Technical Report

A Presence-aware Trust Management System

Yoshiaki Takata[†,‡], Jun Noda[‡], Mie Takahashi[‡], Hisashi Mouri[†],
Itaru Hosomi[‡], Daigo Taguchi[‡], and Hiroyuki Seki[†,‡]

October 21, 2005

[†]Graduate School of Information Science
Nara Institute of Science and Technology

[‡]Internet Systems Research Laboratories
NEC Corporation

Abstract

We propose a new architecture for trust management in ubiquitous environments that deals with RBAC policy, digital signatures, and user presence in a uniform framework. The proposed architecture includes inferences about user presence from incomplete sensor signals based on the hidden Markov model. We implemented a prototype system for a connection service in an office computer network with an RFID tag sensor system. Experimental results show that the proposed architecture is effective in providing both useful and secure services in a ubiquitous environment.

1 Introduction

Trust management is becoming a key technology in modern computer and network systems. In particular, in ubiquitous (or pervasive) environments, it is not realistic to use only password-based authentication as it is impossible to give accounts to all potential users in advance. Trust management provides not only a user authentication method based on public-key infrastructure (PKI) instead of password authentication, but it also offers a flexible and systematic method for assigning roles, capabilities, or authorizations to a user the system doesn't recognize. This is possible through the use of digital certificates; i.e., the system assigns roles or capabilities to a user based on his/her identity, status, or attribute guaranteed by a business partner or a public authority (e.g., the fact that (s)he is employed by a specified company, the amount of business with a specified company, etc.) instead of maintaining the list of all potential users. This technology also plays an important role in a ubiquitous environment since a user is more likely to access a terminal or a server they have never encountered before. Specifically, context information, including user presence, is crucial for the system when deciding whether to permit a user access to the system's resources, and what kind of service the system should provide for the user [1]. For example, an office system in a company may have a policy that allows a company visitor to access a local area network without a digital signature or password if (s)he is working with a full-time company employee in the same room; otherwise, the visitor would be required to show a digital certificate.

This means that the trust management system should take contextual information into account when assigning appropriate roles to a user. However, a naive combination of an existing trust management system and a sensor subsystem may suffer from an array of problems. For example, it is difficult to assure the correctness or appropriateness of the behavior of the combined system since the system must perform some complex inferences using heterogeneous information such as digital signatures and sensor signals; moreover, information from the sensors is often incomplete or even incorrect.

In this paper, we propose an architecture for presence-aware trust management in a ubiquitous environment. The proposed system has the following advantages:

- Our model provides a policy-based integrated framework for role-based access control (RBAC), PKI, and presence awareness. Our model is based on a policy language such as TPL [8]. Heterogeneous information such as verified digital certificates, user presence given by inference engines, and a user's schedule are all represented as first-order atomic formulas. See section 2 for details. We explain the policy language and the trust policy engine based on this language using a running example in section 3.

- Formal semantics of the system behavior are defined by first-order logic. Hence, the trust management procedure can be fully automated once a trust policy is given. Also, having these formal semantics implies that the behavior of the system can be predicted in the logical framework. Hence, a formal verification of the system described by a given policy is possible. The policy language and trust policy engine based on the language are explained by using a running example in section 3.

- The proposed system makes non-trivial inferences about user presence from incomplete sensor signals based on the hidden Markov model (HMM). Inferences about presence using HMM is described in detail in section 4.

- We implemented a prototype system and empirically evaluated the effectiveness of the proposed architecture. The system performs access controls for connection services in an office computer network, and it uses RFID tag sensor information to infer user presence. We numerically evaluated the prediction power of the presence inference engine using human subjects with RFID tags as sensors. Implementation issues and evaluations based on the experiments are described in sections 5 and 6, respectively.

Related work A few trust management models have been proposed although they differ in what role digital certificates and policy description languages play. In [16], trust management models are classified into property-based models and capability-based models. In the former approach, trust management should be performed independently of the application that will use the results of the trust management inference. Thus, a property-based system uses certificates that mainly certify names and attributes that are independent of any specific organization or system. This includes certificates issued by public organizations. TPL [8] is the best-known property-based trust management model. The capability-based approach also uses certificates that certify authorizations (or roles) assigned to the subject in a specific organization. In this model, the expressiveness of certificates is large. One weakness is that the certificate design generally depends on its applications. Capability-based models include PolicyMaker [3], its descendant KeyNote [4] and SPKI [6]. In [15], the requirements for policy description languages are examined and existing policy description languages including TPL and KeyNote are compared.

The proposed trust management model is a property-based one like TPL. The main contribution of the proposed model is that heterogeneous information such as PKI certificates and user presence are formally and uniformly represented. This means that the application, trust management, and sensor subsystem are well separated. Also, the behavior of the system is completely predicted based on formal semantics, which allows the automatic execution of a trust policy engine and static analysis (model checking) for verifying whether a given policy satisfies a given security goal. To the authors' best knowledge, there has been no trust management model like this so far. As in TPL, a capability-based system can also be described in our model since the semantics of a policy are defined in first-order logic; thus we can define the predicates that represent certificates including role information.

Several researches have already incorporated context information into access control (e.g., [5]). Recently, Bertino, et al. [2] proposed a hierarchical schema language for representing an extension of the RBAC model that deals with spatial and location-based information. However, implementation issues were not reported. CSAC [9] is an approach similar to ours; a service provider determines whether a service request is accepted based on information given by a context provider. They implemented a service portal for train travelers. Our model is more general than CSAC in that context parameters are freely introduced in a policy since our policy language is based on first-order logic.

Inferences of user presence based on Bayesian models have also been studied. RightSPOT [10] is an example of the use of static (non-temporal) models, where only the relationship between each position and the observation signals at that position is used. We expect that the precision of presence inference will be improved by involving models of user behavior, and indeed a few temporal Bayesian models including HMM have been used for presence inference. SmartMoveX [11] is a system based on HMM. The Bayesian filter [7] is a general temporal model that includes HMM, and in which the state space can be continuous. Although HMM does not provide an accurate position of a user in a continuous space, such an accurate information is excessive for our motivation on the trust management. Moreover, HMM allows us to introduce a duration time distribution in each state, which improves the precision of the presence inference (cf. Section 6.4.2). The model used in our method is an HMM augmented with observable variables including time intervals and user schedules; thus, it is not an instance of Bayesian filters but an instance of the dynamic Bayesian networks (DBN). Patterson et al. [12]

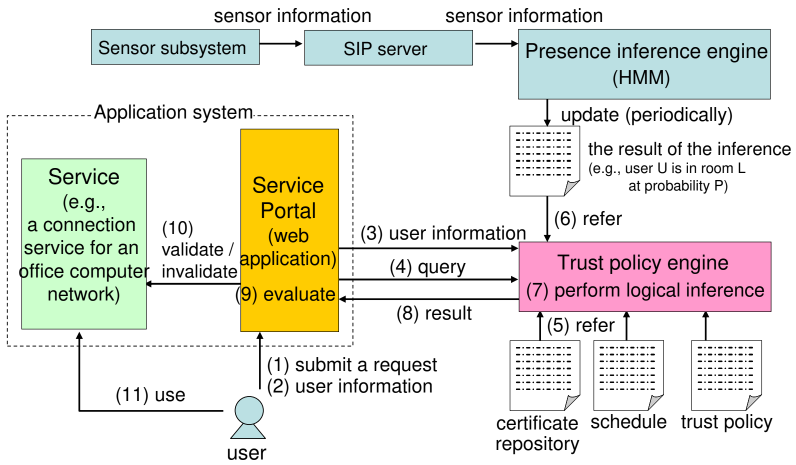

Figure 1: Architecture of trust management system

.

used a DBN for GPS-based presence inference, where a non-observable variable 'transportation mode' ($\in \{\text{bus}, \text{foot}, \text{car}\}$) was involved. Since they were motivated by an outdoor application, their model is quite different from ours.

2 Presence-Aware Trust Management System

2.1 Overview

Figure 1 shows the proposed architecture of the presence-aware trust management system. The trust policy engine assigns roles, capabilities, or authorizations to the user; the application system decides what kind of service it should provide the user based on these roles.

The main feature of the proposed model is that the trust policy engine takes into account not only digital certificates but also user presence information generated by a presence inference engine. This feature enables us to provide a flexible security policy in the following way.

Design issues for a ubiquitous system involve a trade-off between security assurance and usability. The more certificates a system requires a user to present, the less convenient it is for the user to obtain service from the system. Furthermore, the optimal solution to this trade-off often changes depending on several reason, e.g., the decision made by the organization. The proposed model offers both a security-sensitive design and usability-sensitivity by providing a policy that appropriately utilizes user presence in the following way. If we prefer to improve security assurance, for example, the system can use presence information for authentication in addition to a password or digital signature. Specifically, it is easy to implement a policy in which a user is not permitted to access a resource unless the confidence value of his/her presence in a specific location is beyond a certain value. Alternatively, user presence can be used to improve usability. If we would like to temporarily transfer our authorization to another person who works closely with us, we can implement a policy that allows both users to access a resource if the confidence value of their being in the same workspace is beyond a certain value.

2.2 Presence inference engine

The presence inference engine takes sensor information generated by the sensor subsystem. A sensor signal is not always correct and it is desirable to have a systematic way of estimating the real value of the sensor signal. To overcome the problem, the inference engine uses a hidden Markov model (HMM). The HMM state set represents all the possible user presences under consideration. For example, the state set might be $\{\text{office}, \text{meeting\_room}, \text{cafeteria}, \text{home}\}$. The set of output symbols represents all possible sensor signals. Each time an output symbol (i.e. a sensor signal) is given, the presence engine calculates (posterior) probability for each state s the user may be in s. To provide a more refined stochastic model, we also use time intervals $\{\text{AM}, \text{lunch}, \text{PM}, \text{off}\}$ and user schedules as observable variables, and we introduce micro states to represent time duration in a certain place (e.g. the average time of a meeting is forty-five minutes). A detailed description is presented in section 4.

2.3 Trust policy engine

A policy given to the trust policy engine is called a trust management policy or *trust policy*. It is a set of inference rules in the form of definite Horn clauses (including logical facts) representing the information described in section 2.1. The engine makes logical inferences starting from a query generated from a user's access request by the application system. A typical query is 'Can user A log-in to the local area network *wlan*1 without showing digital certificates?' The next section describes the trust policy engine in more detail.

3 Trust policy Engine

3.1 Trust policy

Predicates appearing in a trust policy are divided into basic predicates and policy predicates:

Basic predicates:

- Predicates that represent the results of the presence inference engine (posterior probability for each HMM state).
  Example: $position(U, L, P)$ (user U is in room L at probability P).
- Predicates that represent the existence of a verified digital certificate.
  Example: $cert(N, U, \textit{full-time})$ (There is a verified certificate issued by company N that certifies that U is a full-time employee of N.)
- Predicates that represent information written in a user's schedule book.
  Example: $schedule(U, NY)$ (according to the user's schedule book, user U is now on leave in New York.)

Policy predicates (user defined predicates): Predicates that represent the assignment of roles to users.
Example: $permit(N, U, wlan1, passwd)$ (Company N gives user U the right to access the local area network $wlan1$ if U is authenticated by a password.)

A policy predicate is defined by recursive inference rules based on basic predicates. Sometimes we may like to assign a role to a user with a certain confidence level (or probability value). For example, for a user indicating a certificate issued by a more trusted authority, we may assign a role to that user

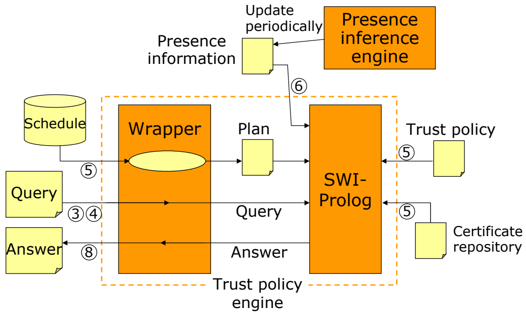

Figure 2: Structure of trust policy engine

with a higher confidence level. Such a value can also be easily represented as an argument of a policy predicate.

3.2 Trust policy Engine

A trust policy engine is an inference engine deriving roles (represented by policy predicates) that can be assigned to the user with an appropriate confidence level as a response to a query given by the user. A trust policy engine can be implemented as an extended Prolog interpreter. The extension (called *wrapper*) first receives a query from a user (or a service application that wants to decide whether a query from a user is allowed). The wrapper then generates specifications of (1) presence information with a confidence level, (2) verified certificates provided by the user, and (3) the user's schedule as Prolog facts. Next, the wrapper makes inferences using the generated Prolog facts, the current trust policy, and the given query as a Prolog program (Fig 2).

When a service provider needs to decide whether a query (or service request) from a user should be permitted, it consults the trust policy engine to see whether the corresponding policy predicate $permit(\ldots)$ is true. That is, a trust policy specifies the conditions for allowing each service.

Since user presence is given as a basic predicate, the trust policy engine can derive roles depending on presence information such as 'user A can access the LAN $wlan1$ without a digital signature or password' when A is with a full-time employee in the same room whereas 'A is required to show a digital certificate to access $wlan1$ when A is alone in the room.'

3.3 A policy example

We first choose basic and policy predicates that are sufficient for specifying a policy under consideration. Basic predicates are chosen depending on what kind of information the wrapper can collect. On the other hand, policy predicates do not depend on the wrapper, though an application (i.e. a query issuer) and a policy designer should agree in advance which policy predicates will be used.

Below we present a service policy in an office environment as an example. Suppose that there are two kinds of wireless LAN, a high-level wireless LAN and a low-level wireless LAN, where accessing the high-level wireless LAN needs more security assurance than accessing the low-level wireless LAN. The policy would include the following individual rules:

(1) A full-time employee (i.e. who has an employee ID card) can access the high-level wireless LAN if one of the probabilities that the employee is in the office room, meeting room, or laboratory

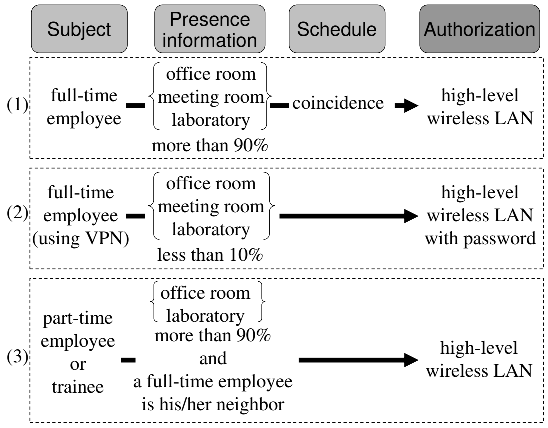

Figure 3: A policy example

is more than 90% and the real position of the employee is the same as the place written in the employee's schedule.

(2) A full-time employee can access the high-level wireless LAN with password authentication if the employee is out of the company (if all the probabilities that the employee is in the office room, meeting room, or laboratory are less than 10%) and uses VPN.

(3) A part-time employee or a trainee can access the high-level wireless LAN if one of the probabilities that (s)he is in the office room or laboratory is more than 90% and a full-time employee is his/her neighbor.

Figure 3 summarizes the above policy.

For example, rule (1) is specified as follows:

```
permit(N,U,wlan1,normal) :-
  cert(N,U,full-time),
  position(U,L,P), room_for_ft(L),
  90 < P, schedule(U,L).
```

The forth argument of $permit$ represents additional information about the service; $normal$ denotes that the connection is allowed without authentication and $passwd$ indicates that password authentication is required. The predicate $room\_for\_ft(L)$ is true when L is in office room, meeting room or laboratory. This predicate is defined as follows:

```
room_for_ft(office).
room_for_ft(meeting_room).
room_for_ft(laboratory).
```

The following query asks whether *ichiro* can use the service *wlan1*.

```
?- permit(nec, ichiro, wlan1, normal).
```

Rule (2) needs checking whether the user uses VPN. If so, the application issuing a query should provide that fact to the trust policy engine using an appropriate basic predicate. In this example, we use a predicate *parameter* to represent such additional information given to the trust policy engine, and the application specifies such information in a query specification as follows.

```
parameter(jiro, connection, vpn).
?- permit(nec, jiro, wlan1, normal).
```

Rules in a trust policy uses *parameter* as the same as the other basic predicates as follows.

```
permit(N,U,wlan1,passwd) :- cert(N,U,full-time),
                            parameter(U,connection,vpn).
```

Appendix A shows a trust policy corresponding with rules (1) – (3).

3.4 Output of the reasoning process

It is desirable that the system can output not only whether a service is allowed, but also why it is allowed or prohibited. In the prototype system, we have extended the policy described above so that history of the inference can be outputted together with the answer to a query.

First, we consider a method for outputting the history when a query is satisfiable (i.e., a reason why service is allowed). We implement it as outputting the ID of the rule used to prove the query satisfiable. For example, if a query is proved satisfiable by rule (1) in the above policy, the reason why the service is allowed is that the subject is a full-time employee who is in the planned position. For each inference rule defining *permit*, we specify a term consisting the ID of the rule and auxiliary data in the fifth argument of *permit* as follows:

```
permit(N,U,wlan1,normal,fulltime_employee_in_planned_position(L,P))
:- cert(N,U,full-time),
   position(U,L,P), room_for_ft(L), 90 < P,
   schedule(U,L).
```

By the following query, the reason why the service is allowed is substituted for variable H and is outputted by the built-in predicate *write*.

```
?- permit(nec,user1,wlan1,normal,H), write(H).
```

Assume that this query outputs, e.g., 'fulltime_employee_in_planned_position(meeting_room, 98).' This means that the service is allowed because the subject is in the meeting room with probability of 98% and this presence information conform the inference rule named fulltime_employee_in_planned_position. The query issuer and the policy designer should agree on the set of rule IDs and the meaning of the auxiliary data (e.g., meeting_room and 98) in advance. There are in general more than one inference rules proving a query satisfiable. When we want to output the IDs of all these rules, we use a built-in predicate *fail* as follows:

```
?- permit(nec,user1,wlan1,normal,H), write(H), nl, fail.
```

Next, we consider a method for outputting the history when a query is not satisfiable (i.e., a reason why service is denied). We use a predicate *permit_or_not* instead of *permit*. In the first five arguments, *permit_or_not* is the same as *permit*. The sixth argument of *permit_or_not* is the list of the reasons why the service is denied. If the sixth argument is the empty list, then the service represented by the first four arguments is allowed. An example of inference rules defining *permit_or_not* is as follows:

```
permit_or_not(
   N,U,wlan1,normal,fulltime_employee_in_planned_position(L,P),Rs2)
:- cert(N,U,full-time),
   schedule(U,L), L \= no_plan,
   Rs0=[],
   (room_for_ft(L) -> Rs1=Rs0 ;
     Rs1=['Not a room for full-time employee'(L) | Rs0],
     (position(U,L,P) -> (90 < P -> Rs2=Rs1 ;
       (1=<P -> Rs2=['Confidence level is too low'(U,R,P) | Rs1]) ;
               Rs2=['Subject is not there'(U,L) | Rs1]))).
```

Although this rule is essentially the same as the above-mentioned inference rule defining *permit*, program code outputting $Rs2$ (the list of the reasons why the service is denied) is added to the right-hand side. For each sub-goal in the right-hand side, if the sub-goal is false, then intuitive meaning of the sub-goal is added into the list. In the above example, if room_for_ft(L) is true, then $Rs1$ is the same as $Rs0$. Otherwise, $Rs1$ is defined as the list obtained by adding a string 'Not a room for full-time employee' into $Rs0$. Built-in predicate A –> B ; C means that B is evaluated if A is true and C is evaluated otherwise. Fundamentally, we only have to substitute an expression (f –> $Rs_j = Rs_i$; $Rs_j = [\text{reason}|Rs_i]$) for each sub-goal f in an inference rule defining *permit*. However, for instance, if $\mathit{position}(U, L, P)$ in the above example is false, i.e., if there is no P which make $\mathit{position}(U, L, P)$ true, then any sub-goal about P cannot be evaluated. If g is a sub-goal that can be evaluated only when f is true, then we substitute (f –> (g –> B ; C) ; D) for expression f, g.

Although inference rules defining *permit_or_not* can automatically be derived from inference rules defining *permit*, at present we need to specify the rules by hand. Automatic derivation of inference rules defining *permit_or_not* is one of future works.

4 Presence Inference Engine

As mentioned in Section 2.2, we define a stochastic model about a user's presence, and infer their presence based on the model and observed sensor signals. The basic design of the stochastic model is as follows. User's (real) presence at a time instance is a random variable over a given set of positions (e.g. $\{\text{office}, \text{meeting\_room}, \text{cafeteria}, \text{home}\}$). While the real presence is not observable, information that correlates to the presence (e.g. sensor signals, user's schedule, and current time) is observable. Moreover, the real presence changes as time goes on, and the current presence correlates to the previous presence. These correlations are depicted by Figure 4, where each ellipse denotes a random variable, and each arrow denotes the existence of a correlation. The magnitude of each correlation is defined as the conditional probability between random variables.

The hidden Markov model (HMM)[13] is one such stochastic model, where the value of a non-observable variable changes as time goes on and is inferred based on the values of observable variables. Below, we briefly review the definition of HMM, and present an extension of HMM for inferring a user's presence using time intervals and the user's schedule as well as sensor signals.

4.1 Hidden Markov model (HMM)

HMM is a stochastic model that has a similar structure to finite state machines. Formally, HMM is a 5-tuple $M = (S, T, \delta, \phi, I)$ where:

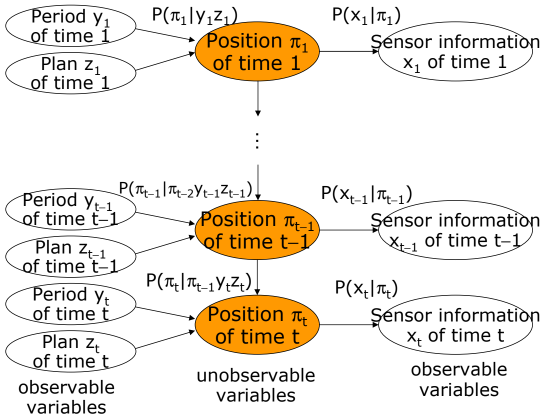

Figure 4: Probability model for presence inference

- S is a finite set of *states*,
- T is a finite set of *output symbols*,
- $\delta : S \times S \to \mathcal{R}$, where $\mathcal{R}$ is the set of real numbers, is *state transition probability distribution*. i.e., $\delta(s, s')$ is the probability of the state transition from s to s'. For each $s \in S$, $\sum_{s' \in S} \delta(s, s') = 1$.
- $\phi : S \times T \to \mathcal{R}$ is *output probability distribution*. i.e., $\phi(s, a)$ is the probability that output symbol a is observed in state s. For each $s \in S$, $\sum_{a \in T} \phi(s, a) = 1$.
- $I : S \to \mathcal{R}$ is *initial state distribution*. i.e., $I(s)$ is the probability that the model is in state s at the starting time instance. $\sum_{s \in S} I(s) = 1$.

We sometimes use the random variable π_t that denotes the state of the model at a time instance t.

Posterior probability $p_s(t) = \Pr(\pi_t = s \,|\, x_1 \dots x_t)$ is the probability that the model is in state s at time instance t given that a sequence $x_1 x_2 \dots x_t \in T^*$ (where T^* is the Kleene closure of T) is observed. The probability $p_s(t)$ can be computed using the following equations (called *forward algorithm*).

$$\begin{aligned}
f_s(t) &= \Pr(x_1 \dots x_t,\ \pi_t = s) \\
&= \begin{cases} \phi(s, x_t) \sum_{s' \in S} f_{s'}(t-1)\delta(s', s) & t \geq 1, \\ I(s) & t = 0, \end{cases} \\
p_s(t) &= f_s(t) / \Pr(x_1 \dots x_t) = f_s(t) / (\sum_{s' \in S} f_{s'}(t)).
\end{aligned}$$

Note that the probability $f_i(t)$ can be computed using the HMM and x_t observed at the current time instance and $f_k(t-1)$ at the previous time instance; that is, we do not have to memory either $x_{t'}$ or $f_k(t'-1)$ for any $t' < t$.

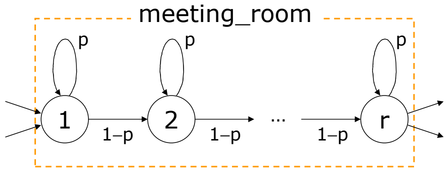

Figure 5: Micro states

4.2 Duration distribution and micro states

In the above-mentioned definition, the length of a stay in the same state follows a geometric distribution. However, when we use HMM to infer a user's presence, it may be unnatural to assume that the probability that will user stay in a room (e.g. a meeting room) for 3 minutes is greater than the probability that (s)he will stay there for one hour.

The use of micro states is a simple approach for representing a hill-shaped duration distribution. A *macro state* is a subset of states that have the same interpretation. In the application of the presence inference, we assign a macro state for each position l, and consider that a user is in l whenever the model is in any member of the macro state. Each member of a macro state is called a *micro state*. The structure of a macro state is depicted by Figure 5. A macro state is defined by two parameters: the number r of micro states and the probability p of the self loop in every micro state. There exists exactly one entrance micro state and exactly one exit micro state of a macro state. The probability $\Pr(d_l = k)$ that the model stays in the macro state l for k time instances satisfies the following equation, which follows a negative binomial distribution:

$$\Pr(d_l = k) = \begin{cases} 0 & k < r, \\ \binom{k-1}{k-r} p^{k-r}(1-p)^r & k \geq r. \end{cases}$$

The expected value and the variance of the duration d_l are $E[d] = r/(1-p)$ and $V[d] = rp/(1-p)^2$, respectively.

4.3 HMM for presence inference

State transition probability distribution As shown in Figure 4, the state π_t at time instance t is decided based on the state π_{t-1} at the previous time instance as well as the current period y_t and the scheduled plan z_t. Thus, the state transition probability distribution should be given by the conditional probability $\Pr(\pi_t \,|\, \pi_{t-1}, y_t, z_t)$ for each quadruple $(\pi_t, \pi_{t-1}, y_t, z_t)$. However, since the size of the domain of the quadruple is very large for practical applications, it is not appropriate to define the conditional probability by hand. Instead, we introduce a method for compounding independently defined probabilities $\Pr(\pi_t \,|\, \pi_{t-1})$, $\Pr(\pi_t \,|\, y_t)$, and $\Pr(\pi_t \,|\, z_t)$ into one probability $\Pr(\pi_t \,|\, \pi_{t-1}, y_t, z_t)$ using the following equation, where α and β are given weights.

$$\Pr(\pi_t \,|\, \pi_{t-1}, y_t, z_t) = \alpha \Pr(\pi_t \,|\, \pi_{t-1}) + \beta \Pr(\pi_t \,|\, y_t) + (1-\alpha-\beta) \Pr(\pi_t \,|\, z_t), \qquad (1)$$
$$0 \leq \alpha,\ 0 \leq \beta,\ \alpha + \beta \leq 1.$$

Note that α and β are not necessarily constants and can be functions of the observable variables y_t and z_t.

Duration distribution As mentioned in Section 4.2, we use micro states to represent the distribution of the length of a stay. For each position or macro state l, the mean μ_l and the standard deviation σ_l of the length of a stay in l are given by an administrator of the presence inference engine. The number r_l of micro states and the probability p_l of the self loop on each micro state of the macro state l are $r_l = \lceil (1-p_l)\,\mu_l \rceil$ and $p_l = \sigma_l^2/(\sigma_l^2+\mu_l)$, respectively.

In the following, we let the random variable π_t range over the macro states, and use a new random variable ρ_t ranging over the micro states. For each micro state s, let $[s]$ be the macro state that s belongs to, $[s]_{\text{ent}}$ be the entrance micro state of $[s]$, and $[s]_{\text{exit}}$ be the exit micro state of $[s]$.

The move probability distribution $\Pr(\pi_t \mid \pi_{t-1})$ is given by the administrator, and the conditional probability $\Pr(\rho_t \mid \rho_{t-1})$ is automatically defined as follows (cf. Figure 5):

$$\Pr(\rho_t \mid \rho_{t-1}) = \begin{cases} \Pr([\rho_t] \mid [\rho_{t-1}])\,(1-p_{[\rho_t]}) & \\ & [\rho_t] \neq [\rho_{t-1}],\ \rho_t = [\rho_t]_{\text{ent}},\ \rho_{t-1} = [\rho_{t-1}]_{\text{exit}} \\ 1-p_{[\rho_t]} & [\rho_t] = [\rho_{t-1}] \text{ and } \rho_t \text{ is the next state of } \rho_{t-1} \\ p_{[\rho_t]} & \rho_t = \rho_{t-1} \\ 0 & \text{otherwise.} \end{cases}$$

Equation (1) is extended to the probability between micro states using the following equation, where $\Pr(\pi_t \mid y_t)$ and $\Pr(\pi_t \mid z_t)$ possibly cause a move from the middle of a macro state into the entrance of another macro state (by the first line of the right hand side), and they also possibly cause a stay in the macro state by either staying at the same micro state or stepping into the next micro state (by the second and third line of the right hand side):

$$\begin{aligned} &\Pr(\rho_t \mid \rho_{t-1}, y_t, z_t) = \\ &\quad \begin{cases} \alpha \Pr(\rho_t \mid \rho_{t-1}) + \beta \Pr([\rho_t] \mid y_t) + (1-\alpha-\beta)\Pr([\rho_t] \mid z_t) & \\ & [\rho_t] \neq [\rho_{t-1}],\ \rho_t = [\rho_t]_{\text{ent}} \\ (\alpha + \beta \Pr([\rho_t] \mid y_t) + (1-\alpha-\beta)\Pr([\rho_t] \mid z_t))\Pr(\rho_t \mid \rho_{t-1}) & \\ & [\rho_t] = [\rho_{t-1}],\ \rho_{t-1} \neq [\rho_t]_{\text{exit}} \\ \alpha\, p_{[\rho_t]} + \beta \Pr([\rho_t] \mid y_t) + (1-\alpha-\beta)\Pr([\rho_t] \mid z_t) & \\ & \rho_t = \rho_{t-1} = [\rho_t]_{\text{exit}} \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Output probability distribution As in Figure 4, we assume that the observed output symbol or sensor signal x_t depends only on the position π_t. The administrator gives the output probability distribution $\Pr(x_t \mid \pi_t)$.

Inference by the forward algorithm The forward algorithm described in Section 4.1 is extended to the above-mentioned HMM for presence inference. Given an observation sequence $x_1 \ldots x_t, y_1 \ldots y_t$, and $z_1 \ldots z_t$, the posterior probability $p_s(t) = \Pr(\rho_t = s \mid x_1 \ldots x_t, y_1 \ldots y_t, z_1 \ldots z_t)$ for each micro state s is computed using the following equation:

$$\begin{aligned} f_s(t) &= \Pr(\rho_t = s,\ x_1 \ldots x_t \mid y_1 \ldots y_t, z_1 \ldots z_t) \\ &= \begin{cases} \Pr(x_t \mid \pi_t = [s]) \sum_{s' \in S} f_{s'}(t-1)\Pr(\rho_t = s \mid \rho_{t-1} = s',\ y_t, z_t) & \\ & t > 0, \\ I([s]) & t = 0 \text{ and } s = [s]_{\text{ent}}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

$$
\begin{aligned}
p_s(t) &= \Pr(\rho_t = s \,|\, x_1 \ldots x_t, y_1 \ldots y_t, z_1 \ldots z_t) \\
&= f_s(t) / \Pr(x_1 \ldots x_t \,|\, y_1 \ldots y_t, z_1 \ldots z_t) = f_s(t)/(\sum_{s' \in S} f_{s'}(t)).
\end{aligned}
$$

5 Implementation

We implemented a prototype system providing a connection service for an office wireless LAN as an application of the proposed method. Access control in connection services is very important for the network security, and thus effective authentication is required. On the other hand, one may think that the current access control mechanisms in ubiquitous environments lack flexibility. Our prototype permits a user to access the office computer network without a password or digital signature if (s)he is judged as having a high enough confidence level to access the network in relation to the user presence inferred by the presence inference engine, which is based on the sensor system and his/her schedule information. The presence inference engine uses a Bayesian model to evaluate the confidence level of user presence. Moreover, the system dynamically closes the user's connection to the network if the confidence level of the user's presence drops below the threshold defined in this service policy. This policy reduces the risk caused by a situation where a user retains authorization to permanently access the network once (s)he obtains it.

5.1 System component

The prototype system determines whether it should give a user the authorization to use the connection service in a wireless office LAN. To do this, the system consults the trust policy engine that makes logical inferences using the presence information generated by the presence inference engine. Implementation issues for the trust policy engine and the presence inference engine are described in sections 5.1.1 and 5.1.2, respectively. Components of the system is depicted in Figure 1. The presence inference engine works with an external sensor system using an RFID tag.

The prototype system is a service portal implemented as a web application. A user can request to use a service via a web browser and can get results from the trust policy engine in the GUI interface. Components of the system and their functions are shown in Figure 1. The flow between components is as follows.

(1) A user submits a request to the service portal.

(2) If possible, the user provides information such as digital certificates to the service portal.

(3)(4) The service portal passes the received information in (2) to the trust policy engine and asks if the user can access the service.

(5)–(7) The trust policy engine makes a logical inference by referring to the digital certificates received in (3), the previously received digital certificates (cert.pl), the user's schedule (plan.pl), the result of the presence inference engine (pos.pl), and the trust policy for the connection service (policy.pl).

(8) The service portal receives a result, which shows whether the user is permitted to use each service.

(9)(10) If the user is permitted to use a service, the service portal validates the use of the service for the user. Otherwise, the service portal invalidates the use of the service.

(11) The user can use a service if they are validated by the service portal.

5.1.1 Trust policy engine

The trust policy engine consists of a Prolog interpreter and an extension (or wrapper) as shown in Figure 2. We use the SWI-Prolog as a Prolog interpreter. The following rules and facts are input to the interpreter:

- a query (described below),
- a trust policy (Section 3),
- the contents of the certificate repository, e.g.,

```
cert(nec,jun,full-time). cert(nec,mie,full-time).
```

- users' positions as stated in their schedules; e.g.,

```
schedule(jun,meeting_room). schedule(mie,meeting_room).
schedule(seki,meeting_room).
```

- presence information; e.g.,

```
position(jun,office,0).
position(jun,meeting_room,93).
position(jun,laboratory,4)....
```

- other information given by an application system; e.g.,

```
parameter(jun,connection,vpn).
```

The trust policy and the contents of the certificate repository are given in advance by the administrator. The position stated in the schedule is extracted from a schedule database whenever the trust policy engine receives a query. The presence information is stored in a file and is periodically renewed by the presence inference engine.

A query defines its goal and other auxiliary facts such as an existence of a certificate. The goal of the query must be named `q`. For example, the following query prints all the services available to a user `seki` when a certificate exists that states (s)he is a temporary employee.

```
cert(nec,seki,temporary).
q :- permit_or_not(nec,seki,S,C,H,R),write([S,C,H,R]),nl,fail.
```

5.1.2 Presence inference engine

The presence inference engine runs independently of the trust policy engine. It periodically checks the sensor signal as well as the current period and the scheduled plan, and computes the *a posteriori* probability based on the given HMM. We implemented a simple database that gives the time interval and the plan for a given time instance. Sensor signals are obtained by periodically asking a sensor system to accepts a request in a Presence Event Package of the Session Initiation Protocol (SIP) [14].

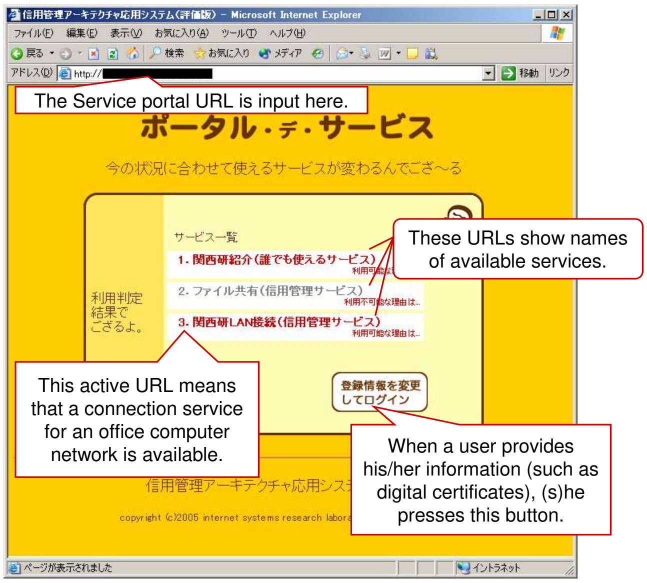

Figure 6: An example of service portal sites

In Session Initiation Protocol, the server accepts a subscribe request, and then sends presence information back to the subscriber whenever the presence changes. Thus, the subscriber (or presence inference engine) has to accept the server's message at any time. For this reason, the presence inference engine consists of the following three concurrent threads: (The second thread is the one that accepts the server's message).

1. A thread that periodically sends a subscribe request to the sensor system.
2. A thread that accepts a message from the sensor system at any time and saves the presence information written in the message in the internal memory.
3. A thread that periodically looks up the internal memory and writes the computed *a posteriori* probability into an external file.

5.2 System behavior

When a user accesses the service portal, (s)he can get a list of services which (s)he can/cannot use in the present context as shown in Figure 6. This is a result of the inference made by the trust policy engine. When the user clicks an active URL, (s)he can use the service corresponding to the URL and get reasons why (s)he can use it. For example, if the user clicks the bottom URL in Figure 6, the service portal permits access to the connection service for the office wireless LAN.

6 Experiments

We conducted a number of experiments to investigate the precision of the output of the presence inference engine. The following two sets of observation sequences were recorded and used as input for the presence inference engine.

Experiment 1. The sequence of sensor signals observed when a subject was moving around in a building for about 30 minutes according to a predefined scenario.

Experiment 2. The sequence of sensor signals observed when a subject performed his/her ordinary work for a single day.

The former is concerned with investigating precision in detail while the latter is concerned with investigation in a realistic setting.

Below we define the measure of precision used in this paper as well as the settings and the results of each experiment.

6.1 Measure of precision

Let N be the length of an observation sequence input into the presence inference engine. From the sequence, the inference engine computes the posterior probability $p_s(t)$ (that the subject is in position s at time instance t) for each s and $1 \leq t \leq N$. Let $sc = s_1 s_2 \ldots s_N$ be the sequence of correct presences, which were defined by the scenario in Experiment 1 and were recorded by the subject in a notebook in Experiment 2.

We define the *precision* of the output of the inference engine as $(\sum_{t=1}^{N} p_{s_t}(t))/N$. Intuitively, the numerator represents the expected value of the number of correct answers if we assume that the inference engine chooses an arbitrary position s with probability $p_s(t)$ for each time t. Divided by the length of the sequence, precision is normalized within a range of between 0 and 1.

6.2 Sensor system

The presence inference engine used in the experiments works with an external sensor system using RFID tags. Each subject is carrying an active RFID tag while moving through a building. There are seven fixed sensors, roughly one per room, on a single floor of the building. Each sensor receives signals emitted by the RFID tags. The control computer (or controller) of the sensor system accepts a request for subscribing to the presence information of a specified subject, and replies with the ID of the sensor that is receiving the signal of the subject's RFID tag. When more than one sensor is receiving the signal, the controller answers the one that is receiving the strongest radio wave signal. When there are no sensors receiving the signal, the controller answers "no signal."

6.3 Design of HMM

The set of HMM output symbols are defined according to the sensor system, i.e., it consists of the seven positions (or rooms) where RFID sensors are placed and one special symbol 'no_signal.' The ranges of the other random variables are defined as follows.

- positions (the range of π_t): a set of the following 12 positions.
  - Seven positions where the sensors are placed: room_301, room_302, refresh_corner, lounge, office, east_office, laboratory.
  - Two positions where no sensor exists: hall, cafeteria.
  - Three virtual positions: another_place_in_building, out_of_building, home.

- Periods (the range of y_t): AM, lunch, PM, off.
- Plans (the range of z_t): meeting_at_room_301, meeting_at_room_302, meeting_at_hall, no_plan.

After a one-hour discussion among a few of the authors, we defined the HMM parameters $\Pr(\pi_t \mid \pi_{t-1})$, $\Pr(\pi_t \mid y_t)$, $\Pr(\pi_t \mid z_t)$, and the mean μ_l and the standard deviation σ_l of the duration distribution so that the parameters would represent the behavior of an ordinary subject in the building. The initial probability was 1 for the office and 0 for the other positions.

The output probability $\Pr(x_t \mid \pi_t)$ was defined as follows. For each room l where a sensor exists, we gave a high probability (more than 0.9) to $\Pr(x_t = l \mid \pi_t = l)$ and a low probability (less than or equal to 0.02) to any other x_t including 'no_signal.' However, room_301 and room_302 were adjacent and the probability that the sensor system would answer the room opposite to the correct one was not marginal. Hence, we gave a relatively high probability (0.13) to $\Pr(x_t = \text{room\_301} \mid \pi_t = \text{room\_302})$ and $\Pr(x_t = \text{room\_302} \mid \pi_t = \text{room\_301})$. We gave the same treatment for the office and east_office. For each place l where no sensor exists, we gave a high probability to $\Pr(x_t = \text{no\_signal} \mid \pi_t = l)$ and a low probability to any other x_t.

The weights α and β in Section 4.3 are defined as $\alpha = 1/(1+r_{\beta,t}+r_{\gamma,t})$ and $\beta = r_{\beta,t}/(1+r_{\beta,t}+r_{\gamma,t})$ where $r_{\beta,t}$ and $r_{\gamma,t}$ are ratios defined in the following way, i.e., we regard the current period and the current plan as important when it is lunch time or when some plan is specified.

$$r_{\beta,t} = \begin{cases} 10 & \text{if } y_t = \text{lunch} \\ 1 & \text{if } y_t = \text{off} \\ 0 & \text{otherwise.} \end{cases} \qquad r_{\gamma,t} = \begin{cases} 10 & \text{if } z_t \neq \text{no\_plan} \\ 0 & \text{if } z_t = \text{no\_plan.} \end{cases}$$

6.4 Experiment 1

6.4.1 Settings

We defined the scenario shown in Figure 7 in which each subject moved through several locations including the office, meeting room, hall, and cafeteria. According to the scenario, six subjects (named a to f) were moving together. The output of the sensor system was recorded every six seconds (0.1 minute). The length N of the input sequence for each subject was 378.

HMM parameters We made the following modifications to the HMM described in Section 6.3. In this experiment, we excluded two virtual positions 'out_of_building' and 'home' from the range of π_t. Moreover, we modified the output probability distribution for two subjects (a and b) as follows, because in this experiment their RFID tags were often missed by the sensor in room 301 and the probability that the sensor system answered 'no_signal' when the subjects were in room 301 was much greater than we expected when designing the HMM system.

$$\Pr(x_t \mid \text{room\_301}) = \begin{cases} 0.5 & \text{if } x_t = \text{room\_301} \\ 0.44 & \text{if } x_t = \text{no\_signal} \\ 0.01 & \text{otherwise.} \end{cases}$$

Because the time spent at each position in the scenario was much shorter than in ordinary behavior, we defined the duration distribution as shown in Table 1. In the table, the mean μ is defined according to the correct duration specified in the scenario. A greater standard deviation $\sigma = 2$ (minutes) was given for the meeting rooms, a medium standard deviation $\sigma = 0.5$ was given for the office, and a small standard deviation $\sigma = 0.1$ was given for the other positions.

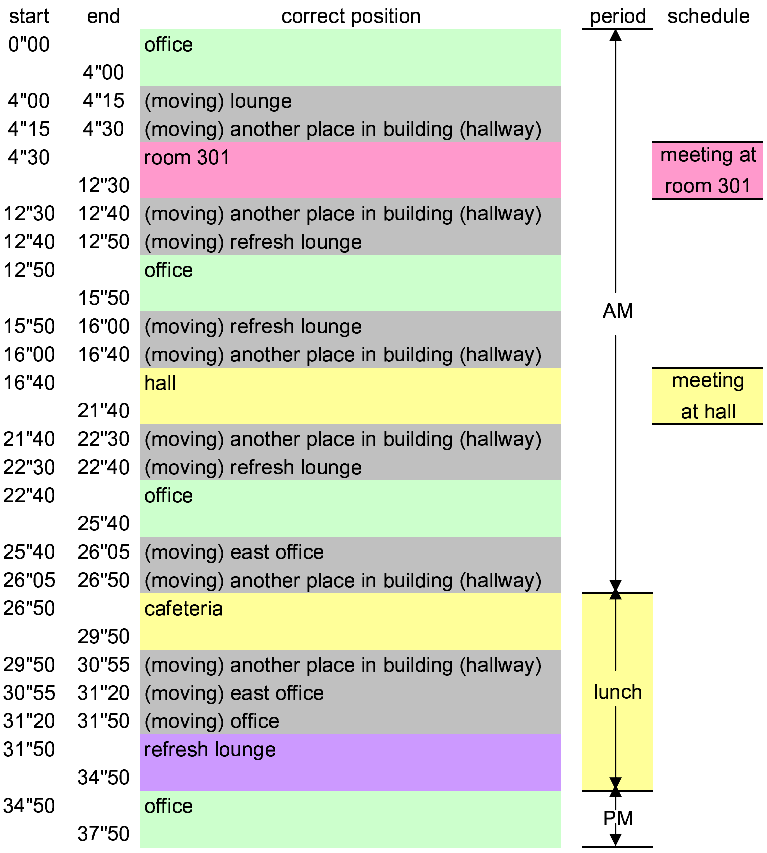

Figure 7: Scenario used in Experiment 1

Schedule We defined the following two different schedules and performed an analysis of precision for each schedule.

- *exact*: Each meeting is specified as taking as long as the correct one.
- *no_plan*: No plan is specified in this schedule.

6.4.2 Result

For each above-mentioned schedule, we compared the precision of the following four inference methods:

- *proposed*: the output of the proposed HMM.
- *no microstate*: the output of the proposed HMM without micro states, i.e., we assigned a single state to each position and the self loop probability was determined so that the duration distribution followed the geometric distribution with the given mean.
- *sensor+schedule*: the output of the sensor system replaced the output symbol ‘no_signal’ according to the following rules.

Table 1: Duration distribution for Experiment 1

	μ (min.)	σ (min.)
room_301	6	2
room_302	6	2
refresh_corner	3	0.1
lounge	0.25	0.1
office	3.5	0.5
east_office	0.4	0.1
laboratory	3	0.1
hall	3	2
cafeteria	3	0.1
another_place_in_building	1	0.1

Table 2: Result of Experiment 1

		subject						
schedule	method	a	b	c	d	e	f	average
exact	proposed	0.774	0.779	0.828	0.763	0.812	0.761	0.786
	no microstate	0.781	0.781	0.826	0.766	0.811	0.732	0.783
	sensor+schedule	0.770	0.775	0.804	0.759	0.796	0.730	0.772
	sensor	0.563	0.556	0.690	0.646	0.683	0.595	0.622
no plan	proposed	0.657	0.652	0.784	0.712	0.770	0.705	0.713
	no microstate	0.637	0.623	0.755	0.697	0.741	0.656	0.685
	sensor+schedule	0.563	0.556	0.690	0.646	0.683	0.595	0.622
	sensor	0.563	0.556	0.690	0.646	0.683	0.595	0.622

  - If the current position l is specified in the schedule, then replace 'no_signal' with l.
  - Otherwise, replace 'no_signal' with 'another_place_in_building.'

  We assumed that the posterior probability was 1 for the output position and 0 for the other positions.

- *sensor*: the same as *sensor+schedule* except that 'no_signal' was simply replaced with 'another_place_in_building.'

The results are summarized in Table 2. In each schedule, the proposed method resulted in better precision than the other methods used. In particular, when no plans were specified in the schedule, the proposed method gave much better precision than both the *sensor* and *sensor+schedule* methods. This suggests that our method is robust against incorrect schedules, which frequently appear in practice.

Table 3: Result of Experiment 2

	subject	
	c	d
proposed	0.788	0.842
sensor+schedule	0.723	0.796
sensor	0.723	0.796

6.5 Experiment 2

6.5.1 Settings

On a single day, two subjects (the same as the subjects c and d in Experiment 1) performed their ordinary work while the output of the sensor system was recorded. Moreover, the correct position at each time was recorded by each subject in a notebook. The output of the sensor system was recorded every minute, from morning till the subjects left the building (approximately eleven hours). The length N of the input sequence was 700 and 642 for subjects c and d, respectively.

6.5.2 Result

The results are summarized in Table 3. The proposed method resulted in better precision than *sensor* and *sensor+schedule*. In this experiment, the precisions of *sensor* and *sensor+schedule* were the same, because there were no time instances during the experiment in which a plan was specified and 'no_signal' was observed.

7 Conclusion

We proposed a presence-aware trust management architecture for a ubiquitous environment and implemented a prototype system with a presence-aware access control policy. We also proposed an HMM for inferring a user's presence and implemented it in a prototype system.

In the implementation, we assumed a connection service for an office computer network and used an RFID tag system as a sensor subsystem. Policies for trust management and access control are completely separated from the policy engine. Similarly, the behavior of HMM is completely determined by its transition/output probability tables. Hence, the proposed model can be adapted for other applications by changing the security policies and HMM parameters. This implies that the proposed model has a variety of potential applications within ubiquitous environments.

References

[1] J. Al-Muhtadi, A. Ranganathan, R. Campbell and M. D. Mickunas, Cerberus: a context-aware security scheme for smart spaces, IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom), 489–496, 2003.

[2] E. Bertino, B. Catania, M. L. Damiani and P. Perlasca, GEO-RBAC: A spatially aware RBAC, 10th ACM Symp. on Access Control Models and Technologies (SACMAT), 29–37, June 2005.

[3] M. Blaze, J. Feigenbaum and J. Lacy, Decentralized trust management, IEEE Security and Privacy, 164–173, 1996.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis and A. Keromytis, The KeyNote trust-management system, RFC 2704, 1999.

[5] M. J. Covington, W. Long, S. Srinivasan, A. K. Dey, M. Ahamad and G. D. Abowd, Securing context-aware applications using environment roles, 6th ACM Symp. on Access Control Models and Technologies (SACMAT), 10–20, May 2001.

[6] C. M. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. M. Thomas and T. Ylonen, SPKI certificate theory, RFC 2693, 1999.

[7] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, Bayesian filtering for location estimation, IEEE Pervasive Computing, 2(3), 24–33, 2003.

[8] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor and Y. Ravid, Access control meets public key infrastructure, or, assigning roles to strangers, IEEE Security and Privacy, 2–14, 2000.

[9] R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben and J. Reitsma, Context sensitive access control, 10th ACM Symp. on Access Control Models and Technologies (SACMAT), 111–119, June 2005.

[10] J. Krumm, G. Cermak, and E. Horvitz, RightSPOT: a novel sense of location for a smart personal object, UbiComp 2003, LNCS 2864, 36–43, 2003.

[11] J. Krumm, L. Williams, and G. Smith, SmartMoveX on a graph — an inexpensive active badge tracker, UbiComp 2002, LNCS 2498, 299–307, 2002.

[12] D. J. Patterson, L. Liao, D. Fox, and H. Kautz, Inferring high-level behavior from low-level sensors, UbiComp 2003, LNCS 2864, 73–89, 2003.

[13] L. R. Rabiner and B. H. Juang, An introduction to hidden Markov models, IEEE ASSP Magazine, 3(1), 4–16, 1986.

[14] J. Rosenberg, A presence event package for the Session Initiation Protocol (SIP), RFC 3856, 2004.

[15] K. E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills and L. Yu, Requirements for policy languages for trust negotiation, IEEE Int'l Workshop on Policies for Distributed Systems and Networks (POLICY), 68–79, 2002.

[16] Y. Yu, M. Winslett and K. E. Seamons, Interoperable strategies in automated trust negotiation, 8th ACM Conf. on Computer and Communications Security (CCS), 146–155, 2001.

A Sample trust policy

```
permit(N,U,high-wlan,normal) :-
   cert(N,U,full-time),
   position(U,L,P), room_for_ft(L),
   90 < P, schedule(U,L).

permit(N,U,high-wlan,passwd) :-
   cert(N,U,full-time),
   out_of_company(U),
   parameter(U,connection,vpn).

permit(N,U,high-wlan,normal) :-
   cert(N,U,R), part_or_trn(R),
   position(U,L,P), room_for_pot(L), 90 < P,
   position(U1,L,P1), room_for_ft(L),90 < P1,
   cert(N,U1,full-time).

out_of_company(U) :- not(inside_of_company(U)).
inside_of_company(U) :-
   position(U,L,P), room_for_ft(L), 10 =< P.

part_or_trn(part-time).
part_or_trn(trainee).

room_for_ft(office).
room_for_ft(east_office).
room_for_ft(meeting_room).
room_for_ft(laboratory).

room_for_pot(office).
room_for_pot(laboratory).
```