jim Kajiya, and Rich Draves for their useful feedback on the design of Herald.

References

[1] Bill Bolosky, John Douceur, David Ely, and Marvin Theimer. Evaluation of Desktop PCs as Candidates for a Serverless Distributed File System. In *Proceedings of Sigmetrics 2000*, Santa Clara, CA, pp. 34-43, ACM, June 2000.

[2] *Gnutella: To the Bandwidth Barrier and Beyond*. Clip2.com, November 6th, 2000. http://www.gnutellahosts.com/gnutella.html.

[3] David Garlan and David Notkin. Formalizing Design Spaces: Implicit Invocation Mechanisms. In *Proceedings of Fourth International Symposium of VDM Europe: Formal Software Development Methods*, Noordwijkerhout, Netherlands, pp. 31-44, October, 1991. Also appears as Springer-Verlag *Lecture Notes in Computer Science 551*.

[4] Ivano Guardini, Paolo Fasano, and Guglielmo Girardi. *IPv6 operational experience within the 6bone*. January 2000. http://carmen.cselt.it/papers/inet2000/index.htm.

[5] *The Inktomi Overlay Solution for Streaming Media Broadcasts*. Technical Report, Inktomi, 2000. http://www.inktomi.com/products/media/docs/whtpapr.pdf.

[6] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, San Diego, CA, pp. 197-212. USENIX Association, October 2000.

[7] Astrid M. Julienne and Brian Holtz. *ToolTalk and Open Protocols: Inter-Application Communication*. Prentice Hall, 1994.

[8] Brian Kantor and Phil Lapsley. *Network News Transfer Protocol* (NNTP). Network Working Group Request for Comments 977 (RFC 977), February 1986. http://www.ietf.org/rfc/rfc0977.txt.

[9] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, November 2000.

[10] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The Information Bus – An Architecture for Extensible Distributed Systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, Asheville, NC, pp. 58-68, December 1993.

[11] Kevin Savetz, Neil Randall, and Yves Lepage. *MBONE: Multicasting Tomorrow's Internet*. IDG, 1996. http://www.savetz.com/mbone/.

[12] Mike J. Spreitzer, Marvin M. Theimer, Karin Petersen, Alan J. Demers, and Douglas B. Terry. Dealing with Server Corruption in Weakly Consistent, Replicated Data Systems, In *Wireless Networks*, ACM/Baltzer, 5(5), 1999, pp. 357-371. A shorter version appears in *Proceedings of the Third Conference on Mobile Computing and Networking*, ACM/IEEE, Budapest, Hungary, September 1997.

[13] Rob Strom, Guruduth Banavar, Tushar Chandra, Marc Kaplan, Kevan Miller, Bodhi Mukherjee, Daniel Sturman, and Michael Ward. Gryphon: An Information Flow Based Approach to Message Brokering. In *Proceedings of International Symposium on Software Reliability Engineering '98*, Fat Abstract, 1998.

[14] Talarian Inc. *Talarian: Everything You Need To Know About Middleware*. http://www.talarian.com/industry/middleware/whitepaper.pdf.

[15] TIBCO Inc. *Rendezvous Information Bus*. http://www.rv.tibco.com/datasheet.html, 2001.

[16] David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, Bill Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. In *Proceedings of the First International Workshop on Mobile Agents*, April, 1997.