A Stateful and Open Publish-Subscribe Structure for Online Marketplaces

Ingar Mæhlum Arntzen Dag Johansen

Department of Computer Science
University of Tromsø, Norway
{ingarma,dag}@cs.uit.no

Abstract

We observe a mismatch between how academia models publish-subscribe systems for marketplace applications and how such applications are actually structured by the industry. To address this mismatch, we propose a new structure for online marketplace applications. A stateful and open publish-subscribe system exposes internal state and events to clients, through both a search mechanism and a subscription mechanism. This way, sellers and buyers may get valuable insight into the activity of the participants of the marketplace. Admittedly, this contradicts the idea that publishers and subscribers should be strictly decoupled. However, we argue that at a high level of abstraction this decoupling is unnecessary, or even unwanted, for many marketplace applications. Additionally, the structure presents the new idea of recursive subscriptions, and argues for its applicability in marketplace applications.

1. Introduction

The current Web basically grew from a small scale implementation of the client-server model. In this pull-based model, a user issues a request to some remote Web server through a browser. Next, he waits for a response, before he can parse and validate the received data. This is a simple, but adequate model for pulling down static HTML pages. However, many of the services available on the Web today are extremely dynamic, with results that are relevant for only a short window of time. With such services, the pull-based access pattern entails a severe overhead for servers, for the network, and for the end-user.

In the WAIF[1] [11] project[2], we conjecture that the Internet should be more push-based. In our approach, we build mediator structures [14] turning passive existing Web services into active publishers. Similarly, we turn traditional browsing clients into asynchronous subscribers. We do not envision that we can change the API of, for instance, Google or Amazon, but instead we can build external proxy structures turning such services into push-based ones. Next, we configure and build distribution and fusion networks between these Web publishers and remote users. This overlay structure is transparent for the user, giving the impression of a personal overlay network system (PONS) pushing data towards the individual user. Expressiveness is high in a PONS since we install and extend the network with programmable filters [2]. These filters run asynchronously, and may occasionally push notifications back to its user. This resembles much how we previously found single-hop mobile agents useful [9, 10]. The WAIF PONS architecture also supports mobility of users [8]. This may be achieved by redirecting the last hop [15], or by dynamically migrating the filters internally to the PONS.

As part of transforming existing and popular Web services into PONS publishers, we identified a certain class of applications especially well-suited for the publish-subscribe paradigm. In online marketplaces such as eBay and Amazon, sellers publish products for sale, and buyers want timely notifications of such publications. Yet the services were not implemented using one of the many publish-subscribe protocols in existence. Consequently, we were interested in studying this mismatch.

The rest of this paper is organized as follows. Section 2 describes inadequacies of the classic publish-subscribe protocol as a basis for marketplace applications. These inadequacies relate to the statelessness, the lack of openness, and the uni-directionality of traditional publish-subscribe systems. To address these inadequacies, we present a stateful and open publish-subscribe structure. A prototype marketplace application, based on this structure, is presented in section 3. Section 4 discusses experiences so far with the structure. In addition, we elaborate the idea of recursive subscriptions and discuss how the issue of expressiveness in publish-subscribe systems relates to our research. The paper is summarized in section 5.

[1]Web of Asynchronous Information Filters, is a joint project with Cornell University and UC San Diego. http://waif.cs.uit.no

[2]This work is partially sponsored by the NFR (Norwegian Research Council) IKT-2010 Program.

2. A publish-subscribe structure for online marketplaces

Online marketplaces allow a large community of clients to sell or buy products over the Internet. Due to its dynamic character, this class of applications is ideal for the publish-subscribe paradigm. For example, research in publish-subscribe systems would typically model the marketplace as follows: Sellers publish products or price updates into the publish-subscribe system, and buyers subscribe to timely notifications of some relevant subset of these publications.

The industry seems to model the online marketplace differently. Popular implementations of online marketplaces are not structured as publish-subscribe systems. Instead, they are essentially pull-based systems, as are most services on the Web. This implies that products are published to the marketplace by storing them in a Web-interfaced database. Buyers (and sellers) then gain access to products and prices by navigating, browsing, or querying the system. To address the problem of clients requiring timely access to publications, the trend is to complement the pull-based structure with primitive subscription mechanisms. Google has recently extended its service by launching Google Alerts, allowing clients to subscribe to e.g. news updates by topic.

We observe a mismatch between how academia and industry structure online marketplaces. This may be an indication that the publish-subscribe protocol has inadequacies as a structure for online marketplace applications.

2.1. Inadequacies of the classic publish-subscribe protocol

By adopting the perspective of the end-users, a comparison of academic and industrial models of the online marketplace reveals some inadequacies of the classic publish-subscribe protocol.

The classic publish-subscribe protocol typically models a uni-directional stateless flow of data from publishers to subscribers. By defining predicates (subscriptions) on this flow, subscribers may receive notifications on some relevant subset of all data in the flow. This system model has grown out of research in group communication and multicast systems [3]. A major focus for many publish-subscribe systems in academia has been efficient and timely large-scale data dissemination [5, 16]. The decoupling (time, space, synchronization) of publishers and subscribers makes publish-subscribe a powerful communication model for distributed systems [6]. This system model is typically used for applications like stock brokering [1], or as middleware for gluing event-based software modules together [4].

The marketplace is not stateless

In contrast to the classic publish-subscribe protocol the marketplace is not stateless. In the view of the user, a marketplace has products for sale, it has prices, it has sellers and buyers, and it has offers and bids. In addition, it keeps historical data required for understanding the current situation of demand and supply. Even though this state is updated frequently, it is stable enough for snapshots to be meaningful. Consequently, a stateful publish-subscribe system may be complemented with a search mechanism.

Adding a search mechanism is attractive because it may increase the usability of the system. Prior to defining subscriptions, the end-user needs to know what data is likely to be published through the system. A good overview of what data the system offers is the key to high recall and precision. Subscribers do not want to miss out on relevant publications, yet subscriptions that are too broad are not an option. In stateless publish-subscribe systems it is difficult to provide the end-user with this essential overview.

A trivial solution is that the user subscribes initially to everything, and then later un-subscribes as he receives data that is not relevant. Topic-based systems help the user by partitioning the information space into topics. In content-based systems such as Siena [4], *advertisements* are broadcast to give subscribers a hint of what data to expect.

Pull-based systems address this problem in a different way. Marketplace applications like, for instance eBay, demonstrate that browsing, navigation and querying are excellent ways of investigating what the system has to offer. This way, end-users of a stateful publish-subscribe system could browse e.g. topics, publisher profiles, or publication history, prior to making any subscriptions. Hence, a sophisticated search mechanism may be a powerful way of presenting a stateful publish-subscribe system to the end-user.

The marketplace is not a black box

By protecting its internal state (e.g. the subscriptions) publish-subscribe systems are often viewed as a black box by its clients. For example, sellers do not even know if any buyers subscribe to what they are publishing. Buyers on their side do not know if their subscriptions are unique, or shared by thousands of other buyers. In contrast, we argue that participants of the marketplace need to be well informed to make a bargain. For example, in order to understand the supply situation of a given product, buyers need to know the number of potential sellers of the product. To understand the demand, buyers need to know the number of competing buyers. Sellers too may benefit from such information. Understanding demand and supply may help them set reasonable prices, or delay the publication of a product until the demand is peaking. By exposing the internals of the publish-subscribe system (e.g. the subscriptions) this information can be accessed by participants of the marketplace. We call such a publish-subscribe system *open*.

Other research projects have previously argued for client access to subscriptions. For example, the Elvin [13] project defines a *quenching* mechanism that allows clients to drop publications if nobody subscribes to them. The markeplace application encourages us to revisit this idea.

Related to the issue of openness is the issue of anonymity. The publish-subscribe protocol often provides anonymity by decoupling publishers from subscribers. This is not always appropriate for the marketplace. For example, sellers that do not know their customers are not able to adapt or improve their service. Similarly, buyers that do not know their service providers have difficulties signaling interests. An open publish-subscribe system may allow sellers and buyers to know each others identity.

The marketplace is not uni-directional

The information flow of the marketplace is not uni-directional, from sellers to buyers, as would be a direct application of the publish-subscribe protocol. Rather, data flows both ways, and both sellers and buyers may want to publish data or subscribe to publications. For example, a buyer may want to publish his interest in buying a certain product, and sellers may want to subscribe to such publications. Hence, when designing a marketplace, it is important to meet the requirements of both sellers and buyers, and allow them to cooperate to reach a common goal. Industrial stock broker systems illustrate this bi-directionality by defining two types of publications; *bids* and *offers*. Sellers publish offers, and buyers publish bids. Both parties may subscribe to any of these publication types.

2.2. A stateful and open publish-subscribe structure

To address the inadequacies presented above, we present a new structure for online marketplaces. The overall idea is that online marketplaces typically need both publish-subscribe mechanisms and search mechanisms. The new structure is constructed by merging properties of publish-subscribe systems and search systems. The structure is a stateful publish-subscribe system that exports its internal state and events through a search-mechanism and a subscription-mechanism, respectively.

A simple formalism for search-systems and subscription-systems is necessary to merge the two. Search-systems have internal state that is available by querying. Incoming queries are matched against internal state and query-results are output. In contrast, subscription-systems focus on exporting internal events, rather than state. Subscribers may subscribe to the event of a new publication. Incoming publications are matched against persistent subscriptions, and notifications are output to the subscribers.

Merging the publish-subscribe mechanism with the search mechanism is not very complicated. We choose the best of both worlds. Our structure has both internal state and internal events. Furthermore, it exports its state by a search mechanism, and its events by a subscription mechanism. This makes sense for the online marketplace. For example, buyers need to search for products that are currently for sale, and they need to subscribe to the event of a new product being published. Note that the internal events of such a system typically correspond to updates of the internal state of the system.

This structure, a stateful publish-subscribe system [7], complemented with a search mechanism, is not a new idea. What is new, is the openness suggested by the above discussion of publish-subscribe inadequacies. The structure allows exposure of all relevant internal state, and all relevant internal events, using the search mechanism and the subscription mechanism. This yields a publish-subscribe system with two new and important features. First, by using the search mechanism, clients may browse both the publications and the subscriptions that are stored in the system. Second, by using the subscription mechanism, clients may subscribe to internal events like incoming publications, incoming search-queries, or even incoming subscriptions.

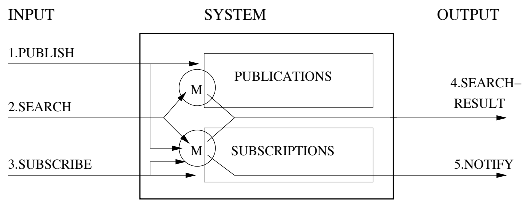

Figure 1. A stateful and open publish-subscribe structure.

Figure 1 illustrates the structure. Squares represent persistent state, and circles represent matching logic. The matching logic executes as a response to input, and has access to internal state. 1) PUBLISH: Products for sale are published to the system, and stored persistently until the product is no longer for sale. In addition, publications are matched against stored subscriptions. 2) SEARCH: Search-queries must be matched against the state of the system. Our structure allows both publications and subscriptions to be queried. Also, to accomodate subscriptions to incoming queries, search-queries are matched against this type of subscriptions. 3) SUBSCRIBE: Subscriptions are stored and bound to internal events. Any input to the system defines an event. Consequently, subscriptions may be bound to incoming publications, search-queries, or even subscriptions. For this reason, incoming subscriptions too must be matched against already stored subscriptions. 4) SEARCH-RESULT: Search-results are returned to the client synchronously. 5) NOTIFY: In the event of a match, asynchronous notifications are output to the subscriber. 1) and 3) also have corresponding un-publish and un-subscribe functionality. Subscriptions may be bound to un-publish and un-subscribe events as well.

2.3. Addressing the inadequacies

We argue that the structure presented above may help to improve some of the inadequacies discussed earlier.

The marketplace is not stateless

By storing both publications and subscriptions persistently, the structure implements a *stateful* marketplace. The structure also includes a search mechanism that promises to give end-users an overview of the marketplace. For example, buyers may query for stored publications to find products for sale. Sellers too may find this useful, for instance to monitor publications of competing sellers. By querying stored subscriptions both sellers and buyers get an impression of the popularity of different products.

The marketplace is not a black box

The marketplace structure is open. This means that end-users may search internal state and subscribe to internal events. The internal state includes publications and subscriptions. The internal events are input to the system: New publications, new queries, or new subscriptions. Hence, by searching and subscribing, buyers and sellers gain insight in the state and activity of the marketplace. For example, a seller may subscribe to queries made to his own products. Buyers and sellers may even subscribe to new subscriptions made by others. Expert users may find this useful to better understand how demand and supply are fluctuating in the marketplace. Especially, the combination of search and subscription mechanisms may prove powerful in this setting.

The marketplace is not uni-directional

The structure has facilities that are attractive to both sellers and buyers. For example, buyers are primarily interested in products, so they subscribe to publications of new products. Sellers, on the other hand, are interested in buyers and their interests. The subscriptions and queries made by buyers express these interests. Therefore, sellers may learn more about potential buyers by searching for and subscribing to these subscriptions. This way, the structure has something to offer for both the sellers and the buyers. By being open and possibly non-anonymous, our marketplace structure also recognizes that sellers and buyers need to co-operate to reach a common goal.

3. Implementation

A *Python* prototype of the publish-subscribe marketplace application has been implemented [12]. This serves as a proof of concept for the ideas presented in this paper. The prototype marketplace allows users to sell or buy products from a wide range of categories, e.g. *books*, *bikes*, and *boats*. The implementation demonstrates certain features of a stateful and open marketplace, for example search for publications and subscriptions to new subscriptions. The latter are named *sub-subscriptions* to distinguish them from regular *subscriptions* (to new publications).

The prototype system is essentially a centralized, topic-based publish-subscribe system, but has additional support for simplistic content-based filtering, e.g. (price $<$ 10\$). The topic namespace is flat and discrete. Both regular subscriptions and sub-subscriptions are bound to topics. A sub-subscription to a given topic means that a notification is expected whenever regular subscriptions are made to the same topic. To emphasize the semantic differences between sub-subscriptions and regular subscriptions, sub-subscriptions are modeled as a separate entity in the system.

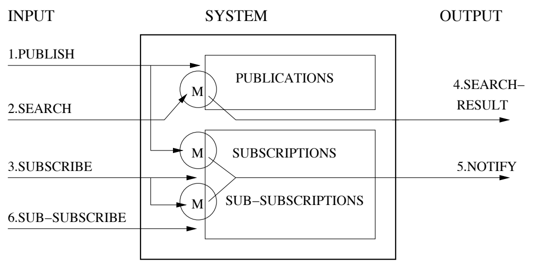

Figure 2. Publish-subscribe system with subscribe-enabled subscriptions.

The overall functionality of the prototype, and its divergences from the structure presented in section 2.2, are illustrated by figure 2. 1) PUBLISH: and 4) SEARCH-RESULT: are unchanged from the presentation given in figure 1. 2) SEARCH: Search-queries are restricted to be matched against publications only, not subscriptions or sub-subscriptions. 3) SUBSCRIBE: Subscriptions are stored and bound to topics. In addition, they are matched against stored sub-subscriptions. 5) NOTIFY: Notifications to clients may be triggered in two ways. Either a new publication matches a subscription or a new subscription matches a sub-subscription. 6) SUB-SUBSCRIBE: Sub-subscriptions are simply stored in the system.

3.1. Data representation and matching

A publication is represented internally as a tuple of three attributes; (topic, price, keywords). The topic attribute is a string representing a product category like e.g. *book* or *bike*. The price attribute is a float representing the price of the product, set by the seller. The keywords attribute is additional metadata, used for content-based filtering and search.

Subscriptions are represented in the same way, but the price attribute is interpreted as maximum price. The matching of publications and subscriptions is done on a per-attribute basis: The topics must match exactly. Price and keyword attributes are optional. If maximum price is given by the subscription, a matching publication must have a lower price. If keywords are given by the subscription, at least one of the keywords must match one of the publication keywords. Subscriptions with the price or keyword attribute set, allow content-based filtering of notifications on a given topic.

Sub-subscriptions do not have a price attribute. Apart from that, the matching between subscriptions and sub-subscriptions is equal to the matching between publications and subscriptions. Publications, subscriptions, and sub-subscriptions additionally have system specific attributes such as owner identification and time-to-live.

Notifications are simply copies of the events causing a match. The notification resulting from a match between a publication and a subscription is hence a copy of the triggering publication. Similarly, when a match occurs between a subscription and a sub-subscription, the resulting notification is a copy of the triggering subscription.

3.2. API and core functionality

Most of the public API of the prototype marketplace, along with its core functionality, is explained by the pseudocode given in figure 3. The code refers to a topic index and a keyword index. These allow fast lookup of stored publications, subscriptions and sub-subscriptions and hence improve the efficiency of matching.

In addition, a *Remove* method is used to delete publications, subscriptions and sub-subscriptions when they are no longer valid. This method may be invoked by the user explicitly, or by the system after the entity lifespan has expired. *Notify* is called by the system to deliver notifications. Currently notifications may be delivered as emails, or as events in the WAIF PONS.

```
var storage   // Reference to storage module
var t_index   // Reference to topic_index
var k_index   // Reference to keyword_index
var match     // Reference to matching function
var notify    // Reference to notify function
SEARCH (topic, price, keywords)
    result = []
    q = Query (topic, price, keywords)
    pubs = t_index.getPublications (topic)
    for pub ∈ pubs do
        if match (q, pub) then result.append(pub)
    return result
PUBLISH (topic, price, keywords)
    p = Publication (topic, price, keywords)
    ref = storage.dump (p)
    // Update indexes
    t_index.update (p, ref)
    k_index.update (p, ref)
    // Match publication with subscriptions
    subs = t_index.getSubscriptions (topic)
    for sub ∈ subs do
        if match (p, sub) then notify (p, sub)
    return ref
SUBSCRIBE (topic, max_price, keywords)
    s = Subscription (topic, price, keywords)
    ref = storage.dump (s)
    // Update indexes
    t_index.update (s, ref)
    // Match subscription with sub-subscriptions
    subsubs = t_index.getSubSubscriptions (topic)
    for subsub ∈ subsubs do
        if match (s, subsub) then notify (s, subsub)
    return ref
SUB-SUBSCRIBE (topic, keywords)
    ss = SubSubscription (topic, keywords)
    ref = storage.dump (ss)
    // Update indexes
    t_index.update (ss, ref)
    return ref
```

Figure 3. The API and core functionality of the prototype marketplace.

4. Discussion

We are currently in the process of deploying the prototype marketplace as a publisher in a PONS for our university students. The goal of the prototype implementation is to illustrate the idea of an open, stateful publish-subscribe system, and possibly to show the benefit and costs of some of its features. The experiences gained from developing this prototype suggest that these goals are met. The prototype is stateful, and supports both searching for, and subscribing to, product publications. Furthermore, the prototype illustrates the benefit of recursive subscriptions by allowing clients to monitor the demand for certain products. For example, by sub-subscribing to the topic *book* the client is notified every time another client subscribes to the same topic.

The cost of sub-subscriptions is increased system complexity. Sub-subscriptions must be stored and matched against subscriptions. However, note in figure 3 that the storage and processing costs of sub-subscriptions are not significantly different from the costs of regular subscriptions. In addition, assuming that the number of subscriptions in the system is far greater than the number of sub-subscriptions, sub-subscriptions seem to be an affordable facility.

This paper presents the idea of recursive subscriptions, yet the implemented prototype supports only one level of recursion, namely sub-subscriptions. In principle, supporting (i)-level subscriptions, i.e. subscriptions to new $(i-1)$-level subscriptions, is not difficult. Both API matching functions would have to be extended to include this attribute (e.g. *subscribe(topic, level)*. Still, it is unclear if (i)-level subscriptions are meaningful for say $(i >= 2)$. At least this is application dependent. On the other hand, there is little cost in providing this feature. Having such a feature may open doors to development of new applications that can benefit from it.

The prototype was implemented as a topic-based publish-subscribe system. This does not mean that the ideas presented are limited to topic-based systems. A content-based marketplace can support content-based search and content-based subscriptions. In fact, giving access to internal subscriptions becomes even more interesting when the subscriptions are not only simple predicates on a topic name. For example, in a content-based system, it is very interesting for publishers to see how subscribers choose to filter their publications. Another issue with content-based systems, is that the interpretation of sub-subscriptions gets slightly more complicated. Using the terminology of Siena [4], there is a match between a sub-subscription and a subscription if the subscription is *covered* by the sub-subscription. A matching function must implement this *covering relation*.

Although the structure suggests untraditional features like recursive subscriptions or search for subscriptions, all of these features need not be included. The main idea is that the system should be stateful and open. Note however that it is application-specific *how* stateful and *how* open the system should be. As demonstrated by the prototype, a marketplace application may well be implemented without enabling search for subscriptions.

Also, with respect to openness there is a range of solutions. If desirable, a reasonable level of anonymity may be supported even if subscriptions may be both searched for and subscribed to. This is the case, if for instance only the *number* of subscriptions are revealed when a client searches for subscriptions on a given topic. Similarly, the notification of a new subscription need not carry any information about *who* made the subscription.

In the WAIF project, the overall goal is that the Web infrastructure pushes highly relevant data through the PONS towards the user. To reach this goal, the ability to express user interest accurately is the key challenge. Content-based publish-subscribe systems support high expressiveness by supporting more or less sophisticated subscription languages. We argue that there are additional ways of addressing expressiveness. One way is to allow individual users to install personalized code into the system [2]. This way the expressiveness is increased to that of the code programming language. Another way is to provide tools that aid users in stating their interests. In this regard, we see the search mechanism of our stateful publish-subscribe structure as an important mechanism for improving the expressiveness of a publish-subscribe system. As discussed in section 2.1, the search mechanism may provide both overview and details about the content of the system. This way it helps the user to define accurate subscriptions.

5. Summary

This paper presented a new publish-subscribe structure for marketplace applications. The structure is motivated by the observation that academia and industry implement marketplace applications differently. We argue that this mismatch is due to inadequacies of the classic publish-subscribe protocol as a foundation for marketplace applications. In particular, these inadequacies relate to the statelessness, the lack of openness, and the uni-directionality of the publish-subscribe protocol. To address these problems, we propose a stateful and publish-subscribe structure for online marketplaces. The structure promises to bring sellers and buyers closer together, by allowing them to investigate each other's interests and activities in the marketplace. Both a search mechanism and a subscription mechanism are tools at hand for this investigation. A special feature of our structure is the support for recursive subscriptions, i.e. the ability to subscribe to subscriptions. This paper demonstrates the benefit of this feature for marketplace applications, and argues that the cost of this feature is not overwhelming.

6. Acknowledgements

The authors would like to thank Dmitrii Zagorodnov for valuable feedback throughout this work and Kai Marius Pedersen for assisting the implementation of the prototype application.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching Events in a Content-based Subscription System. In *Proceedings of the 18th ACM Sympo-*

sium on Principles of Distributed Computing (PODC '99), pages 53–61. ACM Press, May 1999.

[2] I. M. Arntzen and D. Johansen. A programmable structure for pervasive computing. In *Proceedings of the International Conference on Pervasive Services*, pages 29–38. IEEE Computer Society, 2004.

[3] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, 1993.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a wide-area Event Notification Service. In *Transactions on Computer Systems*, volume 19, pages 332–383. ACM Press, 2001.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. In *Journal on Selected Areas in Communications (JSAC)*, volume 20, pages 100–110. IEEE Computer Society, Oct. 2002.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[7] Y. Jin and R. Strom. Relational subscription middleware for internet-scale publish-subscribe. In *DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM Press, 2003.

[8] D. Johansen, H. Johansen, and R. van Renesse. Environment mobility: moving the desktop around. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 150–154. ACM Press, 2004.

[9] D. Johansen, K. J. Lauvset, R. van Renesse, F. B. Schneider, N. P. Sudmann, and K. Jacobsen. A TACOMA Retrospective. *Software Practice and Experience*, pages 605–619, 2002.

[10] D. Johansen, R. van Renesse, and F. B. Schneider. Operating System Support for Mobile Agents. In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems (HOTOS-V)*, pages 42–45, Orcas Island, WA, May 1995. IEEE Computer Society.

[11] D. Johansen, R. van Renesse, and F. B. Schneider. WAIF: Web of Asynchronous Information Filters. In *A.Schiper er al.(Eds.): Future Directions in Distributed Computing*, pages 81–86. LNCS 2584, Springer, 2003.

[12] K. M. Pedersen. Construction aspects of a subscription-based online marketplace. Technical report, University of Tromsø, Dec. 2004.

[13] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Australian UNIX Users Group Technical Conference*, pages 243–255, 1997.

[14] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.

[15] D. Zagorodnov and D. Johansen. The last hop of global notification delivery to mobile users: accommodating volume limits and device constraints. In *Proceedings of the First International Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet (SIUMI)*. IEEE Computer Society, June 2005. To appear.

[16] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20. ACM Press, 2001.