Access Control to Information in Pervasive Computing Environments

Urs Hengartner and Peter Steenkiste
Carnegie Mellon University
{uhengart,prs}@cs.cmu.edu

Abstract

Many types of information available in a pervasive computing environment, such as people location information, should be accessible only by a limited set of people. Some properties of the information raise unique challenges for the design of an access control mechanism: Information can emanate from more than one source, it might change its nature or granularity before reaching its final receiver, and it can flow through nodes administrated by different entities. We propose three design principles for the architecture of an access control mechanism: (1) extract pieces of information in raw data streams early, (2) define policies controlling access at the information level, and (3) exploit information relationships for access control. We describe an example architecture in which we apply these principles. We also report how our earlier work about adding access control to a people location service contributed to the more general access control architecture proposed here.

1 Introduction

Pervasive computing environments, such as the ones studied in CMU's Aura project [4], provide many kinds of information. Some of this information should be accessible only by a limited set of people. For example, a person's location is a sensitive piece of information, and releasing it to unauthorized entities might pose security and privacy risks. For instance, when walking home at night, a person will want to limit the risk of being robbed, and only people trusted by the person should be able to learn about her current location.

The access control requirements of information available in a pervasive computing environment have not been thoroughly studied. This information is inherently different from information such as files stored in a file system or objects stored in a database, whose access control requirements have been widely studied. In this paper, we discuss these differences in detail. This discussion leads to the proposal of three design principles for the architecture of an access control mechanism deployed in a pervasive computing environment. Namely, these design principles suggest: (1) extract pieces of information in raw data streams early, (2) define policies controlling access to information at the information level, and (3) exploit information relationships when performing access control.

Let us illustrate the unique challenges for access control with an example scenario. Figure 1 shows how individuals can locate other people and free food in a pervasive computing environment. To simplify our discussion, we introduce the following notation: We call entities from which original information emanates *source nodes* (file system, access point, camera, and instant messaging client in the example), and entities that are final recipients of information *sink nodes* (Bob). For a pair of entities between which there is a direct information flow (e.g., laptop locator to people locator and camera to face detector), we call the entity from which the information emanates *server node* and the entity receiving the information *client node*. Each source node is always a server node, and each sink node is always a client node. The notation {foo; bar} denotes that a server node reports information "bar" about item "foo" to a client node, whereas the item is optional.

Based on Figure 1, we identify the following challenges for performing access control to information in a pervasive computing environment:

- A piece of information, like a person's location, can be derived from other pieces of information, like a camera picture or the location of her laptop, and multiple source nodes can contribute to a single piece of information. The source nodes can be of different types: they can be devices like sensors or cameras, or they can be of digital nature like files or instant messages. Thus there might be no single point, like a file in a filesystem or an object in a database, where we can deploy access control to protect the information. Instead, we have to run access control in a distributed way, depending on how

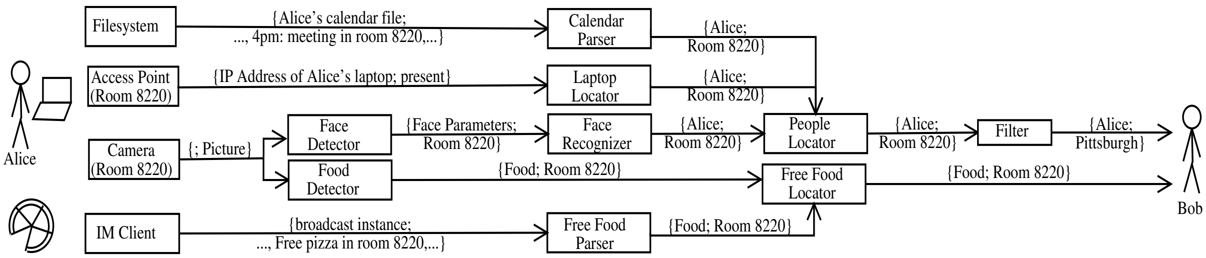

Figure 1: *Retrieval of location information about people and free food. The people locator gathers people location information from various nodes. The calendar parser extracts scheduling information from a user's calendar file. The laptop locator returns the location of the wireless access point to which a user's laptop is connecting. The face recognizer maps between a person and a set of face parameters used by the face detector. The face detector detects faces in pictures delivered to it by cameras in a building. The free food locator locates free food by exploiting information from the food detector and the free food parser. The food detector detects food in camera pictures. The free food parser scans broadcast messages posted to an instant messaging application for messages announcing free food. The filter reduces the granularity of Alice's location information before giving it to Bob.*

information is gathered and processed.

- Information can change in nature (e.g., a picture becomes becomes a user/location pair) or its granularity (e.g., "Room 8220" becomes "Pittsburgh") while flowing from the source to the sink node. The access control mechanism needs to be aware of these changes. For instance, it should be possible to grant only a small set of people access to a piece of information provided at a fine-grained level, whereas a larger set can have access to the same information provided at a more coarse-grained level. Access control to files or objects in a database covers only a single variant of a piece of information, being of a single granularity only.

- The nodes in the environment can be administrated by different entities. Thus access control needs to ensure that information flows only through nodes that are authorized to access the information. A filesystem or a database is typically run by a single administrative entity, and accessing remote filesystems from outside that domain, as, for example, in the case of the Andrew File System (AFS), requires extra configuration beforehand.

In addition to addressing these challenges, the access control infrastructure needs to offer flexible ways for granting entities access to information. Though this flexibility could also be useful for information like files or objects in a database, it becomes more important for information provided in a pervasive computing environment. As mentioned above, we should be able to grant access based on the granularity of information. We also require flexibility along the following lines:

- We should be able to grant access to an individual based on her context. An individual's context can consist of the current time, her current location, or her current activity. For example, Alice can allow Bob to access her location only during office hours.

- Depending on the environment, different entities should be able to grant access to information. For example, in a military environment, a central authority must perform this task, whereas in a university environment, individuals should be able to grant access to their personal information.

- Individuals should be able to let other people grant access for them. For example, an individual can have her doctor decide who should have access to her medical information, such as a sensor measuring her heart rate.

In the rest of this paper, we elaborate on how these challenges and requirements affect the design of an access control architecture deployed in a pervasive computing environment. In Section 2, we introduce three design principles for such an architecture. In Section 3, we present the design of our actual architecture. In Section 4, we discuss how our experience gained in designing and implementing an access control mechanism for a people location service contributed to the proposed design principles and architecture. In Section 5, we discuss related work. We conclude the paper in Section 6.

2 Design Principles

In this section, we elaborate on three principles that we have followed in our design of an access control architecture for information available in a pervasive computing environment.

2.1 Extract Information Early

This principle suggests that for raw data streams, such as a videostream, the pieces of information available in the stream should be extracted as early as possible. Access to the raw data streams should be strictly limited and granted only to the entity doing the extraction. For example, when exploiting cameras for locating people or free food, only the face and food recognizers should have access to the pictures delivered by the cameras. As soon as information has been extracted from a raw data stream, more entities can be granted access to the extracted information. However, these entities should not get access to the raw data stream, which has a lot of other information in it. For example, a person can be granted access to her location information as produced by the face recognizer, but not to the raw pictures. Similarly, each person in the organization can be granted access to the location of free food as produced by the free food parser, but not to the raw pictures.

2.2 Define Policies at Information Level

This principle suggests that users should not have to issue policies controlling access to information at the level of individual nodes. Instead, they should be able to define these policies at the information level. The principle is based on the observation that a pervasive computing environment consists of a lot of nodes. Therefore, the environment is going to require a lot of policies to control what nodes should have access to what information offered by other nodes. To reduce the burden on individual users, we do not have them issue low-level statements about nodes. Instead, we let them make high-level statements about information and relationships between information. For example, Alice should be able to state that "Bob can access my location information" and "Use the location of my laptop for locating me". She should not have to declare that "Bob can access my location information as provided by the people locator, which is derived from the location of my laptop as provided by the laptop locator" and "The people locator can access the location of my laptop as provided by the laptop locator". The high-level statements should be independent of the information flow, that is, users should not have to know about the architecture of the system and how information flows through the various nodes.

2.3 Exploit Information Relationships

This principle suggests that information relationships, as proposed by the second design principle, should be taken into account for access control. Such relationships can be exploited in different ways. We describe two examples. First, if there is a relationship specifying that the location of Alice's laptop should be used for locating her, the laptop locator will grant the people locator access to location information about Alice's laptop. This example is a straightforward mapping from a high-level information relationship to a low-level access control policy. Second, we can require that the people locator proves to the laptop locator that Bob has actually asked for the location of Alice and that Bob is entitled to get this information. This additional constraint implies that the people locator will not be able to ask the laptop locator for Alice's location unless it can present a valid request for the location of Alice. Therefore, we minimize damage in case of a break-in into the people locator; intruders will not be able to ask the laptop locator for the location of Alice since they are not able to generate a valid request for Alice's location (unless they explicitly have the right to generate such a request). This example shows that by exploiting additional knowledge available in an information relationship, we can make the access control process more robust.

3 Design of Access Control Mechanism

Before forwarding a piece of information from a server to a client node, we need to validate that the client node is entitled to get the information. Two possible models for performing access control are end-to-end and step-by-step. In this section, we first discuss these two models and their application in the design of our access control architecture. We then describe our architecture.

3.1 End-to-End vs. Step-by-Step Model

In the end-to-end model, a source node validates that the sink node and all the nodes between the source and the sink node are authorized to receive the requested piece of information emanating from the source node. The source node then instructs all the nodes on the path to the sink node how to deal with the received information; these instructions can be sent together with the information. The advantage of this model is that access control is done at a single point; there are no redundant access control checks. The drawback of the model is that it puts a heavy load on a source node. The task of performing access control might be too heavyweight for resource-limited source nodes, such as a sensor. In addition, there might be multiple source nodes, so there is no single point at which we can perform access control. Another drawback is that a request for information has to flow through the entire system to the source node before an access control decision is made. Intermediate nodes process the request and potentially translate it

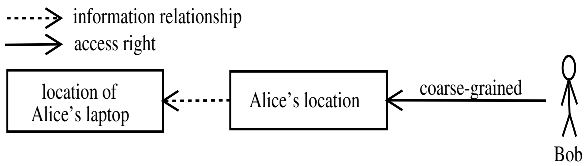

Figure 2: *Example information relationship and access right. Alice's location information is derived from the location of her laptop. Bob has coarse-grained access to Alice's location information.*

into different requests. Therefore, the end-to-end model is more prone to denial-of-service attacks.

In the step-by-step model, for each possible pair of server/client nodes participating in the information flow, the server node validates whether it should give information to the client node. The advantage of this model is that it distributes the access control load over multiple nodes. In addition, an invalid request can be thrown away immediately by the first node receiving the request, thus this method is less prone to denial-of-service attacks. The drawback of this model is that all nodes need to run access control, thus there might be redundant checks. In addition, all nodes need to be able to run access control checks. This requirement is difficult to fulfill when we want to support "dumb" nodes with limited functionality. For example, the filter node shown in Figure 1 might only be able to filter data, but not to run access control.

Our architecture is based on the step-by-step model. We extend the model to support dumb nodes by having them delegate the access control check to a node that is able to perform access control.

Let us now discuss how we apply this access control architecture in the example scenario shown in Figure 1. We limit our discussion to a subset of the nodes; the remaining nodes run access control in a similar way. The nodes uses the information relationship rule and the access right associated with Alice's location shown in Figure 2. Note that the figure does not determine the identity of the entity making these associations. Depending on the environment, this entity can be a central authority, or it can be Alice, whereas Alice can delegate this task to some other entity. Also note that we have to ensure that only valid information relationships are exploited in the access control process. For example, Alice should not be able to state that Bob's laptop should be used for locating her.

The nodes perform access control as follows: The filter is not able to perform any access control. It only filters location data as instructed by the people locator, and has the people locator take care of access control. The people locator validates that Bob has access to Alice's location information and at what granularity. It will instruct the filter to reduce the granularity of the information accordingly. It requests location information from the various location services, for example, from the laptop locator. The laptop locator validates that the location of Alice's laptop should be used to determine Alice's location by looking at the information relationship. In addition, as suggested by the third design principle, it can verify whether Bob has really asked for Alice's location information. The access point is a resource-constrained node; its access control ensures only that information is forwarded to the laptop locator, but not to any other nodes.

3.2 Architecture

Our access control architecture consists of two description languages and the actual access control mechanism.

We use the *policy description language* for the specification of policies controlling access to information. A policy lists the entities that are granted access to a particular piece of information. Policies, and thus the description language, should be flexible. For instance, it should be possible to grant access both to single entities and to groups of entities. In addition, policies should support the specification of various constraints (e.g., time intervals) that need to be fulfilled before access is granted. Also, it must be possible to specify the granularity of a piece of information to which access is given. Depending on the environment, different entities should be able to define these policies. The entity can be an individual or a central authority. For some environments, the individuals should have the option to delegate the decision about her access control policies to other entities. We can employ existing certificate-based trust management frameworks (e.g., KeyNote [2] or SDSI/SPKI [3]) to implement this component.

We use the *information description language* to describe information, its properties, and relationships between information. For example, Alice needs to be able to specify the granularity of information to give granularity-based access rights. In addition, if Alice wants her laptop's location to serve as her location, she needs to be able to explicitly define this relationship. Finally, if there are multiple services that provide location information, Alice must be able to define which of them provides her location information.

At each server node participating in an information flow, the access control mechanism decides whether information should flow to the corresponding client node. To make this decision, the access control mechanism will use the policy and information description languages just described. More specifically, the access control mechanism works as follows.

First, the mechanism checks whether there is a policy allowing the information flow from the server node to the client node. In addition, all the constraints stated in the policy must be fulfilled, and the granularity of the information might have to be reduced. The policy description language lets the access control mechanism learn about the content of policies.

Second, the mechanism also exploits information relationships in the access control process, as suggested by the third design principle. Furthermore, it can exploit additional properties of the information. For example, anyone who has access to a piece of information offered at a fine-grained level should also have access to the same piece of information offered at a coarse-grained level. In addition, the mechanism might be able to automatically derive the access control policy of a piece of information derived from other pieces of information. Namely, it can combine the access control policies of these pieces to form the access control policy of the derived piece. For the combination, we can, for example, exploit principles from Myers and Liskov's decentralized label model [10]. The information description language lets the access control mechanism learn about the characteristics of information.

Finally, to avoid snooping attacks on information exchanged between nodes, the mechanism needs to encrypt this information. We cannot employ end-to-end encryption since intermediate nodes have to be able to process the information flowing through them.

4 People Locator Service

We have built a service for locating people and are currently deploying it at Carnegie Mellon [5]. The service uses multiple sources for locating people, namely, login information, calendar information, and device location information. We have included access control in the design of our service, and the experience gathered during its development and implementation has contributed to the architecture proposed in this paper.

We use SPKI/SDSI digital certificates [3] for specifying different types of decisions, such as policies controlling access to information, and performing the access control check. For example, in the following certificate, Alice grants Bob access to her location information at a coarse-grained level only. Bob can locate Alice only if she is at the locations specified in the certificate and during the listed time intervals. (The signature belonging to the certificate is not shown.)

```
(cert   (issuer (public_key_of_alice))
        (subject (public_key_of_bob))
        (tag (policy alice
           (* prefix world.cmu.wean)
           (* set  (monday (* range numeric
                        ge #8000# le #1200#))
                   (tuesday (* range numeric
                        ge #1300# le #1400#)))
           coarse-grained)))
```

The example shows that we can define flexible access control policies in the tag section of a certificate. Therefore, we are planning on employing SPKI/SDSI certificates in the more general access control architecture. But we have also identified a limitation of this mechanism. The fixed sequential ordering of the access control constraints in the tag section limits the type of constraints that users can specify. It is not possible for users to include additional, individual constraints (such as customized filters) in their certificates.

Clients of our service contact the people locator node. To limit risk in case of a break-in, we do not allow this node to actively ask the various source nodes for information. Instead, it can only forward requests received from its clients to them. These nodes return information to the people locator node only if the entity setting up the access control policy for the requested information trusts this node. While this mechanism is sufficient for the people locator service, it is not flexible enough for a more general system. Forwarding requests assumes that each node provides the same type of information; it does not support the case where nodes offer different types of information (e.g., the location of a device instead of the location of a person). In addition, the notion of trust is rather awkward since it is not directly coupled to the information to which access is controlled. Instead, it is coupled to nodes. These shortcomings lead us to propose the second and third design principle.

5 Related Work

Previous work about dealing with intermediate nodes between a source and a sink node includes, for example, Howell and Kotz's quoting gateways [6] and Neuman's proxy-based authorization [11]. In this related work, there is typically only one intermediate node between the source and the sink node, whereas in a pervasive computing environment, there can be an unlimited number of nodes. In addition, in the related work, the intermediate nodes are authorized to ask for data themselves. This authorization presents a risk if the intermediate node is broken into. We try to avoid such risks by exploiting opportunities offered to us by the information description language.

Kagal et al. [8] propose an extended role-based access control model for pervasive computing. (Al-Muhtadi et al. [1] pursue a similar approach.) They extend the model with the notion of delegation and incorporate context-sensitive roles. The authors use the model for controlling access to services like a printer or a projector. In contrast to our work, the authors do not consider preventing individuals from accessing pieces of information provided by a service. For access control, the proposed architecture relies on a centralized trusted entity running a Prolog-like knowledge base. Our architecture runs access control in a distributed way.

Jiang and Landay [7] and Minami and Kotz [9] perform access control to information by assigning tags to pieces of information. A tag denotes the policy for the piece. The tag of a piece of information derived from other pieces of information is automatically computed based on the tags of these pieces. Whereas Jiang and Landay use a technique similar to Myers and Liskov's decentralized label model [10] for this computation, Minami and Kotz deem this model too conservative and propose a more relaxed model. Though automatic derivation of policies can reduce the number of policies that need to be manually specified in a pervasive computing environment, it is not clear how big the actual benefit is. For example, when information changes its type or becomes more coarse-grained, other people or more people than the ones allowed to access the original information might be granted access to the transformed information. In such a case, the new tag cannot be automatically derived. Finally, both Jiang and Landay and Minami and Kotz assume that the environment is administrated by a single entity and that source and intermediate nodes are fully trusted. Thus, in contrast to our model, there is no need to restrict intermediate nodes from accessing information, and the authors address only preventing sink nodes from getting access.

6 Conclusions

In this paper, we examined access control requirements for information available in pervasive computing environments. We identified several challenges, and we presented three design principles that we applied in an example access control architecture. We are currently refining our design. Namely, we are designing a language for the specification of relationships between different pieces of information and of their granularity. This language will help us in defining policies controlling access to information and in automatically deriving access control policies.

Acknowledgments

We thank Mahim Mishra and the anonymous reviewers for their comments. This research was funded in part by DARPA under contract number N66001-99-2-8918 and by NSF under award number CCR-0205266. Additional support was also provided by Intel.

References

[1] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M. D. Mickunas. Cerberus: A Context-Aware Security Scheme for Smart Spaces. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 489–496, March 2003.

[2] M. Blaze, J. Ioannidis, and A. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, September 1999.

[3] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, September 1999.

[4] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, April-June 2002.

[5] U. Hengartner and P. Steenkiste. Protecting Access to People Location Information. In *Proceedings of International Conference on Security in Pervasive Computing (SPC 2003)*, March 2003.

[6] J. Howell and D. Kotz. End-to-end authorization. In *Proceedings of the 4th Symposium on Operating System Design & Implementation (OSDI 2000)*, pages 151–164, October 2000.

[7] X. Jiang and J. A. Landay. Modeling Privacy Control in Context-Aware Systems. *IEEE Pervasive Computing*, 1(3):59–63, July-September 2002.

[8] T. Kagal, L. Finin and A. Josh. Trust-Based Security in Pervasive Computing Environments. *IEEE Computer*, pages 154–157, December 2001.

[9] K. Minami and D. Kotz. Controlling access to pervasive information in the "Solar" system. Technical Report TR2002-422, Dept. of Computer Science, Dartmouth College, February 2002.

[10] A. C. Myers and B. Liskov. Complete, Safe Information Flow with Decentralized Labels. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, May 1998.

[11] B.C. Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *Proceedings of International Conference on Distributed Computing Systems*, pages 283–291, May 1993.