Computer Science and Systems Analysis

Computer Science and Systems Analysis

Technical Reports

Miami University *Year* 2006

Designing a Publish-Substrate for Privacy/Security in Pervasive Environments

Lukasz Oprychal* Atul Prakash†

Amit Agrawal‡

*Miami University, commons-admin@lib.muohio.edu

†Miami University, commons-admin@lib.muohio.edu

‡Miami University, commons-admin@lib.muohio.edu

Designing a Publish-Subscribe Substrate for Privacy/Security in Pervasive Environments*

Lukasz Opyrchal
Miami University
Oxford, OH
opyrchal@muohio.edu

Atul Prakash
University of Michigan
Ann Arbor, MI
aprakash@umich.edu

Amit Agrawal
Indian Institute of Technology
New Delhi, India
csu02103@cse.iitd.ernet.in

Abstract

The emergence of a multitude of technologies for tracking locations is leading to the design of pervasive location-tracking environments. In order to explore issues in the design of such environments, the University of Michigan is deploying a network of location sensors in a number of buildings. Managing privacy is expected to be a significant concern for acceptance of such pervasive environments. This paper outlines an initial design of a publish-subscribe communication substrate for controlled distribution of sensor data. We describe our prototype as well as a privacy-aware *location-tracking application built on top of the system. The focus of this paper is on policy management so as to provide means for allowing users to control distribution of data tagged with their ID to other users and services. The paper shows how a wide variety of policies can be specified in the system and points out directions for future work.*

1 Introduction

Computational environments are becoming increasingly pervasive due to increased interest in the use of technologies such as RFID tags for tracking objects, people, and for use of data mining methods for determining users' behavior to help deliver more targeted products and information, etc. Applications are also emerging where cell phone companies track users' location to help provide location-based services such as notifying a user when their friends are nearby, or of restaurants and gas stations in the vicinity.

However, such environments also raise significant privacy concerns in the minds of people, whether the concerns are grounded in reality or not. For example, findings from the Active Badge systems [17] suggest that individuals do not wish to have their movements available to everyone. Finding a solution to those concerns is of considerable importance if pervasive environments are to be widely used.

With the help of a National Science Foundation infrastructure grant, a location sensor network is being deployed in several parts of Department of EECS building, consisting initially of RFID and 802.11g location tracking sensors. In addition, a location sensor network is also being deployed in one of the medical clinics at the University of Michigan to help track locations and activities of patients who suffer from memory problems and are recovering in the clinic so that they can be gently reminded if they miss a recommended appointment or activity. Culnan defines privacy as simply the ability of an individual to control the terms for acquisition and usage of their personal information [13]. The question that we consider in this paper is how can one build applications and services around the data that will be collected,

*This work is supported in part by grants from the National Science Foundation (grants 0082851 and 0325332), Intel, IBM, and Microsoft.

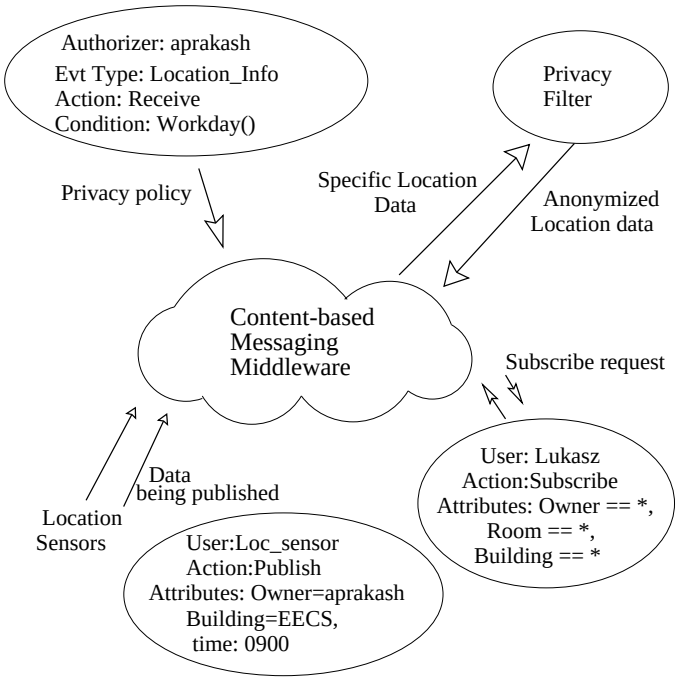

Figure 1: Incorporating policies in a content-based messaging system

while providing means to the users to have significant control over the conditions of distribution of their data?

We note that the focus of this paper is limited to those distribution constraints that can be managed and enforced by a computational infrastructure. At some level, legal mechanisms are likely to be required to deter people from leaking information that they have access to. For this paper, the assumption is that people are not generally looking to maliciously violate privacy rules. For localized environments such as the medical clinic or the EECS department, we believe such an assumption to be reasonable. At the same time, however, the goal of the computational infrastructure should be to prevent inadvertent violations of users' privacy policies to the extent possible, and thus finding formal ways of encoding policies is desirable so that they can be computationally enforced.

To illustrate the issues in policy specifications, an example of a simple policy used by some mobile phone services (e.g., i-mode services from DoCoMo and recently from AT&T) is one that allows users to determine their friends' nearest cell, provided their friends have given them permission to do so (similar to permissions to view status in instant messaging systems). In general, however, the policies can be much richer:

- Environment-dependent sharing. Users may want to share their location information only at certain times of the day, or when they are in certain locations. Users may want to specify that their location is available during specified events, so that they can be more easily tracked down, for example, if they are late for the event.

- Privacy-protected access to location-based notification and autominder services. It should be possible for a user to receive location-based notifications, such as the nearest gas station, without disclosing their location to the gas station. Anonymizing, trusted services will be needed as intermediaries.

Figure 1 shows an example scenario in which our system is used. A user *aprakash* has expressed a

privacy policy to allow receipt of his location information only during the Workday. Location sensors publish data for different users, including *aprakash* (which is time-stamped by either the sensors or the middleware). A user has expressed an interest in receiving location information for all users, but will only get location information for *aprakash* during the Workday period. Of course, as pointed out above, the policies can be richer. The user aprakash may wish to restrict availability of published data to only certain users or services. Furthermore, the user may wish to allow or restrict the ability of those users or services to delegate their right to other users. We consider these issues later in the paper.

In our model, subscribers to data can be either users, end-user applications, or services. Services can include *privacy filters* that anonymize sensor data or aggregate data from multiple sensors or over time. Examples of data anonymizing include hiding the name of the user from location data, abstracting the location data so that a recipient only knows that the user is in a building, but not the specific room, etc.

We note that for some users, relying on a common publish-subscribe infrastructure completely to do the right thing with shared data may still be a significant trust issue. One idea we have considered is to support the use of multiple *publish-subscribe* brokers, in which a user has a designated trusted *privacy broker*, which could be running as a service on a machine that the user owns. The privacy broker could use our policy infrastructure, just like a centralized service would, except that it would only be handling events that pertain to the users that trust it.

The rest of the paper is organized as follows. Section 2 presents related work in publish subscribe as well as privacy areas. Section 3 describes our security models for content-based publish subscribe systems, security policy dimensions relevant to these systems and our security policy language. Section 4 describes the prototype of a secure content-based publish subscribe system based on the security model and policy language described here. Section 5 describes a *privacy-aware* location tracking application built on top of our secure pub-sub system. Section 6 describes our approach to evaluation of the system. Finally, in Section 7, we present our conclusions and directions for future work.

2 Related Work

In recent years, publish subscribe (pub-sub) middleware has become an emerging paradigm for distribution of data among users and services. In publish-subscribe systems, there are two types of users: publishers and subscribers. The infrastructure mediates delivery of events from publishers to subscribers. Current commercial publish subscribe middleware implement the *subject-based* paradigm, where every event is annotated with one of the pre-defined subjects (topics, channels, etc.) [6, 30, 22, 29]. Subscribers are allowed to subscribe to one of the pre-defined topics.

An emerging alternative to subject-based systems are *content-based messaging systems* [28, 4, 12, 16, 21]. These systems support an event schema defining the type of information contained in each event (message). For example, applications interested in location information of users may use the event schema:

LOC_INFO: [user: String, building: String, room: String]

A content-based subscription is a predicate against the event schema, such as

(user = "aprakash" & building == “EECS Building”)

Only events that satisfy the subscription predicate are delivered to the subscriber. Examples of content-based publish subscribe systems include PreCache [26, 14] and content-based prototypes from Microsoft [10] and IBM [4].

To help provide an infrastructure for distribution of sensor data to applications that handle these types of policies, we propose to use a content-based messaging substrate as the underlying mechanism. However, there is a significant difference from earlier work in content-based publish-subscribe systems. In existing systems, the focus is on subscribers being able to control what information they receive by specifying predicates, for example, to handle the information overload problem. In contrast, our primary focus is on publishers being able to control who receives their data and under what terms. We thus need to augment the content-based publish-subscribe paradigm to allow publishers (users) to control dissemination of information they own.

A general description of security requirements in content-based systems is given in Wang *et al* [31]. The authors provide a high level description of potential issues and point in the direction of possible solutions. One of the first attempts at solving the access control problem in content-based systems is presented in [5]. The authors combine role-based access control (RBAC) with a distributed event notification service. Unfortunately, the authors do not describe the details of their policy language and the type of access control rules that can be supported by their system.

A number of systems have emerged that that support privacy in pervasive environments. One such system is the Confab system by Hong and Landay [20]. There are several differences in approach. While our system is based in a content-based messaging middleware, their system is based on hybrid blackboard and dataflow architecture, leading to potentially different programming models. In Confab, each data item is tagged with privacy preferences and at present only relatively simple preferences are supported. In contrast, our approach is to make a distinction between privacy policies and data. Privacy policies generally come from users and can apply to multiple data items, as opposed to being explicitly attached to each data. Both approaches have pros and cons, depending on the threat model and the targeted applications.

Campbell *et al* describe many issues in designing location-based pervasive environments [11]. The authors explore challenges in building security and privacy into pervasive environments and present a solution based on the Gaia authentication service [27] and Mist Routers [3]. Their main concern is not policy controlled dissemination of location information but rather anonymous collection of location information from different sensors.

Project Aura [15] is a large system for pervasive computing environments. Hengartner and Steenkiste provide protection of location information in Aura [18, 19]. Their work concentrates on policies for environments where location information can come from multiple places (GPS phone, wireless networks, or a person's personal calendar). The authors deal with issues of trust (which services can be trusted) and delegation. Their solution is based on SPKI/SDSI certificates. The policy language is similar to ours and policy evaluation is similar KeyNote. The main difference is that their solution is designed for querying location information (i.e., where is Alice? or who is in room 302?) and not for dealing with distribution of such information as in a publish subscribe system as described in this paper.

3 Policy Model

Both the privacy as well as the security research communities have examined issues in representing policies, with P3P [1] being an example representation in the privacy community and techniques such as role-based access control models (RBAC), trust management systems, Chinese wall models, and Clark-Wilson models in the security community [7]. Our work largely builds on the work in the security community because security policies have been a subject of investigation for a long time and the maturity of the tools for enforcing those policies. Also, our long-term interest is in taking a unified approach to security and privacy because ultimately, security underpinnings are required to enforce privacy when unauthorized users attempt to tap available data.

For the purpose of this paper, we will assume that all clients connecting to a content-based publish

subscribe system are authenticated. The data security problem in content-based publish-subscribe systems is further discussed by Opyrchal and Prakash [23]. In this paper, we primarily focus on access control and delegation aspects in publish-subscribe systems and their extensions to managing privacy in pervasive environments.

3.1 Basic Definitions

Event Owner: Similarly to Belokosztolszki *et al* [5], we introduce the notion of an **event owner** in our model. Event owner is an entity who has the right to authorize other entities to perform certain actions. An event owner can authorize other users to subscribe to its events, receive events, or even delegate authority to modify the policy for the events it owns. In most applications, we will associate a different owner for individual events within an event type. For example, events of type "LOC_INFO" can be owned by different users - if an event is about *joe*, then joe is the owner of that event. Each owner will control access to the events it owns

Depending on the application, there may be one owner for an event schema, irrespective of event contents, in which case the owner has complete control over access control rules for that event schema. For example, if a service provides aggregated sensor data, it may claim to be the owner of all that data, irrespective of the ownership of individual events from which the aggregate data was generated. Users may only be able to choose whether to provide information owned by them to the service (under appropriate terms), but not have rights to the aggregated data.

Application: The pub-sub system can support multiple *applications*, where applications refer to broad categories such as a user-tracking system. In turn, each application consists of a number of event types. Each application must have at least one *administrator*. The administrator is a client of the pub-sub system who has the right to delegate authority to perform different actions within the application. For example, the administrator can delegate the *ownership* of different event types, add new event types, etc. Figure 2 shows the administrator of a LOC_APP application (location-tracking application) delegating rights to user *joe* when attribute *user* is equal to "joe".

This paper does not present a new policy evaluation technique. Our original prototype implementation, used the KeyNote Trust Management System [8] for evaluating and checking our security policy. We are currently modifying our prototype to use the CPOL policy evaluation engine developed by one of this paper's authors [9]. CPOL has a C++ interface for specifying policies but its expressiveness is similar enough to KeyNote's that it is possible to show CPOL rules in a KeyNote-like syntax. Therefore, all of our policy examples are based on the KeyNote language syntax. We briefly describe the different fields that are found in KeyNote policy specifications:

Authorizer - this is the entity granting the right. The authorizer can be any entity in the system (provided it is allowed to add policy rules). If an authorizer tries to grant rights which it doesn't have itself, the rule is rejected. The user *admin* is implicitly granted all rights. This is achieved by writing a rule with authorizer being set to "POLICY" (figure 2). Such rule is always trusted. The authorizer field must not be empty.

Licensee - this is the recipient of the right. The licensee field can contain a single entity, a list of entities, or a wildcard [1]. A wildcard indicates that any user who satisfies the condition specified in the *condition field* is given the right.

Condition - this is the condition that is checked when policy rules are evaluated. If the condition evaluates to "true" then the licensee is given the appropriate rights (in practice, conditions are evaluated bottom-up until a rule with authorizer value "POLICY" evaluates to "true").

[1] In the KeyNote language there are no wildcards. To delegate the right to any user who satisfies the condition, the licensees field must be left out of the rule.

```
Authorizer: POLICY
Licensee: admin
Conditions: (app_domain == “LOC_APP”) -> “true”;

Authorizer: admin
Licensee: joe
Conditions: (app_domain == ``LOC_APP'') &&
            (evtType == ``LOC_INFO'') &&
            (user == ``joe'') && (owner == ``joe'') -> ``true'';
```

Figure 2: Entity admin receives all rights for the LOC_APP application and grants ownership rights to user joe when the *user* attribute is “joe”

```
Authorizer: admin
Licensee: joe
Conditions: (app_domain == “LOC_APP”) && (evtType == “LOC_INFO”) &&
            ((action == “SUBSCRIBE”) || (action == “RECEIVE”) ||
            (action == “CHANGE_POLICY”)) && (owner == “joe”)
            -> “true”;
```

Figure 3: Assignment of ownership rights to the owner.

Signature - cryptographic signature verifying that it was the authorizer who wrote the rule (not shown in most of our examples). The signature field is required in all rules except for the “POLICY” rules which are read from file and implicitly trusted (unsigned rules are not accepted over the network).

The *condition rules* are simple logical expressions and may use the **and** operator “&&” and the **or** operator “||”. Condition rules may use a combination of event attributes and external attributes. The availability of external attributes depends on the implementation. They can include current time, number of received events (by each subscriber), etc.

3.2 Access Control

We identify a number of actions that can be performed in our system. Each action has certain security implications and should be controlled through an access control policy. The supported actions are:

- **authenticate** - authenticate to the system
- **advertise** - introduce a new event type into the system
- **publish** - publish an event of a particular type
- **subscribe** - subscribe to an event of a particular type
- **receive** - receive an event
- **change policy** - modify the security policy (add/remove/modify rules)

We consider authentication to be outside of our security model. It is used only to positively authenticate users trying to perform one of the other actions. Authentication must be performed in order to enforce the security policy.

Each *owner* usually receives the right to publish, subscribe, receive, and change policy for the events/event types he owns. A KeyNote rule that would permit this for user *joe* is shown in Figure 3. Using that rule, user *joe* receives the right to subscribe and receive his own events as well as to manage policy for those events. The owner *joe* can in turn authorize other users to perform certain actions.

```
Authorizer: joe
Licensee: alice
Conditions: (app_domain == "LOC_APP") && (evtType == "LOC_INFO")  &&
            ((action == "SUBSCRIBE") || (action == "RECEIVE")) &&
            (owner == "joe") -> "true";
```

Figure 4: User *joe* grants the right to subscribe and receive his events of type LOC_INFO to user *alice*.

3.3 Delegation

One of the problems with some trust management systems is *delegation of rights*. The KeyNote system, which we use in our prototype, allows an entity to delegate any rights that it possesses to other entities. In figure 4, user *alice* is authorized to subscribe and receive all events with attribute *user* equal to "joe".

From a privacy perspective, the ability of a user to grant all or a subset of rights to another user, irrespective of the ownership of data, is not always desirable. In a location tracking application, for example, Joe might give Alice the rights to subscribe and receive his location events. But Joe may not want Alice to, in turn, pass these rights to another user.

Alice could simply send each event about Joe to another entity outside of the pub-sub system. It is beyond the scope of our system (and perhaps any software system) to control Alice's ability to leak the data received outside the system, or even to republish Joe's data with a false claim of ownership within the system. Alice would be a malicious user in that case and this is the classical problem of digital rights management, which we know is hard to do if users are malicious. However, we do want to provide a basic solution for Joe to make sure Alice cannot inadvertently allow others to receive his events within the system by simply adding a rule that grants such rights to others, as allowed by KeyNote.

In order to restrict Alice's ability to grant those rights, we provide the *change_policy* action. In order to allow a user to receive some events, Alice would have to enter a new policy rule. To do this, she would need the right to perform the *change policy* action for events owned by Joe. If Joe does not grant the right to modify policy rules for his events to Alice, Alice will be unable to delegate the rights to receive Joe's events without Joe's permission (figure 4 shows Joe delegating the rights to subscribe and receive events but not to change policy). We note that the CPOL policy evaluation system adds support for delegation which is not present in KeyNote. This feature of CPOL allows better and cleaner control of delegation rights.

3.4 Data Security Policy and Advertisements

An **advertisement** is a way for an authorized entity to introduce a new event type into the system. The access control policy is checked to make sure that the advertiser is authorized to perform the action. An advertisement describes the new event type and indicates the type of access control and data security required. An example of an advertisement is shown in figure 5. More information about issues presented in this section can be found in [24].

Types of access control

The following are the possible types of access control:

No-control - no access control is performed for events of this type. All users of the system are allowed to subscribe and to receive events of this type.

Subscribe-time - access control policy is checked whenever a new subscription for events of the particular type is entered. If allowed, the new subscription should be inserted into the subscription set, otherwise it is rejected. Since subscription requests are controlled through the access control policy,

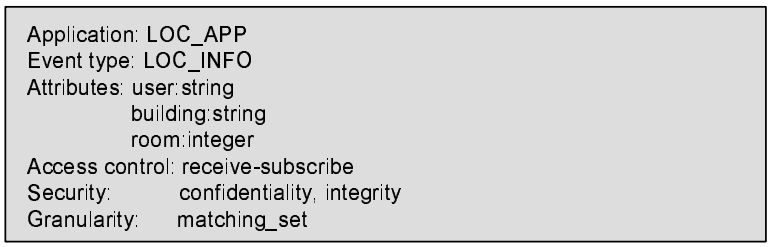

Figure 5: An advertisement for event type “LOC_INFO”

individual events are delivered to the interested (matching) subscribers without further access control checks.

Receive-time - access control policy is checked before events are delivered to interested subscribers. All users are allowed to enter subscription requests for events of this type. When an event is published and a matching process determines a set of interested subscribers, the access control policy is checked whether each subscriber in the matching set is allowed to receive the particular event.

The receive-time policy is useful when access control rules depend on the environment or other dynamic values external to the event itself. For example, an application may allow users to enter any subscription but may limit the number of events received by each subscriber to a particular number. Subscribers may also be limited to receive events during a particular time during the day or only during weekdays (and not on weekends). It is impossible to check these types of rules at subscribe time.

Receive-Subscribe-time - both subscription attempts and event receive attempts are controlled. This policy combines the subscribe-time and receive-time policies.

Data security guarantees

The data security guarantees field specifies which guarantees are required for events of this type. The choices are *confidentiality, integrity*, and *sender authenticity*. When confidentiality is chosen, events are encrypted while traveling through the pub-sub system. Events are also encrypted when they are delivered from a broker to its interested subscribers. Similarly, when integrity is chosen, a message authentication code (MAC) is added to each event.

Granularity of security guarantees

The granularity parameter applies only if confidentiality was chosen as one of the security guarantees. There are a number of ways confidentiality of events can be provided. Events with *no access control* and confidentiality, must only be protected from outsiders (entities not authorized to use the pub-sub system at all). We call this type of granularity *system granularity*.

Other event types may require that only authorized subscribers can gain access to all events of the particular type (authorized subscribers can receive all events of this type). This type of granularity requires that all events are encrypted in such a way that subscribers which are not authorized to receive any events of this type cannot gain access to those events. An example of this type of an application is a stock quote service where users must pay in order to receive quotes. Once they pay, they can receive all stock quotes. We call this *event type granularity*.

Finally, some applications require that only the set of authorized and interested subscribers can gain access to events. This means that for each event, a set of interested and authorized subscribers is determined. Then, only subscribers from this set should be able to gain access to that event. This is useful for applications where subscribers may be authorized to subscribe to and receive any event but can only receive a limited number of such events. Similarly, this is also useful when some users are allowed to

receive events only during certain times of the day. If event type granularity was used some subscribers, who have already reached their event limit, could simply "sniff" network traffic to gain access to more events of this type (since they have the appropriate security keys). We call this *matching set granularity*.

4 Prototype

We have built a prototype content-based publish subscribe system to demonstrate the viability of our model and policy language as described in section 3. The current version implements most of the discussed features. This section describes the implementation of our pub-sub system and the next section describes the *privacy-aware* location tracking application built on top of the pub-sub system.

Our publish subscribe system is implemented in Java (with the exception of KeyNote which is written in "C"). The current implementation only supports equality tests in the subscription language. It is straightforward to add additional operators to the subscription language and we are currently implementing inequality operators. The matching algorithm is based on tree matching algorithms presented in [2, 4, 25]. We are currently integrating our system with the CPOL policy evaluation engine which will replace KeyNote. CPOL [9] was designed to offer expressiveness of the policy language similar to KeyNote. The main advantage of CPOL is its much better performance characteristics. Experiments show that CPOL is several orders of magnitude faster than KeyNote when evaluating policy rules.

The subscription language supports *wildcards*. For example, the subscription

```
(user=="joe" && building=="EECS" && room=="*")
```

specifies interest in "LOC_INFO" events where the *user* attribute is equal to "joe" and *building* attribute is equal to "EECS". In other words, the subscriber is interested in tracking user joe anywhere in the EECS building.

4.1 System Architecture

Our pub-sub system consists of clients (publishers, subscribers, owners, administrators) and the event delivery system. The event delivery system is designed to consist of a network of event brokers. The events will be routed between brokers using a combination of algorithms described in [4, 25]. The current implementation, designed as a proof of concept for our security infrastructure, supports only one central broker.

A broker accepts client connections and performs requested actions. The event broker maintains a database of users who have signed-up to use the system. We assume that users sign-up off-line and that during that process they generate a public/private key pair and submit their public key to the pub-sub system. When connecting to the broker, clients must authenticate first. The broker can be extended to support any type of authentication protocol. A simple solution can be an *ssh-like* protocol based on the public/private key pairs generated when signing up for the service. Currently, we implemented a simple password-based authentication protocol.

Once authenticated, clients can make requests to add subscriptions, publish events, or add new policy rules (presumably to authorize other users to perform some actions). We assume that the communication between a client and a broker is encrypted. This can be achieved by establishing a *session key* during authentication and using that key to encrypt all messages. An alternative solution is to use one of the caching algorithms described in [23] to improve performance.

The architecture of an event broker is shown in figure 6. The *client handler* is responsible for all communication with pub-sub clients. The client handler parses the message, determines the protocol

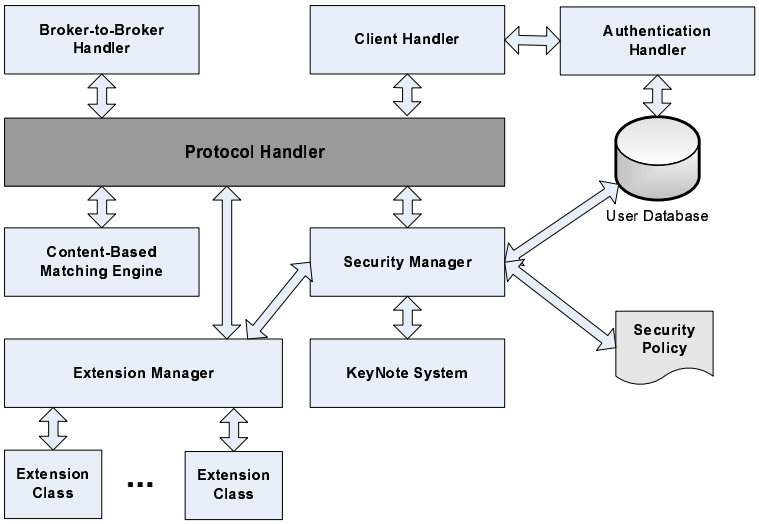

Figure 6: Broker Architecture.

type (**authenticate, publish, subscribe, change policy, etc.**), and calls an appropriate *protocol handler* method (in case of an authentication message, the client handler uses *authenticator* methods instead).

The *protocol handler* is the main part of the broker. It validates the message passed to it from client handler and decides what to do with it. In case of a valid message, the protocol handler checks with the *security manager* whether the requested action is allowed. If the security manager allows the action, protocol handler calls appropriate methods in the *matching engine* module. The security manager uses the *KeyNote System* [8] to determine whether the given action is allowed under the current security policy (the new prototype will use CPOL instead of KeyNote). This is done by constructing an *action query* (based on the parameters passed in from the protocol handler) and calls the appropriate KeyNote method. KeyNote, in turn, evaluates the action request in the context of current security policy and returns *true* if action is allowed or *false* if it is rejected.

The matching engine handles subscription requests by simply adding new subscriptions to the matching tree. In case of a publish request, the matching engine searches the matching tree to determine the set of all subscriptions matching the given event. Since subscriptions are annotated with subscriber id's, the search algorithm returns a list of *matching user id's* to the protocol handler. If the event requires an access control check before sending it to the matched subscribers, the protocol handler must check, for every matching user, whether she is authorized to receive the event. This is done by querying the policy evaluation engine separately for each matching user. We are forced to use this, rather inefficient, algorithm because of the KeyNote API which only allows the permissions for one user to be queried at a time. The new version of the system, which uses the CPOL engine instead of KeyNote, supports groups of users as well as roles which remove some of the inefficiencies. Section 6 describes the new version of the policy evaluation engine in more detail.

Sometimes, an event owner may want to authorize other users to perform an action based on attributes which are not part of the event schema. In the location tracking example above, assume that the user Joe wants to grant access to all of his location events but only during regular work hours. Since the event schema for LOC_INFO event type does not include time, it would be impossible to write such rule if we were only allowed to use event attributes. Another such example is if Joe wanted to allow user Alice to receive his events but only once an hour.

Our pub-sub system supports *external attributes* to enable users to write rules such as the ones de-

scribed above. External attributes are attributes which are not part of the event schema but are added to the event before security policy is evaluated. This allows us to write policy rules which depend on attributes which are not included in the event at publish time. The *extension manager* is the module which determines whether external attributes should be added to a particular event. By convention, external attribute names have *ext* pre-pended to them.

For the extension manager to work, we must implement a special *extension class* for each application domain. This extension class adds appropriate attributes to events of different types within the application domain. The extension manager has two important API calls: **processEvent()** which is called whenever a new event is published. This method allows the extension manager to keep track of different pieces of information, such as the number of events received by each user (as in the example above). This collected information is then used to fill in external attributes by the **addAttribs()** method. The addAttribs() API call is used before evaluating whether the current action is allowed under current policy.

5 Location Tracking Application

The application is a secure, privacy-aware, location tracking system where location sensors (RFID sensors) publish events whenever a *tag-wearing* person enters the sensor's detection radius (the infrastructure for such system will be built into the new CSE building at the University of Michigan).

We have implemented a secure and privacy-aware location tracking application as presented in figure 1. The main goal of the application is to provide a flexible and secure location service while protecting privacy of its users. Our location tracking application uses the security infrastructure of our pub-sub system to give users the power to choose with whom they are willing to share their location information and under what conditions.

The application uses Radio Frequency Identification for Business (RFID) sensors deployed throughout the location tracking area (building, campus, city, etc.). Users of the system carry small RFID badges which are detected by the sensors. The sensors transmit their data to *location publishers* (one or more) which are clients of the pub-sub system. The location publishers convert the sensor data into pub-sub events of type *LOC_INFO*. The event schema for this event type is defined as follows:

```
[LOC_INFO: (user:  String, building:  String, room:  String)]
```

The location publishers publish the events to the publish subscribe system. Users of the system can then subscribe to location information about other participants.

While potentially very useful, this type of an application introduces serious privacy concerns. Obviously, most of us do not want our location to be tracked by strangers or even by most people we know. On the other hand, it may be beneficial for people working on the same project to know where their collaborators are at certain times (for example). Location information may also be necessary for certain services that we may depend on.

It is important to understand that even if a person allows others to track her location, she may want to restrict that access at times. A professor may allow others to track his location only during normal working hours. Another possibility is to restrict location information to two values - whether a person is in her office or not. There are also people who do not want to share their location information with anybody.

A privacy-aware location tracking application has to allow users to control the flow of information about them. It has to provide flexible policy language which allows users to express complex rules such as the ones mentioned above. The location tracking application implemented on top of our pub-sub system does just that.

```
1   Authorizer: POLICY
    Licensee: location_admin
    Conditions: (app_domain == "LOC_APP") -> "true";

2   Authorizer: location_admin
    Conditions: (app_domain == "LOC_APP") && (evtType == "LOC_INFO")
                (action == "SUBSCRIBE') -> "true";

3   Authorizer: location_admin
    Licensee: location_publisher
    Conditions: (app_domain == "LOC_APP") && (evtType == "LOC_INFO")
                (action == "PUBLISH') -> "true";

4   Authorizer: location_admin
    Licensee: owner
    Conditions: (app_domain == "LOC_APP") && (evtType == "LOC_INFO") &&
                ((action == "RECEIVE") || (action == "CHANGE_POLICY"))
                -> "true";

5   Authorizer: Bob
    Licensee: Alice || Eve || Nick
    Conditions: (app_domain == "LOC_APP") && (evtType == "LOC_INFO") &&
                (owner == "Bob") && (action == "RECEIVE") &&
                ((extTime == "WORK_DAY") || (extTime == "WORK_NIGHT")) -> "true";

6   Authorizer: Nick
    Conditions: (app_domain == "LOC_APP") && (evtType == "LOC_INFO") &&
                (owner == "Nick") && (action == "RECEIVE")  && (extCollaborator == "true")
                -> "true";

7   Authorizer: Eve
    Conditions: (app_domain == "LOC_APP") && (evtType == "LOC_INFO") &&
                (owner == "Eve") && (action == "RECEIVE") &&
                (building == extBuilding) && (room == extRoom) -> "true";
```

Figure 7: Sample policy for the location tracking application "LOC_APP".

To allow users to control availability of the sensor data about them, every user is granted ownership of events about them (Rule 4 in figure 7). To get around the problem of granting ownership individually to each user (as in figure 2), the value of *owner* in the Licensee field is derived from the event itself and passed to the policy evaluation engine.

Users can then delegate a subset or all of their rights (for example rights to receive events) to other users. By default, a user's events are private and nobody but the user can receive them.

The right to **subscribe** to events is given universally to all users of the system. This is done to allow subscriptions such as:

```
[LOC_INFO:(user="*",building="EECS", room="2246")]
```

where we want to know if anybody entered room 2246 in the EECS building. Since the event owner is not known at subscribe time, it would be impossible to decide which "subscribe" policy rules apply to this subscription request. Since everybody is allowed to subscribe to all events, users can write rules about who can actually receive the events. A small part of the policy is shown in figure 7 (note that we are omitting parts of the KeyNote language syntax for clarity).

Rule 1 allows *location_admin* to administer the "LOC_APP" application. Rule 2 allows all users to subscribe to events of type "LOC_INFO". Rule 3 allows a special client, *location_publisher*, to publish events of type "LOC_INFO". Rule 4 gives all users the ownership rights to events about themselves. Bob gives permission to Alice, Eve, and Nick to receive his events in rule 5. The receive right is only valid during *work days* and *work nights*. Nick authorizes all users who are his collaborators to receive his events in rule 6. The attribute *extCollaborator* is evaluated externally. Finally, Eve allows all users to receive her events but only if the subscriber and Eve are in the same room in the same building (rule 7).

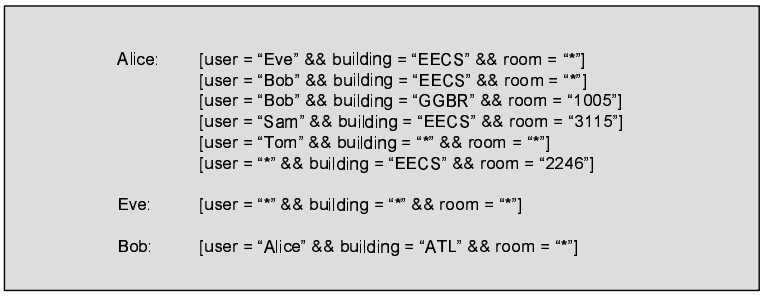

Figure 8: Few example subscriptions from the location tracking application.

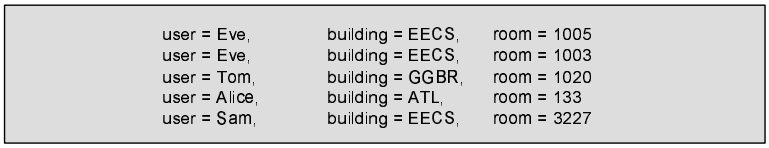

Figure 9: Few example events from the location tracking application.

We notice the use of external attributes **extTime, extBuilding, extRoom**, and **extCollaborator** in rules 5 - 7. The extension class *ExternalLocation* was implemented to add the current time and the location of the subscriber to the event attributes[2].

Figures 8 and 9 show a number of sample subscriptions and events from our location tracking system.

KeyNote assertions can be rather confusing to users and while our pub-sub system allows submission of policy rules using the full KeyNote syntax, we also offer a simplified syntax which is automatically converted to the KeyNote language. The simplified language requires only the name of the *Licensee* and the conditions. This is then converted into a full KeyNote assertion, signed with the user's private key and transmitted to the broker.

6 Evaluation

We want to evaluate our publish subscribe system and the location tracking application in two ways. One is to evaluate the performance characteristics of our pub-sub system and its security infrastructure. This includes testing the behavior of the system with different sizes of subscription sets, event publish rates, and most importantly, different sizes of security policies. In addition to testing the performance of the system, we would like to evaluate the system's usage by real users. Interesting questions involved in such evaluation include the percentage of users willing to share their locations, their trust in the system, the types of privacy rules, etc.

As of this time, the new CSE building at the University of Michigan (together with its network of RFID sensors) is not completed yet and we are unable to perform user evaluation. While our system is implemented as described above, we cannot test it with real users yet and we used a location generator to test the application.

We were able to run performance tests but it became obvious that the KeyNote system is not well suited for this type of application. Our initial experiments with about 100 security policy rules already showed that it takes about 1 second to evaluate KeyNote queries on an average system (2.8 GHz Pentium 4 computer). The KeyNote system worked well in a small prototype system but it is not suited for a larger implementation.

[2]Notice that the subscriber's location is never revealed and only used for policy evaluation.

It was obvious from the beginning that KeyNote was not designed for this type of an application. Recently, Borders, Zhao, and Prakash developed a new, high-performance policy evaluation engine called CPOL [9]. Using different evaluation algorithm and caching techniques, they were able to improve evaluation performance by many orders of magnitude. The paper argues that CPOL's performance is more than adequate for high throughput applications such as content-based publish subscribe (or the location-tracking system built on top). We are currently working on a new version of the system using CPOL instead of KeyNote. Based on the evaluation results in the CPOL paper as well as the performance characteristics of our content-based matching algorithm [2, 4], we expect that we will be able to support a large location tracking system with our new prototype.

7 Conclusion and Future Challenges

We showed that the following types of access control and privacy policies can be formalized in our system:

- Where users wish to make their data available to only selected users.
- Where users wish to place computable conditions before making data available. Those conditions can be enforced at subscription time (to prevent users from even subscribing to data) or at receive time (to allow users to subscribe to data but potentially not receive it if it violates the predicate).
- Where users wish to control the ability of users to delegate the rights granted to other users.

Services such as privacy filters can be accommodated by treating them as both a publisher and a subscriber. As a subscriber, they have to be authorized by users to receive their events. As a publisher, however, the situation gets more complicated. We envisage two scenarios, one in which the service acts as the owner of filtered data and thus controls further dissemination. This would typically be true of services that aggregate data and can be dealt with in the framework to a degree by treating the service as an original source of the data. The second scenario is that the service republishes the filtered event, but continues to associate the user with the event as far as the policy enforcement is concerned.

We presented a solution to controlling delegation of rights by placing restrictions on the ability of users to change policies. However, that solution has its limitations. It handles the situation well when policies are generally static. However, consider a scenario where a user Alice grants all rights to user Bob, including the ability to delegate rights. Bob now subscribes to Alice's events and also delegates those rights to Charlie. Unfortunately, Alice cannot simply modify her rule to take away *change_policy()* right of Bob to revoke Charlie's rights; such a change will only affect future policy update operations of Bob, not the past. From a privacy management perspective, a better solution would be to determine delegation rights dynamically. Our new policy evaluation engine, CPOL, allows direct control over delegation of rights. CPOL supports three levels of *delegation rights*: *Normal, Admin, Delegate*. Normal indicates that the grantee cannot create new access rules. Admin and Delegate levels allow the grantees to add and remove new rules on behalf of the event owner but with different restrictions on that delegate right.

In addition to the support for delegation, CPOL provides support for roles and groups. This feature allows us to express rules that grant access to people based on their role (*e.g.*, all nurses are granted the right to monitor locations of all patients in a hospital). KeyNote, on the other hand, does not support such rules directly

We would also like to provide support in our system to allow users to be prompted to "sign" a contract if they wish to subscribe to a user's data. Our plan is to make the contract be a part of the *Condition* in the policy rule (e.g., as a functional predicate ContractSigned(contractDocumentURL)), and motivated by the realization that not all aspects of privacy terms can be captured as a computable predicate and are

better expressed in a legal framework or as understandings between the publisher and subscriber. It is also possible that a subscriber may wish a publisher to sign a contract before it will accept the publisher's data or provide a service to the publisher. Designing a solution for such scenarios in our framework requires further investigation.

Finally, we would like to apply our policies to not only real-time events but also to events that may be first archived in a database. A good and efficient solution for applying the policies to queries on archived events is not obvious. It is likely to require better integration of the policy framework with the query system on the database.

References

[1] Mark S. Ackerman. General Overview of the P3P Architecture. MIT World Wide Web Consortium, 1997. http://www.w3.org/TR/WD-P3P-arch.

[2] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching Events in a Content-Based Subscription System. In *Proceedings of Principles of Distributed Computing (PODC '99)*, Atlanta, GA, May 1999.

[3] J. Al-Muhtadi, R. Campbell, A. Kapadia, D. Mickunas, and S. Yi. Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments. In *Proceedings of nternational Conference of Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, 2002.

[4] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In *International Conference on Distributed Computing Systems*, June 1999.

[5] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody. Role-based access control for publish/subscribe middleware architectures. In *Processdings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*. ACM Press, June 2003.

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

[7] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2003.

[8] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote Trust-Management System, Version 2, September 1999. Request For Comments (RFC) 2704.

[9] Kevin Borders, Xin Zhao, and Atul Prakash. CPOL: High-Performance Policy Evaluation. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, pages 147–157, Alexandria, VA, November 2005.

[10] Luis F. Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a Global Event Notification Service. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau, Germany, May 2001. IEEE Computer Society.

[11] Roy Campbell, Jalal Al-Muhtadi, Prasad Naldurg, Geetanjali Sampemane, and M. Dennis Mickunas. Towards Security and Privacy for Pervasive Computing. In *Proceedings of nternational Symposium on Software Security (ISSS 2002)*, Tokyo, Japan, 2002.

[12] Antonio Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, December 1998. Available from http://www.cs.colorado.edu/~carzanig/papers/.

[13] Mary J. Culnan. Protecting Privacy Online: Is Self-Regulation Working. *Journal of Public Policy and Marketing*, 19(1):20 – 26, 2000.

[14] Renee B. Ferguson. PreCache Unveils NetInjector Platform. eWeek, January 2003. http://www.eweek.com/article2/0,3959,808317,00.asp.

[15] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1:22–31, 2002.

[16] R. Gruber, B.Krishnamurthy, and E. Panagos. An Architecture of the READY Event Notification System. In *Proceedings of the Middleware Workshop at the International Conference on Distributed Computing Systems*, Austin, TX, June 1999.

[17] R. J. Harper. Why do and don't People wear Active Badges: A Case Study. *Computer-Supported Cooperative Work*, 4(4):297 – 318, 1995.

[18] Urs Hengartner and Peter Steenkiste. Protecting Access to People Location Information. In *Proceedings of First International Conference on Security in Pervasive Computing (SPC 2003)*, Boppard, Germany, March 2003.

[19] Urs Hengartner and Peter Steenkiste. Implementing Access Control to People Location Information. In *Proceedings of 9th Symposium on Access Control Models and Technologies (SACMAT 2004)*, Yorktown Heights, NY, June 2004.

[20] Jason I. Hong and James A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of MobiSys 2004*, June 2004.

[21] B. Krishnamurthy and D. Rosenblum. Yeast: A general purpose event-action system. *IEEE Transactions on Software Engineering*, 21(10), October 1995.

[22] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The Information Bus - An Architecture for Extensible Distributed Systems. *Operating Systems Review*, 27(5):58 – 68, December 1993.

[23] L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe systems. In *Proceedings of the 10th USENIX Security Symposium*, pages 281–295, August 2001.

[24] Lukasz Opyrchal. *Content-Based Publish Subscribe Systems: Scalability and Security*. PhD thesis, University of Michigan, Ann Arbor, 2004.

[25] Lukasz Opyrchal, Mark Astley, Joshua Auerbach, Guruduth Banavar, Robert Strom, and Daniel Sturman. Exploiting ip multicast in content-based publish-subscribe systems. In *Proc. of Middleware 2000*, April 2000.

[26] PreCache. http://www.precache.com.

[27] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, October-December 2002.

[28] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, Brisbane, Australia, September 1997.

[29] Dale Skeen. Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview. Technical report, Vitria Technology Inc., 1996. http://www.vitria.com.

[30] *TIBCO Messaging Solutions*. http://www.tibco.com/ software/enterprise_backbone/messaging.jsp.

[31] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security Issues and Requirements for Internet-Scale Publish Subscribe Systems. In *Proceedings of the HICSS-35*, Big Island, Hawaii, January 2002.