Publish/Subscribe on the Web at Extreme Speed

João Pereira*
INRIA Rocquencourt
Joao.Pereira@inria.fr

Françoise Fabret
INRIA Rocquencourt
Francoise.Fabret@inria.fr

François Llirbat
INRIA Rocquencourt
Francois.Llirbat@inria.fr

Radu Preotiuc-Pietro†
"Politehnica" University of Bucharest
rady@ss.pub.ro

Kenneth A. Ross†
Columbia University
kar@cs.columbia.edu

Dennis Shasha‡
Courant Institute of Mathematical Sciences
New York University
shasha@cs.nyu.edu

1 Introduction

This demonstration presents *Le Subscribe* an event notification system for the Web.

It is widely accepted that the majority of human information will be on the Web in ten years. As pointed out in [6], besides systems for searching, querying and retrieving information from the Web, there is a need for systems being able to capture the dynamic aspect of the web information by notifying users of interesting events. This functionality is crucial for web users (or applications) who want to exploit highly dynamic web information such as stock markets updates or auctions. A tool that implements this functionality must be scalable and efficient. Indeed, it should manage millions of user demands for notifications (i.e. subscriptions); It should handle high rates of events (several millions per day) and notify the interested users in a short delay. In addition, it should provide a simple and expressive subscription interface and efficiently cope with high volatility of web user demands (new subscriptions, new users and cancellations). Finally, it should facilitate integration of similar kinds of information issued by different publishers (e.g. new auctions coming from distinct auction sites).

The classical approach for query subscription is a mediator system where queries are periodically evaluated against static data. This static approach does not scale for high rate of events and a large number of volatile subscriptions, since it requires the storage of large event histories between two successive computations and requires repeated complex multi-query optimization. In Le Subscribe we adopt a different approach where events are processed on-the-fly to discover matching subscriptions. Our main contributions in Le Subscribe are:

- A semi-structured event model which is well suited for the information published on the Web, and flexible enough to support easy integration of publishers.
- A subscription language which is designed to be simple while supporting the most usual queries on event notifications.
- An efficient matching algorithm for processing events in real time which can handle a large number of volatile subscriptions and supporting high event rates.
- Simple interfaces for publishing and subscribing which enable an easy integration of the system in the *Web*. The system supports both HTTP protocol and Java RMI.

The demonstration consists of an application whose goal is to notify interested users of items put up for auction. These items are captured by Le Subscribe from a set of *auction Web sites*, e.g. *ebay* [5], *amazon* [2] or *yahoo* [16], or can be transmitted to our system using a web interface. Every day, the system has to handle a large number of items. For example, *ebay* publishes about 560000 new auction items per day.

*Founded by "Instituto Superior Técnico" - Technical University of Lisbon and by a JNICT fellowship of Program PRAXIS XXI (Portugal)

†The work of R. Preotiuc-Pietro and K. Ross was performed while visiting INRIA. The work of K. Ross was partly supported by grant 9812014 of the United States National Science Foundation.

‡The work of D. Shasha was partly supported by grant 9531554 of the United States National Science Foundation.

**Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.**

2 Overview of the system

The main characteristics of our system are its event model, its subscription language and its matching algorithm.

2.1 Event Model

The event model supported by Le Subscribe is in the spirit of the LDAP [12] data model. It consists of a set of attributes, and a set of event types. Each attribute has a domain that may be *numeric*, *string*, *enumerated* or *hierarchical*. The *hierarchical* domain is specific to our model: it is an *enumerated* domain where the elements are organized according to a hierarchy. Hierarchical domains are useful to depict categories and sub-categories. For example, a hierarchical domain ranging over furniture categories can be organized in bedroom, dining room, outdoor categories and sub-categories like table, chair, $\cdots$. An event type is always associated with a set of attributes each of them being either mandatory or optional. For example, an item of type *antiques* is described by three mandatory attributes *price*, *period* and *quantity*. An item of type *furniture* can be described using three mandatory attributes: Attributes *price* and *quantity* are in common with the *antiques* event type; Attribute *furniture_category* has a hierarchical domain ranging over furniture categories. Furniture description could be enriched with the optional attribute *material*.

An event instance can be associated with several event types. It is defined by a set of (attribute, set of values) pairs. Among these pairs there is always a pair of the form (*event_type*, T) where *event_type* is a distinguished attribute and T is a set of event types. An event instance definition has to include a non-empty set of values for each attribute that is a mandatory attribute of at least one event type in T, values for other attributes are optional. Let us point out that our model permits publishers to present a given event from several points of view by associating several event types with this event. For example, using event types *antiques* and *furniture*, a publisher can present a table of the Louis XVI period as furniture, as antique or as both. In the last case the event instance definition will associate two event types (*antiques* and *furniture*) to the distinguished attribute and will provide values for attributes *price*, *quantity*, *period* and *furniture_category*, plus possibly for attribute *material*.

2.2 Subscription Language

A *subscription* is defined as a conjunction of elementary predicates. The language provides predicates of the form $X\theta y$, where X is an attribute name, y is a value belonging to the domain of X and θ is a comparison operator. In the case of hierarchical domains $\leq$ and $<$ operators are semantically equivalent to the standard *is a kind of* relationship. In addition, the language supports X *contains* y predicates. Such predicate is true for an event instance e, if value y occurs in the set of values associated with attribute X in e. For example [*(event_type contains antiques) and (furniture_category* $<$ *dining table)*] describes a subscription for all the new auctions concerning any antique which belongs to any sub-category of *dining table*.

An event instance e matches a subscription s if e provides a binding for every attribute occurring in s and all predicates of s are true with respect to this binding. A subscription is satisfied by any matching event instance.

2.3 Matching Algorithm

We have implemented several main memory matching algorithms from the literature (Hanson et al[11], NEONRules[15], Gough et al[9] and Aguilera et al[1]) and have arrived at a synthesis that fits better with the *Web* context. The algorithm is predicate based. Its global optimization strategy exploits predicate redundancy and predicate dependencies among subscriptions to reduce the number of predicate evaluations. Such a strategy is particularly efficient in the *Web* context where a lot of attributes have enumerated domains ranging over a limited number of values. Our matching algorithm can easily be adapted to a multi-processor environment for performance enhancement. Experiments have shown the efficiency of our algorithm even at high event rates, large number of subscription and frequent subscriptions modifications. A detailed description of the matching algorithm, a performance analysis and a comparison with existing matching algorithms can be found in [7, 8].

3 Overview of the demonstration

To show the main features of our system, we have developed an application that uses Le Subscribe. This application notifies interested users of items put up for auction on the *Web*. These items can be extracted from real auction sites (e.g. ebay and amazon) or can be directly published in our system.

The items published by auction sites are represented by *generic_auction* event type. This type has the attributes *description* (string), *category* (hierarchical) and *price* (numeric). The *description* attribute is a set of keywords that describe the item in a summary way. Auction sites classify the items according to their *category*. The existing categories form a hierarchy, e.g. category *Antiques* (which gathers antique items) has the subcategories *Books & Manuscripts*, *Ceramics*, *Furniture*, $\cdots$. The *category* attribute identifies the item's category. Finally, the *price* attribute specifies the starting price of the auction. For the demonstration, we have designed a *Web* site for Le Subscribe which describes the event types, attributes and domains currently defined in the system. We will show that this site can be modified in real time to incorporate new event types, attributes or domains added by publishers. Subscribers browse through this *Web* site as an aid to define their subscriptions.

General architecture

The general architecture of the application is shown in Figure 1. This application consists of three types of entities: *subscribers*, *Le Subscribe* system and *publishers*. Subscribers interact with Le Subscribe through its *subscription* interface. In the demonstration, subscribers send HTML requests to Le Subscribe and receive HTML pages as answer. Publishers publish their events through the publication interface of Le Subscribe. We consider the existence of two kinds of publishers: users publishing events through a *web* browser and auction sites. The former publish events through the HTML interface. The latter publish events via wrapper components (one wrapper per auction site). A wrapper component polls its auction site periodically to get the HTML pages that describe the new items. Next, it parses these pages and translates each item into the event format corresponding to *generic_auction*. Finally, it communicates the events to the system using Java RMI.

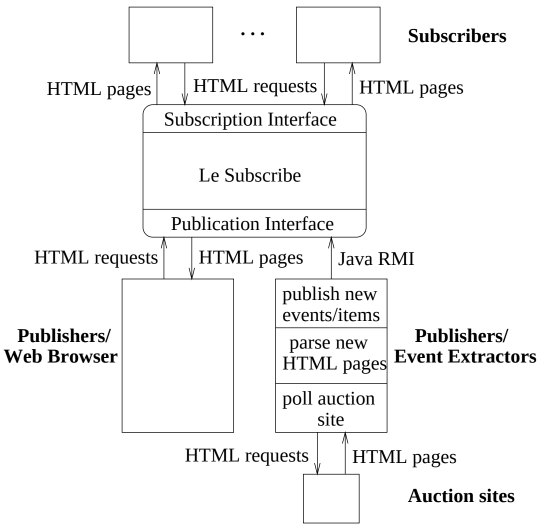

Figure 1: The architecture of the application.

Web subscription interface

Figure 2 shows the *subscription* page used by subscribers interested in events belonging to the *generic_auction* event type. The page has two parts. One part allows to define the predicates and the other to name the subscription[1] and to specify the notification mode. There are three *notifications* modes, *push*, *pull* and *email*. In the *push* mode, for each new notification, the system updates a HTML page that the user has loaded in his browser. In the *pull* mode, the notifications are stored in the system, and are sent on demand to subscribers (in a HTML page). Finally, in the *email* mode, notifications are sent to subscribers by email.

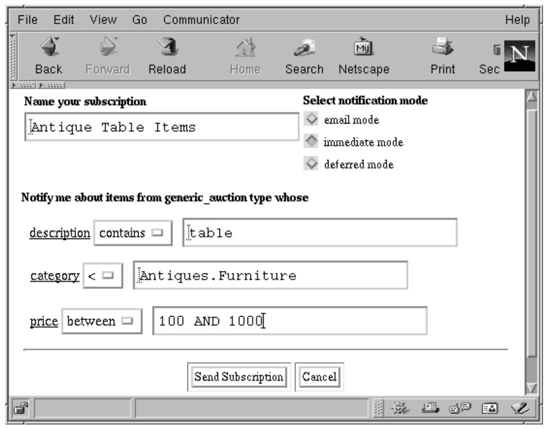

Figure 2: The HTML *subscription* page for *generic_auction* event type.

Demonstration scenario

During the demonstration we will run various scenarios that illustrate some nice features of Le Subscribe:

- The efficiency of our real time matching algorithm will be shown using a scenario with a large number of subscriptions (1000000) and with a high rate of events (400 events per second). If an internet connection is available we will use new auction items extracted on real time from existing *Web* auction sites. We will also subject our system to frequent changes of the set of subscriptions (cancellation and insertion of subscriptions) and show how slightly the system performance is affected.
- Event types and attributes can be defined on-the-fly. They are immediately taken into account by the system. To demonstrate the responsiveness of our system we will show the following scenario: We consider a new auction publisher who is specialized in antiques and wants to offer to interested subscribers a more complete description of antique auction items. To do this the publisher defines (through the publication interface) a new event type, *antiques*, that contains attributes *period*, *style*,$\cdots$. The publisher can immediately publish event instances which include attributes values for both *antiques* and *generic_auction* types. We show that all the subscriptions on *generic_auction* items of category antiques will benefit immediately from the additional information. Moreover, new defined attributes are automatically made visible to subscribers via the *Web* site. This way, subscribers that imperatively want information about the period and style of antiques can subscribe to the *antiques* type.

[1]The name is used afterwards in notifications to refer the subscriptions which were matched by the notified event.

- We also show that our system can take advantage of the filtering capabilities of publishers. To do this we extend our wrapper functionality with a subscription interface where categories of relevant new items can be specified. Thanks to this interface, wrapper components look only for new auctions belonging to the required categories and reduce the number of HTML pages loaded from the remote web sites. By using this interface to push predicates toward the wrapper components, the system behaves as a subscriber with regard to the wrappers. This optimization significantly improves performance when all users are interested in a same (small) set of categories.

4 Related work and conclusions

To our knowledge Le Subscribe is the first proposal for using an event notification (or publish/subscribe) service on the web to deal with highly dynamic Web information. A lot of event notification services have been already developed as middleware for gluing together distributed applications or systems. These systems differentiate from each other by their filtering capabilities, the efficiency of their matching algorithm and additional features like QoS guarantees[2].

There are two kinds of publish/subscribe systems: *subject-based* and *content-based*. In subject-based systems, events are classified by groups and can be filtered only according to their group. Examples of such systems are TIB/Rendezvous[4] and OrbixTalk[13]. Content-based systems are an emerging type of publish/subscribe system where events are filtered according to their attribute values. Le Subscribe is a content-based system. We can cite other content-based systems, like Gryphon[3], NEONet[14], or READY[10] and publish/subscribe mechanisms integrated in commercial DBMS products like Oracle8i, SQL Server 7.0, or Sybase. Compared to subject-based systems, content-based systems offer more subscription expressiveness. The cost of this gain in expressiveness is an increase in the complexity of the matching process: the more sophisticated the constructs, the more complex the matching process. This complexity combined with a large number of subscriptions may severely degrade the matching efficiency. So, systems devoted to support a large number of subscriptions as Le Subscribe system does, have to face a tradeoff between the subscription language sophistication and matching efficiency. In Le Subscribe we designed an expressive language that lends itself to very efficient matching.

The subscription languages of Gryphon and NEONet are quite similar to our language. Their matching algorithm do not exploit predicate redundancy nor dependencies (as Le Subscribe does). The READY system[10] has a more expressive subscription language supporting grouping constructs, compound event matching and event aggregation. Its matching algorithm uses only local optimizations unlike Le Subscribe which intensively exploits global optimization opportunities. Commercial DBMS products use SQL as their subscription language, and these products are designed for contexts where the number of subscriptions is relatively small, as might occur in the context of enterprise application integration.

[2]In Le Subscribe we do not consider these additional features.

References

[1] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *Eighteenth ACM Symposium on Principles of Distributed Computing (PODC '99)*, 1999.

[2] Amazon.com, Inc. *http://www.amazon.com.*

[3] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *International Conference on Distributed Computing Systems*, 1999.

[4] Arvola Chan. Transactional publish/subscribe: The procative multicast of database-changes. In *SIGMOD'98*, page 521, 1998.

[5] eBay Inc. *http://www.ebay.com.*

[6] Phil Bernstein et al. The asilomar report on database research. *ACM Sigmod record*, 27(4), 1998.

[7] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, 2000. http://www-caravel.inria.fr/~pereira/matching.ps.

[8] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *Proc. of Fifth International Conference on Cooperative Information Systems*, 2000.

[9] K J Gough and G Smith. Efficient recognition of events in distributed systems. In *Proceedings of ACSC-18*, 1995.

[10] R. E. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the ready event notification service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop*, 1999.

[11] Eric N. Hanson, Moez Chaabouni, Chang-Ho Kim, and Yu-Wang Wang. A predicate matching algorithm for database rule systems. In *SIGMOD'90*, pages 271–280, 1990.

[12] T. A. Howes, M. C. Smith, and G. S. Good. *Understanding and Deploying LDAP Directory Services.* Macmillan Technical Publishing, 1999.

[13] IONA Technologies. *OrbixTalk http://www.iona.com/products/messaging/index.html.*

[14] New Era of Networks Inc. *NEONet http://www.neonsoft.com/products/NEONet.html.*

[15] New Era of Networks Inc. *NEONRules http://www.neonsoft.com/whitepapers/MQSIRules.html.*

[16] Yahoo! Inc. *http://auctions.yahoo.com.*