Role-Based Access Control for Publish/Subscribe Middleware Architectures

András Belokosztolszki David M. Eyers Peter R. Pietzuch

Jean Bacon Ken Moody

University of Cambridge Computer Laboratory
JJ Thomson Avenue
Cambridge CB3 0FD, United Kingdom
{firstname.lastname}@cl.cam.ac.uk

ABSTRACT

Research into publish/subscribe messaging has so far done little to propose architectures for the support of access control, yet this will be an increasingly critical requirement as systems move to Internet-scale. This paper discusses the general requirements of publish/subscribe systems with access control. We then present our specific integration of OASIS role-based access control into the Hermes publish/subscribe middleware platform. Our system supports many advanced features, such as the ability to work within a network where nodes are attributed different levels of trust, and employs a variety of access restriction methods which balance expressiveness with the content-based routing optimisations available. We illustrate our achievements by discussing an application scenario in which our system will be of particular use.

Keywords

publish/subscribe, role-based access control, restriction of advertisements/subscriptions, broker trust

1. INTRODUCTION

The Opera Research Group at the University of Cambridge has worked in both publish/subscribe (pub/sub) middleware [1, 7], and in Role-Based Access Control (RBAC) [3, 8]. We believe that combining these two technologies would be a useful contribution to pub/sub system research.

Publish/subscribe technology has to date focused on event filtering, efficient event routing, delivery semantics, and event composition. Access control has been largely neglected. Access control is an important requirement when customers are expected to pay for publish/subscribe services. A simplistic approach would be to run the entire pub/sub system over a secure network infrastructure (for example using Netscape's Secure Socket Layers SSL). However, this does not allow access control policy to be expressed on a subscription or publication basis, and it lacks differential treatment of various nodes in the network by assuming complete trust. It would not be practical for Internet-scale applications with nodes of varying reliability and trust. Further, this approach does not allow the policy governing the privileges of nodes to evolve based on experience.

To address these issues, we have extended our pub/sub system to support a form of role-based access control, and use roles as the basis for security-oriented filtering of both publication and subscription of events within the network. We also support the efficient, dynamic update of pub/sub RBAC policy.

To our knowledge, this paper presents the first complete architecture for policy-based access control in publish/subscribe research.

This paper is organised as follows. Section 2 describes the aims and contributions of our research. Section 3 introduces the fundamental pub/sub and RBAC concepts used. We describe a general architecture for pub/sub access control in Section 4. We extend this architecture to include trust management for brokers in Section 5. Details of our implementation using the specific OASIS [3] and Hermes [7] technologies are presented in Section 6. Some common types of access control policy, based on what we expect to be useful applications, are presented in Section 7. Section 8 describes some of the future research directions we shall pursue. We discuss research closely related to our work in Section 9. Finally, Section 10 concludes this paper, summarising our research contributions.

2. RESEARCH GOALS

The traditional goals of security cannot easily be reconciled with the philosophy of pub/sub. The conventional use of pub/sub is in the efficient dissemination of information through some network of intermediaries (referred to as *event brokers*). This communication paradigm allows the publishers and the subscribers to be loosely-coupled, thus avoiding the need for them to maintain state about all the other publishers and subscribers with which they might interact. Particular registered *event types* exist regardless of any changes to the number of publishers and subscribers interested in them.

Our goal was to design a system in which security is managed within the pub/sub middleware. Each registered event type in the system must have a defined *owner*; we chose to certify them via the public key infrastructure (specifically X.509 certificates [5]) used within our current OASIS implementation. These owners are trusted, and have control over the policy used to access their event type. They need not be a publisher of that type.

Most importantly, we wanted our security considerations to add only a few extra steps for those aiming to publish or subscribe to some particular event type. Indeed, our basic approach allows publishers and subscribers to be completely agnostic with regard to the presence of access control in the pub/sub network. In a standard pub/sub system, publications are of much higher volume than subscription or advertisements. We therefore tried to keep the overhead due to access control checks per event publication as low as possible. This is achievable because we trust the event brokers in the system (partial trust is explored in Sect. 5), and thus access-control state can be distributed throughout the pub/sub middleware. Thus, security checks can be entirely handled by the event brokers directly adjacent to publishers or subscribers.

3. BACKGROUND

This section provides a quick overview of the technology used within our research, including the Hermes publish/-subscribe middleware, and the OASIS role-based access control system. It introduces fundamental concepts that will be used throughout the paper.

3.1 Publish/Subscribe

Publish/subscribe systems quantise data transmission into *events* and disseminate events from producers to interested consumers. A distributed pub/sub system will usually support content-based routing and filtering, and it consists of a network of subscribers and publishers.

3.1.1 Hermes

Hermes is an event-based middleware developed within the University of Cambridge. It has strong ties to the earlier Cambridge Event Architecture. A system built using Hermes [7] consists of two types of components: *event brokers* and *event clients*. Event brokers form the overlay broker network using peer-to-peer routing techniques and implement all the pub/sub functionality. Event clients can be either *publishers* or *subscribers*. They are light-weight and need to connect to an event broker before using any of the services provided by Hermes. An event broker that maintains client connections is called a *local event broker* and can be *publisher-hosting*, *subscriber-hosting*, or both. Otherwise it is called an *intermediate event broker*. Any broker in the system can potentially become a local broker.

Hermes supports proper event typing so that every published event is an instance of an *event type*. An event type has an *event type name*, a list of *typed event attributes*, and an *event type owner*. All event types are organised in an inheritance hierarchy. Event type definitions are kept in a (logical) event type repository and events are type-checked at publication time by Hermes. The content-based routing algorithm used by Hermes supports *type-based* subscriptions that filter events according to their event type only and *type- and attribute-based* subscriptions that support content-based filters over the event attributes as well. In addition to subscriptions, *advertisements* are sent by publishers to express their desire to publish particular event types. They help to set up routing state in the form of an event dissemination tree for the future flows of events. Advertisements and subscriptions are forced to meet in the network at dedicated *rendezvous nodes* that are set up on per event type basis.

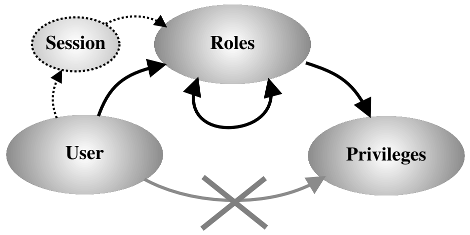

Figure 1: The basic Role-Based Access Control components.

3.2 Role-Based Access Control

Role-Based Access Control (RBAC) introduces *roles* as an abstraction between *principals* (users) and *privileges*. The arc crossed out in Fig. 1 is intended to symbolise that users are never directly assigned privileges. This grouping eases administration since roles often intuitively collect and relate both users and the protected rights in a system. Moreover the access control policy itself becomes more loosely coupled from the software being protected, allowing for policy changes to be made dynamically.

There are usually two stages to acquiring privileges in an RBAC system; the first is authentication as some user. RBAC systems will then require that you request a role to activate. Note that some RBAC designs introduce the concept of a user *session* in which to collect currently active roles. Thus sessions relate a set of active roles for a user, and facilitate deactivating all roles in this set simultaneously when the user finishes their session. As indicated by the reflexive arrow on the 'roles' in Fig. 1, RBAC systems may allow the activation of certain roles to be a prerequisite for further role activations.

3.2.1 OASIS

The Open Architecture for Secure Interworking Services (OASIS: [2, 3, 4]), is an architecture for role-based access control in distributed environments. It supports parameterised roles, giving the ability to add, and then make decisions based on, attributes which are relevant to access control attached to each role instance. For example, the role paidUpSubscriber may have an attribute rate to indicate the level of subscription payment. OASIS provides a simple but expressive policy language based on rules which list the prerequisites for entry into a given role, or acquisition of a privilege. Another OASIS RBAC extension is the concept of *appointments*; persistent qualifications or capabilities with a session-independent lifetime. Finally, OASIS rules may contain conditions that are monitored throughout the time active roles are dependent on them. These *membership con-*

ditions are the basis of our fast revocation; if they become false a user will lose the privileges dependent on them immediately.

4. PUB/SUB ACCESS CONTROL

As described above, the Hermes pub/sub infrastructure consists of a network of event brokers, which connect publishers to subscribers. This section provides an overview of our architecture, discussing security issues for publishers and subscribers. Note that until Section 5 we assume that the broker network can be trusted. We employ the following types of OASIS policy:

Broker-client connection policy determines acceptable initial authentications of clients to their local brokers.

Type management policy specifies via roles which principal is privileged to create, modify and remove event types from the pub/sub system.

Advertisement policy and **subscription policy** dictates who is allowed to send or receive events for each particular event type.

4.1 Publishers and Subscribers

From the perspective of our access control, publishers and subscribers are in most ways equivalent, apart from the privileges they are attempting to acquire. In either case, their authentication to their local broker along with their presented credentials will be used to determine *what events they are permitted to publish and subscribe to* by their local broker, based on the policy provided by the owner of that particular event type.

The design of each application's access control policy will determine how to minimise the risk of damage to a system by malicious participants. Two broad strategies are provided by OASIS for the authorisation of publishers and subscribers to send and receive events, one based on authentication, and the other based on capabilities. OASIS can easily be extended to allow access control decisions to be based on other factors, such as location, if that is relevant to any particular application. Authentication provides fine-grained support for revocation, while capabilities reduce administrative complexity by allowing capability certificates to be distributed without requiring modification of OASIS policy. Of course these approaches are not mutually exclusive.

For the rest of this section, we treat both publishers and subscribers as clients of the pub/sub system, noting that the access control state for these clients is maintained in their local broker. This policy is in the form of OASIS RBAC rules, as specified by the event type owner. We discuss how the pub/sub system allows us to efficiently scale dissemination of access-control policy in Section 6.

In Figure 2 the steps of an access controlled advertisement are shown. The publisher advertises an event type and at the same time provides its credentials to a local broker. Based on these credentials, access is either granted, thus the advertisement is accepted either fully or partially, or access is denied, thus the advertisement is rejected. If the advertisement is partially accepted then a *restriction* is created for this publisher and event type. Such a restriction will limit the advertisement, e.g. to a particular sub-type in the event type hierarchy, as discussed in the next section. Note that the small 'R's represent restrictions.

Similarly in the case of a subscription, a subscriber can

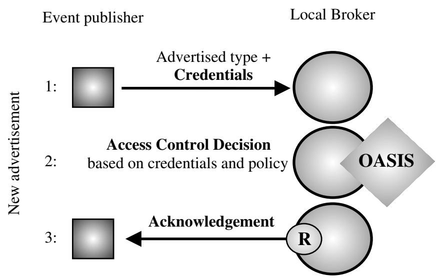

Figure 2: Advertisement with access control.

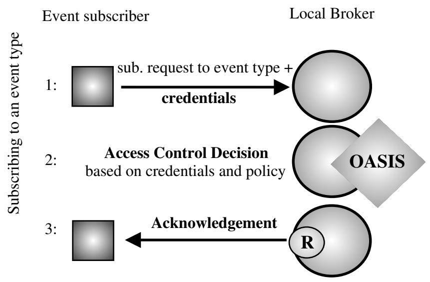

Figure 3: Subscription with access control.

notify a local broker regarding its interest in an event type. Once again, an access control decision will be made based on the subscriber's credentials and the broker's policy. This is illustrated in Figure 3.

Every time an event is published the event goes through the relevant local broker. If there is a restriction for the publisher, then it will be enforced at this broker. This is shown in Figure 4.

Note that by default clients will not be aware whether restrictions are applied to them – an alternative, complementary scheme is discussed in Section 8.

An example pub/sub infrastructure is shown in Figure 5. All of the publishers and subscribers sit outside the internal broker network (whose interconnections are shown expanded within the network 'cloud'), connected to their local brokers via the displayed restriction points. Note that intermediate event brokers are not necessarily OASIS-aware, but local event brokers must be, since all policy checks are carried out by them.

4.2 Policy-based restriction of advertisements and subscriptions

It is important to note that the OASIS policy defining access control for an event type is only employed at the time of a client's advertisement or subscription. Depending on what roles are activated by this client, the local broker will use OASIS policy to determine whether the client's advertisement or subscription is too powerful and thus exceeds privilege. If so, the local event broker will admit the original advertisement/subscription only after having limited it using an appropriate restriction.

In simple cases, this restriction is a predicate that is based entirely on the event type. There will be an OASIS privilege associated with advertisement and one with subscription for each event type. In more complex cases, access control decisions will be based on predicate evaluation, and may also take into account event attributes. The predicate here will indicate whether any published event should be

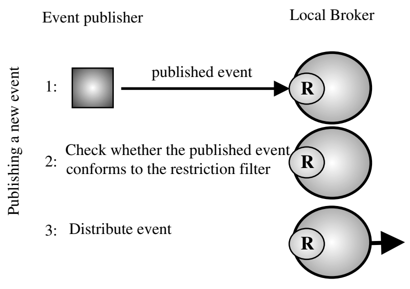

Figure 4: Publishing an event.

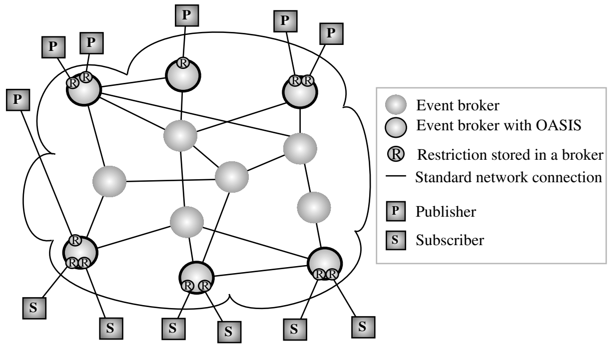

Figure 5: General architecture.

blocked. The related OASIS privilege must contain the following information:

Predicate name. This predicate will return a true or false answer based on the parameters it is given. It could be a simple logical formula, but might involve complex computation, possibly interacting with external services.

Event type. All restrictions are associated with an event type.

Parameter Binding. We must specify how the predicate parameters are bound to attributes of the given event type or constants.

Attribute-based event filtering in the pub/sub system also uses predicate computations, but the set of predicates provided is much more limited – each predicate must have well understood semantics suitable for distributed filtering and aggregation throughout the pub/sub network. Thus we can specify three different classes of predicates that can be used for access control in our pub/sub architecture:

(1) Generic Non-Optimised Restriction Predicates. The most generic case that we use in our model is when a predicate is handled as a *black box*. In this case it is not understood by the pub/sub system, however it provides all necessary expressiveness required by our OASIS access control evaluations.

Take for example size constraints for event publication. The black box predicate can check constraints placed on the size of published events, even though the events themselves might not contain information about their size. Black box predicates might involve evaluation of external information, for example a predicate for a tennis match event could filter the events that involve only players who are in the top 50 of the league table, where the league table is stored in a globally available database.

A disadvantage of these predicates is that the event middleware cannot optimise event filtering by distributing their computation throughout the broker network.

(2) Publish/Subscribe Restriction Predicates. The second class of restriction occurs when the set of access control restriction predicates is precisely the set supported by the pub/sub system. We can thus take advantage of the support for handling subscriptions in the event-based middleware, at the cost of reduced predicate expressiveness.

Restrictions in this class make use of the event type hierarchy. They may also utilise attribute-based filtering provided by the pub/sub system. If a client attempts to subscribe to a type that they do not have authorisation to access, the system will check if they are instead authorised to access any of this event's sub-types. If so, their original subscription is transformed into a potentially different effective subscription. In the case of attribute-based filtering, the pub/sub system will efficiently filter events by pushing subscriptions as far as possible up the event dissemination tree, closer to the publishers.

(3) Hybrid Schemes. The final class involves using information about the predicate itself to compute a subscription understood by the pub/sub system, as well as operating per-event filtering at the local broker based on the full expressiveness of general predicates.

For example, say a subscription restriction prevents a client from receiving any event that has an attribute value other than an even number between 1 and 10. The pub/sub system is capable of filtering attribute values except those from 0 to 10. It is not capable of filtering all but even numbers, however. Thus a per-event filter in the manner of the first class above operates in tandem with a pub/sub filter predicate.

The most efficient policy evaluation will occur when all predicates are understood by the pub/sub system and can be compiled into restricted advertisements and subscriptions. Merely modifying advertisements and subscriptions does not add any overhead to the pub/sub system during event flow, because the logic to manage advertisements and subscriptions is already provided by the pub/sub infrastructure in order to make routing decisions.

Our access control framework also introduces a new event type for the sake of allowing event type owners to modify the policy related their event types. *Policy evolution events* are meta-events published by event type owners which must reach any broker that might cache policy for an event type.

5. MANAGING BROKER TRUST

In very large scale peer-to-peer pub/sub networks, it is probably a naive assumption that all brokers, and the paths between them, can be completely trusted. Not only does one have to consider the possibility of malicious brokers, but the network of event brokers may span several different administrative domains. In addition, performance considerations may come into play — some brokers may have more powerful computational resources than others. This section explores our proposal for dealing with authorisation of brokers to transport particular event types. Thus, particular event types may employ a broker network which is a sub-graph of the entire pub/sub system. One noteworthy corollary is that the local broker of a given publisher or subscriber will need to be trusted. Generally a client will communicate with several brokers for reliability. Using our extended ar-

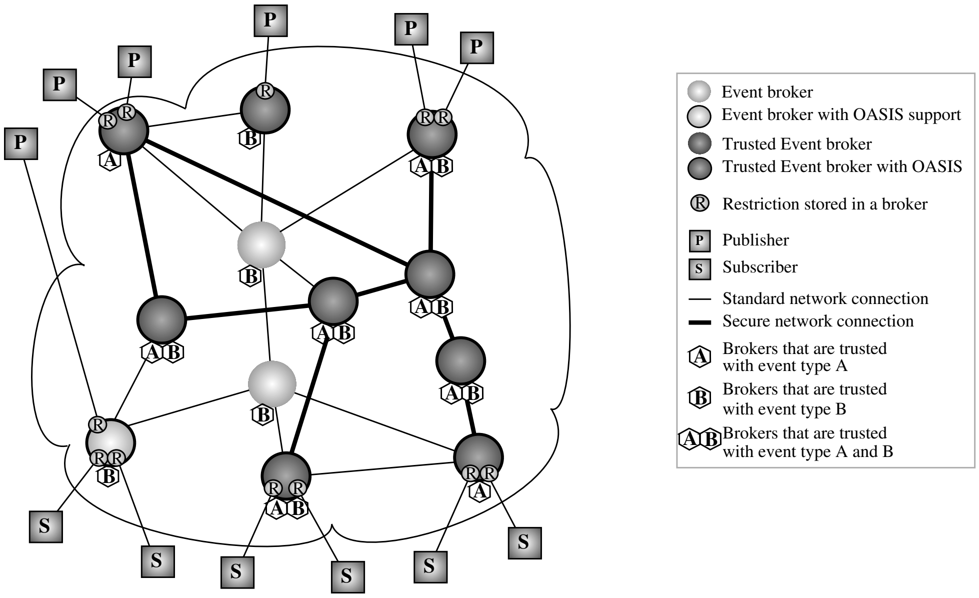

Figure 6: Trust of brokers.

chitecture, wherein brokers have different trustworthiness, will strengthen this requirement — certain brokers may not be permitted to serve certain event types at all.

Event type protection comes through the restriction of brokers' authorisation to use the access control policy rules associated with this type. Which brokers are permitted to participate in a given event type's transport is under the control of the type management policy, probably owned by the event type owner.

It is preferable that trust in the brokers is distributed throughout the pub/sub network. Indeed, the event type owner probably, on average, knows little about the brokers in the network. However brokers are likely to have metrics to judge the performance of their neighbouring brokers (e.g. delays or errors in communication). Thus we propose the use of the *certificate chains* in OASIS appointments to form a web-of-trust that spans the authorised sub-graph in the pub/sub system. For this we again rely on X.509 certificates. In Figure 6, we have developed the network of Figure 5 to indicate two levels of network trustworthiness — only brokers with secure interconnections are permitted to route event type 'A'.

We assume that each broker holds an appointment in the form of a unique X.509 certificate (this is required by our existing OASIS implementation for the SSL links established between services). The event type owner signs the certificates of those brokers connected to it that it trusts. These brokers can then sign their immediate connections and so forth. Assuming that publishers and subscribers have a trusted root certificate for event type owners (owned by the administrator of the pub/sub system itself), they will be able to verify that a local broker who accepts their advertisement is in fact authorised to carry such events.

Once the advertisement or subscription has been set up, the actual events flowing through the system can be symmetrically encrypted between brokers, if it is feared that the pub/sub infrastructure itself is at risk of being packet sniffed. Note that the actual allocation of network resources to facilitate inter-broker communication is an implementation decision below our level of interest. Logically we consider a separate network channel for each event type, although in all likelihood most if not all of these events will be able to travel down, for example, a small number of actual TCP connections, one or two encrypted and the others not.

6. IMPLEMENTATION

In this section, we present the enhanced implementation of Hermes that includes the role-based access control mechanisms introduced in this paper. Since our proposed architecture makes few assumptions about the internal implementation of the publish/subscribe middleware that it is deployed on, only small parts of the current Hermes system required modification. The major changes were the addition of new parameters to the event broker API and the integration of OASIS with local event brokers for carrying out policy-based access control decisions. An authoritative replica of the subscription and advertisement policy is conveniently stored at the rendezvous node (cf. Section 3.1.1) for this event type. This allows any local event broker to access policy via Hermes' peer-to-peer overlay network in a simple and scalable way. In addition, local brokers maintain a policy cache to reduce network load. Other types of policy are kept in a centralised repository.

6.1 Hermes Broker API

Figure 7 shows the public API of a Hermes event broker after OASIS integration. For simplicity of exposition, we do not show the exceptions thrown by the calls in case of failure or rejection. All API calls require a client to authenticate itself when invoking a function by providing a set of credentials (`creds`), which will effectively communicate this client's role in the system (`publisher`, `subscriber`, or `typeowner`). The identity and credentials are then passed to the OASIS


```
connectPublisherToBroker(publisher,creds)
disconnectPublisherFromBroker(publisher,creds)
connectSubscriberToBroker(subscriber,creds)
disconnectSubscriberFromBroker(publisher,creds)

addEventType(typeowner,creds,eventType)
removeEventType(typeowner,creds,eventType)

subscribeType(subscriber,creds,eventType,callback)
unsubscribeType(subscriber,creds,eventType)
subscribeTypeAttr(subscriber,creds,eventType,filter,
  callback)
unsubscribeTypeAttr(subscriber,creds,eventType,filter)

advertise(publisher,creds,eventType)
unadvertise(publisher,creds,eventType)

publish(publisher,creds,event)
...
```

Figure 7: The public API of an OASIS-aware Hermes event broker

implementation that performs a policy check in order to determine whether the particular operation should be executed by the local event broker. The `connect` calls are governed by the *broker-client connection policy*. If a client is not authorised to connect to a particular broker, it will not be able to invoke any further API calls and thus consume resources at the broker. The `addEventType/removeEventType` functions are invoked by an event type owner to manage event types according to the *type management policy*.

As mentioned before, subscriptions in Hermes come in two flavours: *type-based* (t-based) subscriptions that only specify an event type from the event type hierarchy and *type- and attribute-based* (t/a-based) subscriptions that additionally include a content-based filter over the event type attributes. The two `subscribe` calls in the API enable clients to create subscriptions. Following our model, a restriction may have to be imposed on a submitted subscription so that it complies with the *subscription policy*. The three types of restriction outlined in Section 4.2 were implemented as follows:

(1) Generic Non-Optimised Restriction Predicates. Since a generic restriction predicate can be an arbitrary predicate on the t-based or t/a-based subscription, it can only be evaluated at the local event broker. The content-based routing cannot handle the unbounded expressiveness of these predicates so that the original subscription is propagated through the overlay broker network. As a result, all the events received at the publisher- and subscriber-hosting local event brokers are filtered by the generic restriction before being propagated through the publish/subscribe network or before being delivered to the subscriber.

(2) Publish/Subscribe Restriction Predicates follow the expressiveness of t-based or t/a-based subscriptions in the sense that the result of applying a pub/sub restriction function to a subscription is another, more limited subscription. This restricted subscription, instead of the original one, is then propagated through the Hermes network. The original subscription is nevertheless stored in case the pub/sub restriction function changes due to dynamic policy evolution.

(3) Hybrid Schemes. These are a hybrid form of (1) and (2). When such a restriction is in place, the original subscription is replaced by a more limited version that is then propagated through the pub/sub system. However, all future events that match this limited subscription are also filtered by the restriction predicate at the local broker.

Advertisement requests submitted by publishers are handled analogously to subscriptions. Again, three variants of restrictions are implemented. However, pub/sub restriction functions can only limit the type within the type hierarchy, since Hermes advertisements cannot include content-based filters.

The API call to `publish` will verify the client identity but will not cause a verification of the published message against the access control policy if pub/sub restriction functions specify the advertisement policy. Since Hermes ensures that a published message adheres to a previously submitted advertisement, this is sufficient to prevent the publisher from violating policy. However, for generic restriction predicates, each event message is validated separately.

6.2 Policy Dissemination

Another major requirement of our implementation is to be able to disseminate access control policies through the pub/sub system automatically. It is natural to use the pub/sub system itself for this. When a particular policy such as a subscription or advertisement policy for an event type needs to evolve, the policy owner (e.g. the event type owner) publishes a *policy evolution event*. These events never reach publishers and subscribers, but as mentioned earlier, they will reach every broker that might currently be caching policy for this event type. They are passed to each local broker's OASIS implementation, which then updates its policy repository and handles the policy change. From the perspective of Hermes, policy meta events are regular events and do not need to be treated specially.

Local brokers should maintain the original advertisements and subscriptions from their clients so that the access control restrictions can be recomputed as necessary based on such policy evolution. This approach permits policy to either become more or less restrictive transparently to the clients of a broker; that is, publishers and subscribers do not need to take any action based on changes in access control policy.

6.3 Broker Trust Management

If event brokers are partially trusted and are only allowed to handle certain event types, as explained in Section 5, Hermes' content-based routing algorithm must be adjusted to support this. We adopted a simple solution by marking brokers that are not trusted as failed. This has the advantage that the content-based routing algorithm will ignore these brokers and automatically attempt to create a valid dissemination tree that involves only brokers that are part of the certificate chain for this event type. In essence, we are creating a separate (trusted) overlay broker network for each event type. If a previously trusted broker becomes untrusted, a policy change event is published by the event type owner. All brokers which are part of the trusted overlay network for this type will subscribe to these events. However, when a broker becomes untrusted, it may prevent the propagation of the policy change events to downstream broker in the event dissemination tree. In order to prevent this from happening, trusted event brokers receive periodic, signed timestamp events from the event type owner. Once a broker is untrusted, it will no longer receive these time events

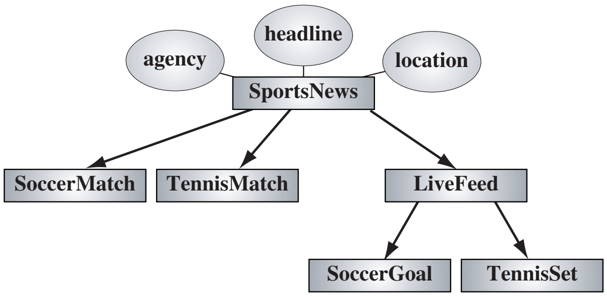

Figure 8: Our example event type hierarchy.

(since these events would need to be signed by its neighbouring brokers, which no longer trust it). All downstream brokers will notice the absence of timestamp events and resubscribe to the policy change events from the event type owner forming a new trusted overlay network.

7. AN EXAMPLE APPLICATION

This section illustrates our research by discussing how it would apply to an electronic sports news dissemination service. From the pub/sub perspective, it is appealing for publishers broadcasting sports scores not to have to concern themselves about who is subscribed to their events. Equivalently, provided the news service is authentic, subscribers will usually not care exactly how the events in which they have expressed interest are routed to them.

From the access control perspective, if we require that subscribers and publishers are paying members of our pub/-sub community, we must ensure that outsiders cannot connect to the service unless authorised. It is desirable that we might ensure this condition is met with little administrative overhead on behalf of the event type owners, and without unduly complicating other pub/sub network clients' interaction with the system. We believe the approach presented in this paper satisfies all of these goals.

Any user intending to use the sports news service would first contact the root of the service access control policy (or some delegate) to acquire an authentication credential. Once authenticated at a local-broker, the user of the service may begin to make subscription or advertisement requests.

For our example, we assume that the root of the event type hierarchy is the SportsNews event. It has attributes providing a text version of the headline, a text string describing the location of this story, and the agency which has published it. Event types SoccerMatch, TennisMatch and LiveFeed each come directly under the sports news event root. The SoccerGoal and TennisSet types release immediate details of sporting matches as they occur, which are then edited by internal news agency participants and resent under an appropriate classified event type when games are complete. This event type structure is shown in Fig. 8

When an authenticated client makes a subscription or advertisement request to one of their local brokers, this request will be translated into an OASIS privilege request. In this simple example, the privileges can be expressed as the above event names prefixed by `pub` or `sub`. Each privilege will have the parameters `headline`, `location`, and `agency`.

A complete OASIS policy for this example is outside the scope of this paper, however we discuss a few possible scenarios. Firstly, it might be possible to purchase different levels of service. For example, a bottom-level package may provide only tennis news, but no soccer. A medium-level package may add soccer news. Finally the highest-level package would allow access to all sports news events including the live feed types. The event publishers do not need to consider subscription policy, since this is handled by the brokers trusted to do so.

Consider a client who pays for the bottom-level package. In our basic system this client is free to issue a subscription to soccer and tennis match news, but the OASIS privilege request for the soccer match subscription will fail (however tennis match events will be delivered).

We have discussed modes of operation in which the client's subscription is automatically downgraded to lower event types, versus those in which the client has to communicate directly with the local broker if their initial request fails. In the former, a simple client such as a PDA might issue a SportsNews request, even if the corresponding user is a bottom-level package holder. This subscription will be automatically downgraded into a tennis match event subscription (since these are sub-types of the SportsNews type). Note that the original subscription will be maintained by the local broker. If the SoccerMatch event type owner decides to allow bottom-level package holders access, an OASIS policy change event will be published, and all brokers currently on the path of such events will update their policy. In the case of our PDA user, they will automatically now also receive soccer match news, based on the re-computation of their original request in the face of policy evolution at their local broker. Note that here, access control is levied via the underlying pub/sub type system. Thus, except when subscribing or advertising, there is no additional computation cost to achieving this access control.

Subscriptions which require determination of predicates unsupported by the underlying pub/sub system will incur per-event computational overhead at the local broker. For example, a particular client might be given a promotional subscription which just shows events relating to matches where the score is changing more than once a minute. Only a small amount of state is required to track this situation at the local broker, but the underlying pub/sub system will probably not provide any assistance – it would be unusual to maintain such per-subscriber state throughout the broker network. Thus all events will be routed to the local broker and the restriction applied there.

Utilising access control over advertisement would easily facilitate agreements with partner agencies. Such an agency may process all the soccer goal events in a particular set of locations, for example, and be permitted to publish consequent soccer match events, but only for those locations.

Note that we have chosen a rather arbitrary event type hierarchy. In situations where the classes of events are likely to change frequently, it may be more appropriate to avoid creating explicit event types, and instead to handle events with classification attributes. In the above case all sports news stories could have been published as SportsNews events, and all the access control might have been applied to attributes of these events. This is really a matter of policy design – type classification may be too rigid a solution, similarly more flexible approaches increase the likelihood of accidental security holes. Also, aggregating events through a single type in this way will require efficient content-based routing to be provided by the pub/sub system, and may involve a greater number of brokers having to cache larger collections of OASIS policy.

8. RESTRICTION QUERYING

There are a number of areas of future research stemming from the work presented in this paper. This section presents some preliminary work into situations where clients are not agnostic to our access control additions, and are permitted to query their local brokers about the restrictions being placed on their publications and subscriptions. Note that this may be inappropriate for some applications; the nature of a client's restriction may itself be a sensitive piece of information.

There are a number of advantages to allowing clients to query the filter being applied to their event stream. It may be that the restriction was caused due to a lack of OASIS credentials provided to the local broker, when in fact the client has further qualifications they would now know to present. Alternatively, the client may choose to optimise their behaviour based on knowing which events it is currently pointless to try to publish, and which events it now knows that it cannot rely on receiving.

An extended version of this restriction querying might allow a client (publisher or subscriber) to query the pub/sub system as to restrictions which may be in place throughout the broker network, and not merely at their local broker. This would permit a subscriber, say, to determine whether the complete extent of their subscription is currently satisfiable, given the union of filters over the publishers of that event type. Similarly, publishers could gain some understanding of the extent of interest in their events distributed throughout the network, perhaps reducing the workload of this publisher by determining an appropriate event transmission granularity. This corresponds to an access control variant of a *source quenching* mechanism as introduced in the Elvin system [9]. Source quenching enables a publisher to temporarily stop publishing when no subscribers are interested in any of its event types.

9. RELATED WORK

We found two pieces of recent research which were directly related to our work. In [10], Wang et al. present a number of considerations for access control in general publish/subscribe systems. As discussed in Sect. 2, their paper also discusses the need to be placed within a continuum from totally secure, accountable systems, to systems with efficient broadcasting and routing of events largely powered by a lack of need for security-related accounting.

A similar introduction is presented in [6]. Miklós devotes significant attention to specifying maximum and minimum security restrictions by assuming an ordering of events. We feel this approach is too restrictive, and unnecessarily prescriptive. Another problem with his work is the apparent lack of implementation details. Our approach provides a more expressive policy language with which to control security. After we have completed privilege negotiation we can compile down any restrictions into the event filters of the pub/sub system itself, enabling their enforcement to be managed efficiently.

10. CONCLUSION

We have presented a general architecture for integrating role-based access control into publish/subscribe messaging systems. We demonstrated our approach through an implementation that combines the Hermes pub/sub system and OASIS RBAC. We discussed likely application scenarios and then related our work to other current research. Our architecture is expressive and powerful; it supports optimisation of access control checks by the pub/sub infrastructure where possible, as well as the management of networks in which intermediate nodes are not all equally trustworthy.

11. ACKNOWLEDGEMENT

András Belokosztolszki is supported by King's College Cambridge, the John Stanley Graduate Fund, and the United Kingdom Overseas Research Students (ORS) Awards Scheme. David Eyers is funded by the Cambridge Australia Trust and the ORS Awards Scheme. Peter Pietzuch's research is supported by UK EPSRC and QinetiQ, Malvern.

12. REFERENCES

[1] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri. Generic Support for Distributed Applications. *IEEE Computer*, pages 68–77, Mar. 2000.

[2] J. Bacon, K. Moody, and W. Yao. Access control and trust in the use of widely distributed services. In *Middleware 2001*, volume LNCS 2218, pages 300–315. Springer-Verlag, November 2001.

[3] J. Bacon, K. Moody, and W. Yao. A model of OASIS role-based access control and its support for active security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, Nov. 2002.

[4] J. H. Hine, W. Yao, J. Bacon, and K. Moody. An architecture for distributed OASIS services. In *Middleware*, volume LNCS 1795, pages 104–120. Springer-Verlag, 2000.

[5] ITU-T International Telecommunication Union. ITU-T recommendation X.509, 2000.

[6] Z. Miklós. Towards an access control mechanism for wide-area publish/subscribe systems. In *International Workshop on Distributed Event-based Systems*, July 2002.

[7] P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02), July 2002.

[8] R. Sandhu, E. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[9] B. Segall and D. Arnold. Elvin has left the Building: A Publish/Subscribe Notification Service with Quenching. In *Proc. of AUUG Technical Conference '97*, Brisbane, Australia, Sept. 1997.

[10] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. Security Issues and Requirements in Internet-scale Publish-subscribe Systems. In *Proceedings of the Thirty-Fifth Annual Hawaii International Conference on System Sciences (HICSS'02)*, page 303. IEEE.

