# Cerberus: A Context-Aware Security Scheme for Smart Spaces*

Jalal Al-Muhtadi Anand Ranganathan Roy Campbell M. Dennis Mickunas
*Department of Computer Science,*
*University of Illinois at Urbana-Champaign,*
{almuhtad, ranganat, rhc, mickunas}@uiuc.edu

## Abstract

Ubiquitous computing has fueled the idea of constructing sentient, information-rich "smart spaces" that extend the boundaries of traditional computing to encompass physical spaces, embedded devices, sensors, and other machinery. To achieve this, smart spaces need to capture situational information so that they can detect changes in context and adapt themselves accordingly. However, without considering basic security issues ubiquitous computing environments could be rife with vulnerabilities. Ubiquitous computing environments impose new requirements on security. Security services, like authentication and access control, have to be non-intrusive, intelligent, and able to adapt to the rapidly changing contexts of the spaces. We present a ubiquitous security mechanism that integrates context-awareness with automated reasoning to perform authentication and access control in ubiquitous computing environments.

### Keywords

Ubiquitous computing, security, smart spaces, Gaia, authentication, access control, context-awareness.

## 1. Introduction

Ubiquitous computing advocates the construction of massively distributed computing environments that feature thousands of transparent devices and sensors. These gadgets enable the seamless integration of computing resources and physical spaces, and surround users with a convenient, information-rich atmosphere that we refer to as *a smart space*. Smart spaces should sense and react to situational information. They should tailor themselves to meet users' expectations and preferences, as long as the system's security policies are not violated. Context awareness is an important mechanism to achieve the "disappearing computer" vision [1, 2].

However, ubiquitous computing raises security and privacy issues. Smart spaces extend computing to physical spaces, thus, information and physical security become interdependent. Furthermore, the dynamism and mobility that smart spaces advocate can give additional leverage for cyber-criminals, techno villains, and hackers by increasing opportunities to exploit vulnerabilities in the system without being observed. Home and workplace smart spaces require proper and adequate security measures to be laid out to prevent unauthorized access and enforce security policies.

Traditional authentication and access control methods require much user interaction in the form of manual logins, logouts, and file permissions. These manual interactions violate the vision of non-intrusive ubiquitous computing. In addition, we believe that the security requirements of a smart space may vary according to the context of the space. Some situations (like during a confidential meeting or homeland security alerts) require greater security to be in place; while other situations may not require a very high level of security. Traditional security mechanisms are context-insensitive, i.e. they do not adapt their security policies to a changing context. In this paper, we try to address some of the security concerns in smart spaces by blending the security service into the background and removing user distractions. We apply context awareness and automated reasoning to the identification and authentication of users and access control to resources and services.

### 1.1 Security Requirements for Smart Spaces

Because ubiquitous computing revolutionizes human-machine and human-physical space interactions, it imposes additional requirements on security and privacy. Some of these new requirements include the following.

The security service itself has to be "ubiquitous," non-intrusive, and transparent.

The security has to be multilevel, i.e. able to provide different levels of security services depending on security policies, environmental situations and available resources.

The security system has to support a security policy language that is descriptive, well-defined, and flexible. The language should be able to incorporate rich context information as well as physical security awareness.

* This research is supported by a grant from the National Science Foundation, NSF CCR 0086094 ITR.

Finally, in an open, massively distributed, ubiquitous computing system, authentication should not be limited to authenticating human users, but rather it should be able to authenticate mobile devices that enter and leave the smart spaces, as well as applications and mobile code that can run within the smart spaces.

### 1.2 Gaia

In the *Gaia* project [3-5], we define a generic computational environment that integrates physical spaces and their ubiquitous computing devices into a programmable computing and communication system. Gaia provides the infrastructure for constructing smart spaces. This infrastructure consists of the core services that make up smart spaces. We believe that security and context awareness are two essential core services for any smart space. In this paper, we present *Cerberus,* a core service in Gaia that integrates identification, authentication, context awareness, and reasoning. Cerberus enhances the security of ubiquitous applications that are built using Gaia.

The remainder of this paper is divided as follows. Section 2 gives a brief overview of Cerberus. Section 3 talks about the security service of Cerberus. Section 4 discusses the context infrastructure of Cerberus. Section 5 discusses the knowledge base and security policies of Cerberus. Section 6 discusses the inference engine of Cerberus. Section 7 briefly illustrates a scenario and its implementation. Section 8 looks into some related work. Finally, Section 9 concludes.

## 2. Cerberus Overview

The Cerberus core service of Gaia aims to capture as much context information as possible by deploying different devices and sensors, identifying entities and reasoning automatically in order to provide an unobtrusive computer environment. Figure 1 shows the high-level overview of Cerberus. Cerberus consists of four major components: (1) the security service, (2) the context infrastructure, (3) a knowledge base that stores various security policies, and (4) an inference engine, which performs automated reasoning and enforces the security policies. In the following sections we talk about each of these components individually. Note that in Figure 1 we show the context infrastructure and the security service as black boxes, which will be expanded later on.

## 3. Gaia Security Service Component

First, we give some definitions of some security terms within the context of smart spaces. *Identification* links an *entity* with an identity. The entity can initiate identification (e.g. a user typing his user id) or the system can automate identification through sensors and detection. Entities are people, programs, devices, sensors, or even physical spaces. *Authentication* provides assurance for the claimed or detected identity of an entity in the system, i.e. it attempts to verify the identification of a particular entity. We use *principal* to refer to the entity that possesses the identity. Users, physical spaces, devices, applications, and mobile code snippets are all principals. *Security policies* guide the implementation of security in a system to match the requirements of the system. In a smart space setting, flexible security policies must incorporate dynamic and changing contexts.

Ubiquitous computing authentication mechanisms offer a balance between authentication strength and non-intrusiveness. A smart badge that transmits short range radio signals, for instance, is a good non-intrusive authentication mechanism; but provides a weak form of authentication. A challenge-response mechanism provides stronger authentication, but may require user interactions. Context may dictate *the strength* of authentication. The smart space should not intrusively dictate that users carry or wear specific devices. Instead, principals should authenticate themselves to the system using a variety of means depending on which approach least impacts the principals and provides enough assurance to the system. Authentication mechanisms include wearable devices, voice and face recognition, presenting a badge that contains identification information, fingerprint identification, retinal scans, etc. Different strengths of authentication are associated with *confidence values* that an entity has a given identity. This confidence value represents how “confident” the authentication system is about the identity of the principal. We represent this by a number in the range [0, 1]. This confidence value depends on the authentication devices and the authentication protocols used. Principals can employ multiple authentication methods in order to increase the confidence values associated with them. Access control decisions can now become more flexible by utilizing confidence information. Several reasoning techniques can be used to combine confidence values and calculate a net confidence value for a particular principal. The techniques we have considered so far include simple probabilities, Bayesian probability, and fuzzy logic [6]. In Section 5, we give more details on how we use confidence values in access control decisions.

Since identification and authentication can use a large number of diverse devices and as technology improves and new authentication devices become available, security systems need a dynamic method for adding new authentication devices and associating them with access control policies and protocols. Some methods of authentication are more convenient, reliable, or secure than others. For example, it is easy for smart badges to be misplaced or stolen. On the other hand, the use of biometrics, like an iris scan, for instance, is a more reliable means of authentication. Because of the various authentication methods and their different strengths, an adaptable security system should assign different levels of confidence to different authentication mechanisms and incorporate ad-

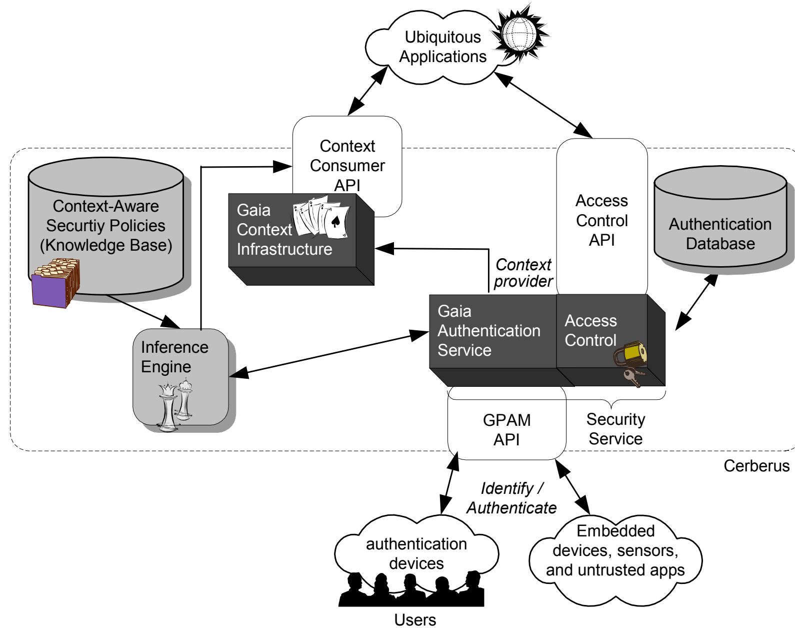


**Figure 1: Cerberus overview**

ditional authentication mechanisms, context, and sensor information to infer additional confidence to a principal's identity. The techniques can assist in detecting intruders, unauthorized accesses and assessing possible threat levels.

The various means of authenticating principals and the notion of different confidence levels associated with authenticated principals constitute additional information that can enrich the context awareness of smart spaces. In a later section, we illustrate how such information is inferred and exchanged with other Gaia core services.

To meet the stated requirements we propose a federated authentication service that uses distributed, pluggable authentication modules. Figure 2 provides a sketch of the authentication architecture that incorporates the objectives mentioned above. PAM (Pluggable Authentication Module) [7] provides an authentication method that allows the separation of applications from the actual authentication mechanisms and devices. Dynamically pluggable modules allow the authentication subsystem to incorporate additional authentication mechanisms on the fly as they become available. The Gaia PAM (GPAM) extends traditional PAM by providing support for federated, CORBA-based authentication modules. This GPAM is wrapped by an API that is made available for ubiquitous applications, services, and other Gaia components to request authentication of entities or inquire about authenticated principals. Since the authentication service may run anywhere in the space (possibly federated), we use CORBA facilities to allow the discovery and remote invocation of the authentication services that serve a particular smart space. The authentication modules themselves are divided into two types:

1. Gaia Authentication Mechanisms Modules (GAMM), which implement general authentication mechanisms or protocols that are independent of the actual device being used for authentication. These modules include a Kerberos authentication module, a SESAME [8] authentication module, the traditional username/password-module, and a challenge-response through a shared secret module.

2. The other type of modules is the Gaia Authentication Device Modules (GADM). These modules are independent of the actual authentication protocol; instead, they are dependent on the particular authentication device.

This decoupling enables greater flexibility. When a new authentication protocol is devised, a GAMM module can be written and plugged in to support that particular protocol. Devices that can capture the information required for completing the protocol can use the new authentication module with minimal changes to their device drivers. When a new authentication device is incorporated to the system, a new GADM module is implemented in order to incorporate the device into the smart space, however, the device can use existing security mechanisms by using CORBA facilities to discover and invoke authentication mechanisms that are compatible with its capabilities. In effect, this creates an architecture similar to PAM but federated through the use of CORBA. Many CORBA implementations are heavyweight and require significant resources. To overcome this hurdle, we used the Universally Interoperable Core (UIC), which provides a lightweight, high-performance implementation of basic CORBA services [9]. More implementation details about GPAM can be found in [10]. The access control part of the security ser-

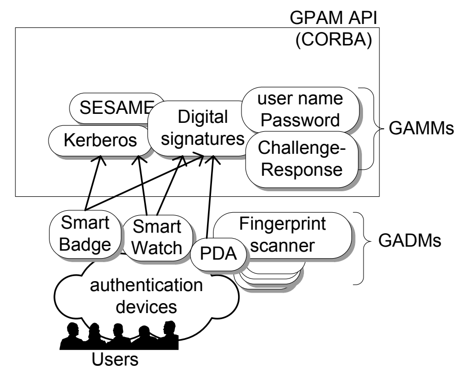


**Figure 2: Gaia Authentication Service**

vice provides an API, which ubiquitous applications and service providers can use to check whether principal *P* can perform a particular operation or not. The access control component forwards such inquiries to the inference engine. Depending on available context information and applicable security policies the inference engine replies with either ‘yes’ or ‘no.’ The access control component provides support for callbacks to the application, which can inform an application of possible context changes that may trigger a change in the access decision. We discuss the inference engine in Section 6.

# 4. Gaia Context Infrastructure

In this section, we describe our context infrastructure and a few of the key context operations. Our context infrastructure uses first-order predicate calculus and boolean algebra. This allows us to write various complex rules involving contexts easily and evaluate these rules in a manner similar to Prolog.

The security mechanisms (like authentication and authorization) in Cerberus make use of this predicate model of context. The development and use of a specific context model simplifies the development of these security mechanisms.

## 4.1 Basic Structure – the context predicate

We represent contexts as first-order predicates. The name of the predicate is the type of context that is being described (like location, temperature or time). It is also possible to have relational operators like “=” and “<” as arguments of a predicate.

Example contexts predicates are:

*Location (Bob, entering , room 2401) ;*
*Temperature ( room 3231 , “=” , 98 F);*
*Sister( venus , serena) ;*
*StockQuote( msft , “>” , $60);*
*PrinterStatus( srgalw1 printer queue , is , empty) ;*
*Time( New York , “<” , 12:00 01/01/01)*

The values that the arguments of a predicate can take are actually constrained by the predicate. For example, if the predicate is “location”, the first argument has to be a person or object, the second argument has to be a preposition or a verb like “entering,” “leaving,” or “in” and the third argument must be a location. We do perform simple type-checking of context predicates to make sure that the predicate does make sense.

This logical model for context is quite powerful. It is possible to express a rich variety of contexts using first order logic. This model of context allows us to describe the context of a system in a generic way, which is independent of programming language, operating system, or middleware.

## 4.2 Operations on Contexts

### 4.2.1 Boolean Operations on contexts

It is possible to construct more complex context expressions by performing boolean operations like conjunction, disjunction and negation over context predicates. For example:

- *Location( Alice , Entering , Room 3211)* $\wedge$ *Social Activity( Room 3211, Meeting)* refers to the context that Alice is entering Room 3211 and that there is a meeting going on in that room.
- *EnvironmentLighting( Room 2401 , Off )* $\vee$ *EnvironmentLighting( Room 2401, Dim )* refers to the context that the lighting in Room 2401 is either Off or Dim.
- NOT *Location( Alice , In , Room 3211)* refers to the context that Alice is not in Room 3211.

### 4.2.2 Quantification over Contexts

It is possible to have one or more arguments of the context predicate be variable and then quantify over this variable. This allows us to parameterize the context and represent a much richer set of contexts. The model allows both universal and existential quantification over variables.

The existential quantifier (i.e. “there exists”) indicates that the context which follows is true for at least one value of the variable within the indicated scope of the variable. Thus, $\exists_S\, x\, P(x)$ is true iff $P(x)$ is true for some value of $x$ belonging to the set S. For example, to express the condition that Chris is in some location, we can write:

$\exists_{Location}\, y$ *Location (Chris, In, y)*

The universal quantifier (i.e. “for all”) indicates that the context which follows is true for all values of the variable that lie in the scope of the variable. Thus, $\forall_S\, x\, P(x)$ is true iff $P(x)$ is true for all values of $x$ belonging to the set S. For example, to refer to all people in room 3231, we write an expression of the form:

$\forall_{People}\, x$ *Location( x, In, Room 3231)*

Existential and universal quantifiers allow specifying various complex contexts fairly easily. For example, a room controller application could associate the context $\exists_{Person}\, s$ *Location( s, Entering, Room 3234 )* with the action of playing a welcome message. This means that whenever any person enters Room 3234, the room controller application plays a welcome message. It is possible to construct more complex contexts by performing boolean operations on context predicates. Possible operations are disjunction (“or”) and conjunction (“and”).

The model uses the many-sorted logic model that quantifies over a specific domain of values. For example, we define various sets of values (like Person, Location, Stock Symbol, etc) where the Person set consists of the names of all people in our system, the Location set consists of all valid locations in our system (like room num-

bers and hallways), and the Stock Symbol set consists of all stock symbols that the system is interested in (e.g. IBM, MSFT, SUNW, etc.). Each of these sets is finite. Quantification of variables uses the values of one of these sets. Because it quantifies over finite sets, evaluations of expressions with quantifications will always terminate.

The Gaia Context Infrastructure, illustrated in Figure 3, allows applications to obtain a variety of contextual information. Various components, called Context Providers, obtain context from either sensors or other data sources. Context Providers allow applications to query them for context information. Some Context Providers also have an event channel where they keep sending context events. Thus, applications can either query a Provider or listen on the event channel to get context information. All Context Providers support a similar interface for getting contexts and listening to context events. So, applications do not have to worry about the actual type of Context Provider they are querying. They can query any Context Provider in the same way. This aids development of context-aware applications greatly.

Context Synthesizers are components that get sensed contexts from various Context Providers, derive higher-level or abstract contexts from these simple sensed contexts and provide these inferred contexts to applications. For example, we have a Context Synthesizer which infers the activity that occurs within a room based on number of people in the room and the applications that are running. Context Synthesizers use a variety of reasoning and learning mechanisms to infer new high-level contexts from the low-level sensed contexts.

The Context Provider Lookup Service allows searches for different context providers. Providers advertise the set of contexts they provide with the Context Provider Lookup Service. This advertisement is in the form of a first order expression that describes the context provided by the Provider. Applications can query the Lookup Service for a context provider that provides contextual information it needs. The Lookup Service checks if any of the context providers can provide what the application needs and returns the results to the application.

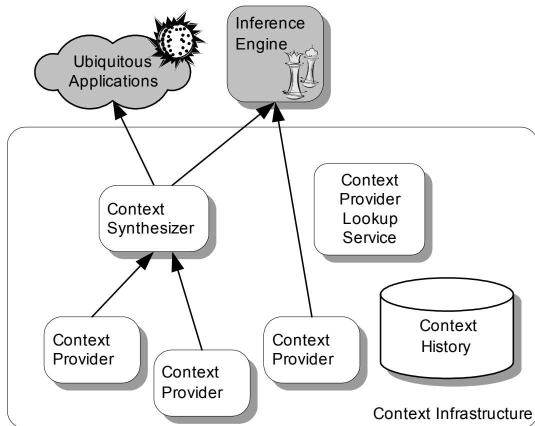


**Figure 3: The Gaia Context Infrastructure**

Context History is maintained in a database, where all past contexts are stored. The Context History makes use of one of the features of our event service, which allows event channels to be "persistent", i.e. all events sent on these channels are stored in a database along with a timestamp indicating when the event was sent.

The various entities in our Context Infrastructure are implemented as components on top of Gaia. They use CORBA to communicate with each other and can also be discovered using standard CORBA mechanisms like the CORBA Naming and Trading Services.

## 5. Security Policies

Security policies in Cerberus are written as rules in first order logic. There are two kinds of policies used in Cerberus. One set of policies is used by the authentication server at the time of logon or authentication. These policies determine the confidence level of authentication. The other set contains access control policies, which determine whether a principal is allowed access to a particular resource.

To illustrate, we present a simplified example of such policies. The various authentication devices are assigned confidence values, using the following rules:

*ConfidenceLevel (smart_watch, 70%)*

*ConfidenceLevel (smart_badge, 10%)*

*ConfidenceLevel (fingerprint_scan, 90%) …*

These values are set by the system administrator based on the strength of the authentication device used and the protocol employed. If a principal *P* has been positively authenticated using its smart watch, say, then the authentication service inserts a new fact into the knowledge base:

*Authenticated(P, smart_watch)*

Similarly if the principal is authenticated using different forms:

*Authenticated(P, password)*

*Authenticated(P, fingerprint)*

We can define the confidence value (*V*) associated with an authenticated principal *P* as:

$ConfidenceValue\ (P, V)$ :- $\exists_{device}\ X\ (\ Authenticated(P,X) \wedge ConfidenceLevel\ (X, V)\ )$

Now, access control decisions can take the confidence information into account by defining rules like the following:

$CanAccess\ (P,\ ColorPrinter\ )$ :- $\exists_{number}\ V\ (ConfidenceValue(P, V) \wedge V>60\%)$

Here, *P* can only access the color printer if the authentication system has identified *P* with a confidence value of more than 60% (i.e. the principal has authenti-

cated itself using at least one device whose confidence level is more that 60%). Note that in the example above, we do not calculate a net confidence value, but instead we grant access only if a user performed an authentication that grants her a confidence value of more than 60%. A more flexible way of doing this would permit us to combine multiple confidence levels and produce a net confidence value, i.e.:

*CanAccess (P, ColorPrinter ) :-* $\exists_{number}$ *V (NetConfidenceValue (P, V)* $\wedge$ *V>60%)*

Representing system policies in first order predicate logic provides greater flexibility and dynamism while allowing rules to be evaluated efficiently.

## 6. Inference Engine

The Inference Engine performs two kinds of tasks:

1. It gives a level of confidence when a person authenticates himself. It makes use of the authentication policies as well as contextual information to assign the confidence level.
2. It evaluates queries from applications about whether a certain entity is allowed to access a certain resource. It makes use of application-specific access control policies, the credential of the entity, and contextual information to decide whether an entity has access to a resource.

The Inference Engine has access to all the authentication policies of the smart space and the access control policies of all the components in the smart space. It can also get context information from different context providers. It can either query various context providers or it can listen for events from context providers. It makes use of the Context Provider Lookup Service to look up various context providers. It can also get authentication information of various people in the space from the authentication service.

The authentication and access control policies are represented as first order expressions. The contextual information that the Inference Engine gets from context providers is also in the form of first order expressions. The Inference Engine evaluates queries in a way similar to how Prolog handles queries. It tries to resolve any query using the information it has about the policies and the context. Our current implementation has a very simple evaluation engine. It evaluates the query using the standard techniques of resolution and unification. If a unification that leads to all variables in the query being bound is obtained, then it returns the result to the application, else it returns nothing.

For example, a component that controls a wall display in a particular room has an access control policy that says that if there is a UbiComp Seminar going on in the room, then the presenter has access to the display. The policy may look like

$\forall_{People}$ *X Access(X, Display) :- SocialActivity(Room 2401, UbiComp Seminar)* $\wedge$ *IsPresenter(Ubicomp Seminar, X)*

So, when somebody (say "Bob") tries to access the display, the display component gets the credential of the person to see who it is. It then queries the inference engine to see if the person is allowed to use the display. This query would look like

*?Access(Bob, Display)*

To answer this query, the Inference Engine needs to know what the social activity in the room is. If it does not already know this information, it queries a context provider which knows about the social activity in the room. So, it sends a query to this context provider that looks like

*?SocialActivity(Room 2401, UbiComp Seminar)*

It gets back a reply of either "True" or "False."

If it gets a "True" reply, it asks about the presenter from a context provider that knows such information about the seminar. It then evaluates the rule (and any other access rules) to determine if Bob is to be given access to the display and sends this decision back to the display component.

Applications maintain the concept of sessions with principals. The first time a principal tries to use an application, it checks with the security service to see if the principal is allowed access. Subsequent accesses to the same application are not checked with the Security Service. Thus, the principal is allowed access to the application until the application is notified by the Security Service to act otherwise.

Since a ubiquitous computing environment is very dynamic, the context of the environment changes very frequently. This affects any access control decisions that may have been made. For example, a person may have access to a certain device when there is a meeting going on in the room and he is the presenter, but not otherwise. So, if he is initially granted access to the device and later on, the activity in the space changes from "meeting" to "demo", then he should no longer have access to the device. Applications can ask to be notified when changes in context of the space require changes in access control decisions.

In the example, described above, the display component would ask the Inference Engine to notify it whenever the following expression becomes true:

*NOT Access(Bob, Display)*

The Inference Engine in turn asks the social activity context provider to provide a notification when the condition NOT *SocialActivity(Room 2401, UbiComp Seminar)* becomes true. It also asks the PresentationManager Context Provider to provide a notification when the condition *NOT IsPresenter(Ubicomp Seminar, X)* becomes true. When the Inference Engine gets any such notification, it re-evaluates the rules; and if the expression *Access(Bob,*

*Display)* no longer evaluates to true, it sends a notification to the display component.

For evaluating rules with quantification, the Inference Engine has access to the set of values that the quantified variable can take. In our model, quantification is done over finite sets of values. The Inference Engine just tries each of the values and evaluates the rules using these values. Our Inference Engine supports dynamic assertion of facts, and dynamic retracting of these facts.

An issue in logic programming is ensuring that the evaluation of queries can be terminated and is, hence, safe. In our system, the Inference Engine maintains only a finite set of sentences. Also quantification is done over finite sets. Thus, query evaluations will always terminate. More detailed analyses of these issues can be found in [11-13].

# 7. Implementation

In this section we discuss our implementation, where we use Cerberus facilities to authenticate users, capture context information, and make access decisions for one of the Gaia applications: the "Powerpoint Viewer." The Powerpoint Viewer application is a wrapper for Microsoft™ Powerpoint that uses Gaia facilities programmatically to control which displays to use for the presentation, as well as the ability to synchronize between different displays and move slides from one display to another. The Powerpoint "Input Sensor" is a special component of this application, which allows a person to control the presentation (e.g. moving to next or previous slide).

The Gaia testbed is a prototype room containing state-of-the-art equipment, including 5.1 programmable surround audio system, four touch plasma panels with HDTV support, HDTV video wall, X10 devices, electronic white boards, IR beacons, Wi-Fi access points, and flat panel desktop displays. Authentication devices supported include smart watches, USB key chains, fingerprint scanners, Java iButtons®, and the Space Selector (an application that runs on laptops and some PDA devices). Currently, this smart space is used for group meetings, seminars, presentations, demos, and for entertainment (listening to music and watching HDTV). These different uses translate into different contexts. The smart space has a number of immobile devices and displays that are secured in the room and are assumed to be trusted. This includes the plasma panels, and the PCs that run Gaia kernel, services, and some applications in the room, including the Powerpoint Viewer application.

We have considered seminars that occur in this room and use the Powerpoint Viewer. Our implementation works as follows.

1. One or more principals log into the Cerberus system using a subset of the devices or gadgets they have in their possession (or through their biometric features). A credential is created for each principal, which holds its confidence level. Figure 4 contains a snapshot of the authentication policy that deals with authentication and the calculation of a net confidence value. The policy is written in Prolog. The policy shown in the figure uses probability theory to calculate a net confidence. I.e., if a principal receive confidence values of $V_1, V_2, .. V_n$ from different authentication methods, then the net confidence value $V_{net}$ is calculated as: $V_{net} = 1 - (1-V_1)(1-V_2)\ldots(1-V_n)$
2. For different spaces, applications, and resources, access policies are defined. When a user tries to use some application or resource, the inference engine evaluates the policies to see if the user has permissions to use the application or resource. These policies are based on the current context, the confidence level of authentication, the role of the user, etc.
3. If a particular policy makes use of some context information, the inference engine contacts the context infrastructure as illustrated in Figure 3 and mentioned in Section 4. Applications also submit callback information – so that when context changes and a certain access is no longer valid for the user, then the application is notified to stop providing the service to the user.

Using this framework, we are able to write policies to designate presenters based on dates and times, e.g. Bob is the presenter on Monday 9/23 from 1 – 2 PM, and assign different permissions for different principals or roles. For instance, our regular default setting grants the presenter the ability to run the Powerpoint Viewer, control the presentation, and choose any displays for showing the slides. Authorized attendants are not allowed to control the slides or to move them from or to public displays, however, they are granted permission to copy slides or duplicate the slideshow to their personal devices. Principals designated as "guests" are not granted any control over the presentations and are not allowed to move the slides into their personal devices. We plan to have more details about the implementation and performance of our system in the camera-ready version of this paper.

# 8. Related Work

Covington et al. [14, 15] tackled the problem of securing a smart home environment. They refer to this environment as the "Aware Home." In their work, Covington et al. extend the RBAC access control model to develop a non-intrusive access control system that can make use of environmental and contextual information. The system is meant to be usable and easy to manage for homeowners and to act as a safeguard against remote attacks or break-ins. In their model they capture context information in the form of environmental roles. Environmental conditions, which activate environmental roles, are defined. Their access control mechanism is integrated with a toolkit for gathering context information from sensors. While their proposed language is based on logic it appears to be too

```
%policies related to the authentication service
clauses
  % confidence associated with
  %authentication devices:
  %must be set by the admin based on the policy.
  confidenceLevel("SmartWatch-ChalResp",70).
  confidenceLevel("SmartWatch-Passwd",70).
  confidenceLevel("SmartBadge",10).
  confidenceLevel("SmartBadgev2",50).
  confidenceLevel("Terminal-Passwd",60).

confidenceLevel("FIU510_FingerprintScanner",90).
  confidenceLevel("USB_keychain",60).
  confidenceLevel("SpaceSelector-Passwd", 75).
  ...
  %facts dynamically asserted by
  %the authentication service:
  authenticated("Bob", "SmartBadge").
  authenticated("Alice","SmartWatch-ChalResp").
  authenticated("Alice", "SmartBadge").
  identified("Charlie").
  ...
predicates
  % Each principal P has a list of
  % confidence values (CV),
  % one per authentication device he/she used
  confidence_value_list(P,CV) :-
    build_confidence_table(P, [], L).
  ...

  % building list of confidence values.
  build_confidence_table(P, L, L) :-
    not(authenticated(P,_)).
  build_confidence_table(P, L1, [E | L2]) :-
    authenticated(P, D), confidenceLevel(D, E),
    retract( authenticated(P,D) ),
    build_confidence_table(P, L1, L2).
  ...

  % calculate the net confidence per principal
  % using probability.
  netConfidenceValue (P, CV_NET) :-
    confidence_value_list(P, CV_LIST),
    calc_net_conf_prob (CV_LIST, TEMP),
    CV_NET = (1-TEMP)*100, !.
  calc_net_conf_prob ([], 1).
  calc_net_conf_prob ([CV_H | CV_REST], VALUE):-
    calc_net_conf_prob( CV_REST, TEMP),
    VALUE = (1-CV_H/100)*TEMP.
```

**Figure 4: Portions of the security policy used in the Gaia testbed, written in Prolog syntax. This portion shows how confidence values are maintained and how the net confidence for a particular principal is combined. Note that some facts are asserted dynamically based on context and authentication information**

simplistic. In Cerberus we present a more expressive rule language that support binary operators, quantification, and complex inferring. Stajano [16] gives an overview of the security problems and vulnerabilities that ubiquitous computing brings along. Our solution addresses some of these issues. In a previous work [17], we examined some issues of authentication and privacy in ubiquitous computing environments and laid out a preliminary design for a solution.

# 9. Conclusion

Security for smart spaces is an interesting and challenging research endeavor. The dynamism, ubiquity, and non-intrusiveness of the ubiquitous computing paradigm present more challenges and raise new issues. We have tackled some of these problems by introducing Cerberus, a federated, context-aware, security scheme. Our system supports multilevel authentication, where principals are associated with confidence values. Our context infrastructure captures rapidly changing context information and incorporates it into our knowledge base. Context-aware security policies are described in an expressive language and can be evaluated efficiently using an inference engine. We present a simple and efficient method for revoking access if context related information changes.

# 10. References


[1] M. Weiser, "Hot Topics: Ubiquitous Computing," *IEEE Computer*, 1993.
[2] M. Weiser, "The Computer for the Twenty-First Century," in *Scientific American*, vol. 265, 1991, pp. 94-104.
[3] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces," *IEEE Pervasive Computing (accepted)*, 2002.
[4] M. Roman and R. Campbell, "GAIA: Enabling Active Spaces," presented at 9th ACM SIGOPS European Workshop,, Kolding, Denmark, 2000.
[5] M. Roman, C. Hess, A. Ranganathan, P. Madhavarapu, B. Borthakur, P. Viswanathan, R. Cerqueira, R. Campbell, and M. D. Mickunas, "GaiaOS: An Infrastructure for Active Spaces," University of Illinois at Urbana-Champaign Technical Report UIUCDCS-R-2001-2224 UILU-ENG-2001-1731, 2001.
[6] L. Zadeh, "Fuzzy sets as basis for a theory of possibility," *Fuzzy Sets and Systems*, vol. 1, pp. 3-28, 1978.
[7] V. Samar and R. Schemers, "Unified Login with Pluggable Authentication Modules (PAM)," RFC 86.0, 1995.
[8] P. Kaijser, T. Parker, and D. Pinkas, "SESAME: The Solution to Security for Open Distributed Systems," *Computer Communications*, vol. 17, pp. 501-518, 1994.
[9] M. Roman, F. Kon, and R. H. Campbell, "Reflective Middleware: From Your Desk to Your Hand," *IEEE Distributed Systems Online Journal, Special Issue on Reflective Middleware*, 2001.
[10] J. Al-Muhtadi, D. Mickunas, and R. Campbell, "The Gaia Authentication Architecture," UIUC Technical Report (number pending) 2003.
[11] A. K. Chandra and e. al., "Horn Clauses Queries and Generalization," *J Logic Programming*, 1985.
[12] O. Shmueli, "Decidability and expressiveness aspects of logic queries," presented at sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of database systems, San Diego, CA USA, 1987.
[13] M. Jarke and e. al, "An Optimizing PROLOG Front-End to a Relational Query System," presented at ACM SIGMOD '84 Conference, Boston, MA, 1984.
[14] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd, "Securing context-aware applications using environment roles," presented at Proceedings of the Sixth ACM Symposium on Access control models and technologies, Chantilly, Virginia, United States, 2001.
[15] M. J. Covington, M. J. Moyer, and M. Ahamad, "Generalized Role-Based Access Control for Securing Future Applications," presented at 23rd National Information Systems Security Conference, 2000.
[16] F. Stajano, *Security for Ubiquitous Computing*: Halsted Press, 2002.
[17] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M. D. Mickunas, "A Flexible, Privacy-Preserving Authentication Framework for Ubiquitous Computing Environments," presented at International Workshop on Smart Appliances and Wearable Computing (Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops 2002), Vienna, Austria, 2002.