The design and implementation of a P2P-based composite event notification system.

Simon Courtenage
Steven Williams

School of Informatics

The Design and Implementation of a P2P-Based Composite Event Notification System

Simon Courtenage and Steven Williams
Cavendish School of Computer Science
University of Westminster
London, W1W 6UW, UK
courtes@wmin.ac.uk

Abstract

The development of large, open, and heterogeneous distributed systems is becoming increasingly dependent on event services to bind together the components of an application in such a way that they are able to react to changes in other components.

One way to distribute event notifications around a distributed environment is to use content-based publish/subscribe communication. Such a system mediates between publishers of information and subscribers who sign up to receive information by routing messages across the network from their source to the point of subscription using the message content and the client subscriptions. Although content-based publish/subscribe has been used successfully to develop simple event notification systems, in which events are routed through from external publisher to external client, more complex systems are possible that create new events, known as composites, based on the detection of patterns of events.

Composite event notification, however, poses a number of challenges, including network management and network routing. In this paper, we discuss the design and implementation of a composite event notification system over a Chord-based peer-to-peer network using JXTA, and how we have addressed these challenges.

1. Introduction

Publish/Subscribe systems [6] form an important communications paradigm in distributed systems, one in which servers (or producers of messages) are decoupled from clients (or consumers) by the network. Instead of clients contacting servers directly to request services or information, clients register a subscription with the network, via a local access point, to receive messages satisfying certain criteria. Servers publish information onto the network, without knowing who will receive it, and the network (via its servers or brokers) undertakes to route messages to the appropriate clients based on the set of subscriptions currently in effect.

Within the publish/subscribe paradigm, there are many different classifications of system based on how a subscriber makes a subscription. Traditional publish/subscribe systems create channels, groups or topics, sometimes hierarchial, under which messages may be classified. In this case, a subscription is simply a statement of the channel, group, or topic that a user wants to receive messages from.

Another, more radical, approach, known as content-based publish/subscribe, allows the subscriber to specify the kind of message content they want to receive [2]. Content-based routing relies on the subscriber being able to create a subscription using criteria that specify conditions over the message content, typically in the form of predicates over the elements of the event notification structure. The advantage of this style of publish/subscribe over more conventional systems is the far greater flexibility that is permitted in creating subscriptions. Subscribers are in effect allowed to create their own message groupings rather than simply sign up to predefined ones. Typical applications of this form of publish/subscribe are event notification services, such as Elvin [11], Siena [1] [2], Gryphon [13], and Hermes [9].

One of the key problems of content-based publish/subscribe concerns message routing. As messages appear on the network from publishers, individual servers in the publish/subscribe network must examine the message content in order to make routing decisions. The message content must be compared against what is known about current subscriptions in order to decide which server the message should be routed to next. In order for this to happen, both publication and subscription must meet somewhere in the network - hence the typical use of broker or rendezvous servers in content-based publish/subscribe sys-

Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)
1550-445X/06 $20.00 © 2006 IEEE

tems. Most research in this area has been on the routing of messages from external sources to subscribers, for example, Elvin [11] and Hermes [9]. However, the issue of how to route messages representing composite event notifications, i.e., messages that are raised internally as a result of the occurrence of patterns of events is more complicated, because of the complex re-configuration of routing paths within the network needed to co-ordinate the routing of component events of a pattern. A proposal for automatically generating event dissemination trees for configuring routing paths for composite event notification notifications was presented in [4]. This took the form of a simple functional event specification language whose expression structures could be automatically analyzed to gather the information necessary to reconfigure the network as and when a new composite event had to be detected. However, that work left open how the information derivable from a subscription expression should be used.

In this paper, we describe the design and implementation of a distributed composite event notification system that uses the functional event specification language from [4] and which successfully solves the complex challenges of network configuration and routing composite events. The system is a content-based publish/subscribe network that uses a DHT-based peer-to-peer system to carry out network configuration and the JXTA open P2P framework for event routing. We show how the FEL (Functional Event Language) architecture is dynamically configured to the requirements for routing composite event notifications, as new client subscriptions are made, using the hashing scheme provided by the underlying DHT-based P2P system, and how the JXTA pipe abstractions simplify routing of events.

2. Event Specification in FEL

An event is defined as the occurrence in a system of some situation of interest, usually one which requires an automatic response by the system [16]. In active databases [7] [3], typical events are inserts or deletions from tables or extents. Detection of events permits the specification of actions to be carried out automatically by the system when they occur.

In distributed systems, events are used to create an asynchronous style of communication. In the client-server model of distributed communication, client and server processes communicate directly using explicit addresses, with the server responding synchronously to client requests. Distributed event-based systems allow an asynchronous and decoupled communication model, known as publish/subscribe. Occurrences of events are represented by event notifications, in the form of messages. Clients, or subscribers, make known to the system what messages they are interested in receiving, and the system undertakes to send them all event notifications that match their interests asynchronously, i.e., as and when they arise.

There are two general types of events: primitive and composite. Primitive (or atomic) events are those happenings that are considered atomic and instantaneous. They occur either as the result of some internal state of affairs or some external interaction with the system. The actual definition of primitive events depends upon the application domain. For example, in a real-time data application, atomic events might be the publication of a particular stock market transaction or of the result of a football match.

We define composite events as events that arise as the result of the occurrence of other events, in some order or pattern. For example, we might want to detect whenever an event of type T2 occurs directly after we have seen the occurrence of an event of type T1 (a *sequence* of T1 and T2), or whenever either a T1 or a T2 type event occurs (a *disjunction* of T1 and T2). The detection of such patterns of *component events* can be represented by raising a composite event notification that represents the pattern occurrence (including information about the pattern).

The heart of our publish/subscribe system is the subscription language FEL (Functional Event Language) [4]. FEL is a simple but formal language for making subscriptions in a content-based publish/subscribe network. It is declarative and strongly-typed. Composite event specifications in the functional event language take the form of functions, which expect a number of arguments. The arguments to the function are the component events of a composite event. Once these have been fully supplied, the composite event value is produced. Being a functional language, functions can be partially applied to their arguments through currying. This models the way that composite events progress towards their occurrence as component events occur over time. In addition, the structure of the functional expression that specifies a composite event provides vital information about how events should be routed through the distributed system (see [4] for full details).

3. FEL Architecture

Content-based publish/subscribe systems work by matching up subscriptions with publications. Typically, as is the case with FEL, subscriptions are in terms of types of events plus some predicates over the values of those types. Matching subscriptions to publications means that subscriptions and publications for particular event types must meet at a certain point in the system so that they can be compared. For this reason, content-based publish/subscribe systems use a broker-oriented communications architecture, where each broker is a server in the network and undertakes to receive publications and subscriptions for a particular event

type and match them. If a broker matches a subscription to a publication, it then sends the publication onto the client making the subscription.

Most content-based publish/subscribe systems deal with primitive events. In this case, routing a message from a publisher to a subscriber is reduced to finding the right broker node for a publication or a subscription. When a publication arrives at a broker, it can be matched against known subscriptions and, if successfully matched, sent by the broker to the appropriate subscriber. In such systems, new event types are created by publishers who advertise new events they wish to publish.

One of the key problems to be solved in a composite event notification system lies in the changes needed to the topology of the network as new subscriptions create requirements for new composite events. This problem is different to that posed by publications of new types of primitive events. Advertisement of a new type of primitive event requires only the election of a new broker, who may be chosen by a simple hashing of the event type to a broker identifier (as is the case in Hermes [9], for example). Once this has been done, any subscription for publications of that type are directed to the new broker by the same hashing scheme.

New composite events, however, are subscriber-driven, not publisher-driven. New composite events may be required, even if there are no new types of events being published, simply because subscribers wish to combine them into different patterns. This asynchronous creation of new event types to meet subscriber demand creates new problems for management of the publish/subscribe network topology.

As an example of this, Figure 1 shows a composite event notification system comprised of two publishers P1 and P2, publishing events of type A and type B respectively, a broker node B1 that receives events of these types and detects sequences of type A events and type B events, and a subscriber S1 who has subscribed to receive composite events which represent sequences of type A and type B events.

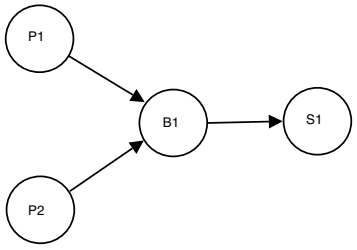

Figure 1. Composite Event Notification with Single Broker

If a new subscriber S2 wishes to be notified about disjunctions of A and B events, then this creates a new composite event, and hence a new broker is required, as Figure 2 shows.

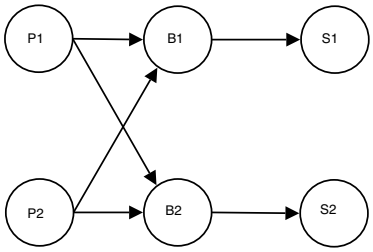

Figure 2. Composite Event Notification with Multiple Brokers

It could be argued that a new broker is unnecessary if the only difference between B1 and B2 is in the pattern of A and B events they are detecting. However, if the broker is located in the network by hashing the event type to a node id, we cannot guarantee that a composite event type representing sequences of A and B will hash to the same broker id as a disjunction of A and B. Also, the algorithms or state machines used to detect patterns will be different for different composite events even if the component events are the same. For these reasons, therefore, it is simpler to split composite event detection across different brokers, regardless of which component events are involved.

However, adopting the approach of new brokers for new composite event types, in response to new subscriptions creates two (related) problems for the organization of the event notification network.

Firstly, publishers of a particular type of event no longer know, as a result of hashing the primitive event type, the identities of all the brokers to whom they should send event notifications - this applies to publishers of primitive events who are outside the publish/subscribe network and also to servers within the network acting as composite event detectors and publishers. This is because the identity of an event detector broker is found by hashing the event type, but in an event notification system with new composite events being created dynamically, event notification publishers do not know the identities of new composite events in order to use this information to derive broker identifiers through the hashing scheme (unless we choose to broadcast this information throughout the network).

Secondly, the new locations to which publishers should send their events come into existence asynchronously with respect to the publishing of events, since they are subscriber-driven. In an event notification network that publishes only primitive events (i.e,. simply routes external events to subscribers), the identity of the broker is known by a publisher when it joins the network - if a broker for the event type does not exist, then the network elects a broker

to handle the new event type at the point when the publisher joins. In an environment with dynamically-created composite events, however, new brokers that want to receive notifications of a particular event may be started at any point after a publisher of that event type has joined the network.

4. Decentralized Management in FEL using Meteor

The solution proposed by FEL to these problems was to make explicit in the structure of the subscription expression all of the high-level routing information needed to configure the publish/subcribe network. The key challenge, however, in the design of the FEL system architecture was how to use the information provided by the subscriber to configure the network, without imposing scalability constraints on the network itself. For example, an initial proposal was to overlay the network of event detector servers with a static layer of management servers that would react to new subscriptions by allocating unused event detector servers to new composite events and create the necessary connections between them. However, the fact that this management layer was static created its own scalability problems, since the required capacity of the management layer depended on an *a priori* assessment of the likely traffic on the publish/subscribe network in terms of the number of different events to be detected. If FEL was to be deployed in a wide-area, internet-scale environment, for example, this would be highly impractical.

Hence, we decided to investigate how to decentralize the management processes in FEL. The principal management activities involved identifying event detector brokers in the publish/subscribe network. The approach we took was to build FEL on top of a DHT-based peer-to-peer network. Since all nodes in a DHT-based P2P system have knowledge of the hashing scheme in use (in order to identify other nodes in the network), implementing the management process as a hashing scheme results in decentralization, since any node in the network can carry out management activities by using the P2P hashing scheme.

The P2P architecture[1] we used was Chord [5] [12]. Chord is a popular distributed P2P service with a single operation: node lookup based on Distributed Hash Tables (DHTs). According to the Chord protocol, each node in the network is assigned a Chord id, derived from the hashed value of its IP address (or some other address value). The topology of the Chord P2P network is then arranged as a ring of nodes in ascending order of Chord id value, with each node maintaining a *finger table* containing the addresses of nodes further round the ring at certain intervals as an aid to routing (see [12] for full details). Given a particular hash key, the Chord architecture allows fast and efficient lookup of the P2P node associated with that particular key. For our purposes, we used (with some modifications) Meteor [8], which implements Chord in Java using the JXTA framework.

The architecture of the Meteor-based version of FEL is illustrated in Figure 3. In order to achieve decentralization, the nodes in the Meteor-based FEL system take on multiple roles, acting as service containers. Each node can, by spawning separate threads, act as the local access point for an event notification to be published onto the network and for a client subscription to enter the system, as well as act an event detector for a particular event type. Each particular service, whether it be, for example, the local access point for publication of a particular atomic event or a composite event detector, is also given an identifier based on the hashing of the event type associated with the service. This hash id allows the service to be placed on a particular node on the Chord ring, i.e., that node whose Chord id is the closest successor to the service identifier.

Figure 3 illustrates this by showing four nodes in a 256-node network with Chord ids 14, 48, 97, 167. Node 14, as shown, is a container for two services: a composite event detector (CED) and an atomic event detector (AED), i.e., a publisher of primitive events. Both the composite and atomic event types have been hashed to associate values with the CED and AED, which are shown as 10 and 191 respectively, and are therefore located on node 14, which is the closest successor to the service identifiers. This allows us to automatically balance the load around the network, for example, as nodes enter and leave the network. For example, if a new node joins with Chord id 255, then the AED (with an id of 191) can be migrated from node 14 to node 255.

When a new user subscription is first created, it is sent to a node based on a hash of the composite event type represented by the subscription. If the node that receives the subscription does not currently contain an event detector for the composite event, then one is created on the node. Similarly, the component event types of the composite event are then extracted and also hashed to locate their container nodes on the P2P network. This process is followed until atomic event detectors are reached or until a composite event detector representing a subtree of the original subscription is found (created as part of a previous subscription), resulting in a traversal of the P2P nodes representing the composite subscription expression graph.

As each subtree of the expression graph is completed

[1] As an aside, we note that in general peer-to-peer systems, it is quite possible to have cycles in the network. For example, a search request in a file-sharing P2P network can return to its originating node via a cyclical route, requiring cycle detection or time-to-live counters. This does not occur in FEL because the FEL subscription language is strongly-typed and does not allow infinite types, i.e., that a subscription expression for a composite event cannot contain itself.

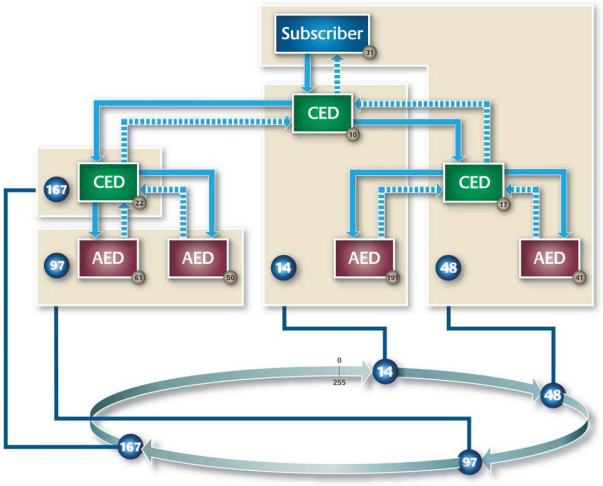

Figure 3. FEL over Meteor

(including creation of new event detectors where necessary), acknowledgement messages are returned to indicate completion. The event detector discovery and acknowledgement messages are carried by the underlying Meteor layer of FEL, through a modified Meteor API that allows application-generated messages to be injected into the P2P network. However, the subscription is finalized at the application layer by further, separate use of JXTA. Once an event detector is ready, and before it sends acknowledgement messages to its parent (i.e., its subtree of the subscription graph has been fully configured), it creates a JXTA output pipe to publish its event notifications if one does not already exist. On receipt of the acknowledge message, the parent event detector binds to the JXTA output pipe in order to receive event notifications.

The use of JXTA, separate from the use of JXTA in the Meteor implementation of Chord, for event publications has the advantage that the P2P network is not congested by potentially heavy volumes of event notifications. Instead, the potentially expensive routing of messages around the Chord ring are reserved for the much lighter load of messages implementing new routing configurations. Direct use of JXTA pipes for event publication also has the advantage of solving the problem posed by the asynchronous, subscriber-driven creation of new composite event detector nodes mentioned in Section 2. A JXTA output pipe can be used to implement a form of loosely-coupled multi-cast, where one source can broadcast to potentially many clients, without needing to know which clients are receiving the outputs written to the pipe. This feature allows composite event detectors to configure themselves to receive component event notifications asynchronously without action on the part of the component event detector. To implement discovery of which pipe a composite event detector should bind to in order to receive component events, we used the JXTA discovery service. A composite event detector needs to know only the type of the component event to use the JXTA discovery service to locate the pipe and to bind to it. Using JXTA in this way also has the further advantage that if an event detector is migrated to another node (as nodes join or leave the underlying P2P network), the routing of events between event detectors through JXTA pipes is unaffected.

5. Related Work

The work most directly related to ours is Hermes [9]. Hermes is an attempt to develop a middleware-based content-based event notification system, and delivers primitive events using a broker-oriented P2P-based architecture. In the composite event extension of Hermes [10], composite event detectors are external to the event notification system and have to be manually (and statically) placed at strategic positions around the system, subscribing to receive component events and, when a particular pattern is detected, injecting the composite event back into the system disguised as a primitive event. Although mobile CEDs are discussed, they require a logical overlay layer is required to move CEDs around the system, requiring as well additional administrative services and potentially complex interactions between event brokers to decide where they should be placed. Our work avoids all the problems of complexity, manual intervention, and administration posed by this approach by utilizing the hashing scheme of the underlying P2P network to perform automatic load balancing, service placement and dynamic service migration, as well as the JXTA technology to connect event brokers.

The peer-to-peer protocol Chord has been used before in the implementation of publish/subscribe systems: for example, content-based publish/subscribe systems using Chord is described in Triantafillou *et al* [15] and Terpstra *et al* [14]. The primary goals of [14] are the robustness of routing primitive events over a content-based publish/subscribe network and the implementation of a filtering strategy on top of a DHT-based P2P system. In [15], the concern is with how range predicates (more complex filters on content, such as *less-than* or *greater-than*) can be implemented for primitive events in DHT-based P2P systems such as Chord where the use of hashing to locate nodes makes support of filters other than equality difficult.

6. Conclusion and Further Work

We have shown how a DHT-based P2P architecture and the use of the JXTA P2P open framework can solve the complex problems involved in the dynamic configuration of a composite event notification system that uses content-based routing of events. The DHT-based P2P layer allows

us to carry out management activities in the network in a completely decentralized and automatic way, while the use of JXTA pipes for routing messages allows event producers to be decoupled from those who wish to asynchronously sign up to receive events. The use of JXTA pipes also greatly simplifies the management activities, for example, of load balancing; no reconfiguration of routing paths is needed as services are migrated, for example, since the connection between event brokers is at the abstract level of JXTA pipe identifiers.

The Meteor-based version of FEL has been used in a small-scale evaluation exercise, using a Chord ring of 5-10 nodes, and has been shown to work well. Composite events can be created and implemented in the network, with nested composite events to create subscription expression trees of arbitrary depth. Without manual intervention or static administration, composite events can be detected as patterns of component events occur and be routed onto subscribers. The network also copes with dynamic creation of new composite events as new subscriptions are made, as well as with dynamic joining and leaving of nodes from the network (to achieve this, we extended the Meteor implementation to allow a node to handover its subscriptions etc. before gracefully exiting the network).

The advantage of Chord for FEL is principally that management of the publish/subscribe network comes "for free" as a result of the performance of the hashing scheme used in the underlying DHT-based P2P network. The performance of FEL, therefore, is dependent on the performance of the hashing scheme. In Chord, the distribution of the load around the network ring has come under investigation, which may affect the performance of the overlying publish/subscribe system. FEL, however, is not tied to using Chord, but can make use of any DHT-based P2P architecture.

Future work will concentrate on large-scale evaluation of FEL in order to assess its performance with large numbers of subscriptions and event notification volumes. This work is currently underway.

7. Acknowledgements

We gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council (under EPSRC Grant GR/S01573/01) for this work.

Our thanks to Nick Stenning for producing the diagram in Figure 3.

References

[1] A. Carzaniga, D. Rosenblum, and A. Wolf. Interfaces and algorithms for a wide-area event notification service, Oct. 1999.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[3] S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. *Data and Knowledge Engineering*, 14(1):1–26, November 1994.

[4] S. A. Courtenage. Specifying and detecting composite events in content-based publish/subscribe systems. In *1st International Workshop on Discrete Event-Based Systems*, jun 2002.

[5] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with Chord, a distributed lookup service. In IEEE, editor, *Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII). May 20–23, 2001, Schloss Elmau, Germany*, pages 81–86, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2001. IEEE Computer Society Press.

[6] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe, 2001.

[7] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In *Proceedings of the 18th International Conference on Very Large Databases*, 1992.

[8] N. Jiang, C. Schmidt, V. Matossian, and M. Parashar. Enabling applications in sensor-based pervasive environments. In *Proceedings of the 1st Workshop on Broadband Advanced Sensor Networks (BaseNets 2004)*, San Jose, USA, October 2004.

[9] P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proc. of the 1st Int. Workshop on Distributed Event-Based Systems (DEBS'02)*, pages 611–618, Vienna, Austria, July 2002.

[10] P. R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *IEEE Network*, 18(1):44–55, 2004.

[11] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUUG'97*, 1997.

[12] I. Stoica, R. Morris, D. Karger, M. Kaashock, and H. Balakrishman. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Proceedings of the ACM SIGCOMM*, pages 149–160, Aug. 2001.

[13] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering, 1998.

[14] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM Press, 2003.

[15] P. Triantafillou and I. Aekaterinidis. Content-based publish/subscribe over structured p2p networks. In *1st International Workshop on Discrete Event-Based Systems*, May 2004.

[16] D. Zimmer and R. Unland. The formal foundation of the semantics of complex events in active database management systems, 1997.