

# Privacy Engine for Context-Aware Enterprise Application Services

Marion Blount<sup>1</sup>, John Davis<sup>2\*</sup>, Maria Ebling<sup>1</sup>, William Jerome<sup>1</sup>,  
Barry Leiba<sup>1</sup>, Xuan Liu<sup>1</sup>, Archan Misra<sup>1</sup>

<sup>1</sup>IBM T J Watson Research Center, <sup>2</sup>Ling Fling, Inc.

{mlblount, ebling, wfj, xuanliu, archan}@us.ibm.com; john@lingfling.com; leiba@watson.ibm.com

## Abstract

*Satisfying the varied privacy preferences of individuals, while exposing context data to authorized applications and individuals, remains a major challenge for context-aware computing. This paper describes our experiences in building a middleware component, the Context Privacy Engine (CPE), that enforces a role-based, context-dependent privacy model for enterprise domains. While fundamentally an ACL-based access control scheme, CPE extends the traditional ACL mechanism with usage control and context constraints. This paper focuses on discussing issues related to managing and evaluating context-dependent privacy policies. Extensive experimental studies with a production-grade implementation and real-life context sources demonstrate that the CPE can support a large number of concurrent requests. The experiments also show valuable insight on how context-retrieval can affect the privacy evaluation process.*

## 1. Introduction

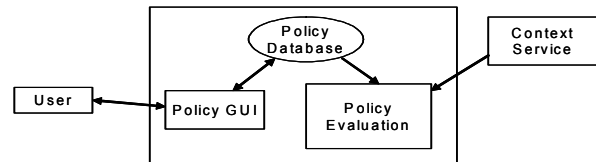
Context-aware computing applications, such as real-time asset tracking, location and calendar-aware conferencing, and opportunistic collaboration, have long held the promise of improving the productivity of people and organizations. To offer the most benefit, however, these applications require detailed knowledge of the dynamic state of an individual, device or business process. Privacy concerns thus remain a serious roadblock (e.g., [1]) to the practical realization of such pervasive computing scenarios.

In this paper, we present the design and evaluation of the *Context Privacy Engine (CPE)*, a privacy engine for regulating access to an individual's "context" data, such as location, movement patterns, and desktop activities. CPE does not directly enforce privacy, but is used by a context service to assist in the controlled release of sensitive information. An important aspect of CPE's design was the need to accommodate context-dependent policies.

At a high level, CPE is an extended ACL-based mechanism for context-dependent access control. The CPE privacy model specifies not only who can access what data, but also for what purpose and under what context. A variety of prior work (e.g., [3][4]) has explored the use of ACL-based privacy systems. However, CPE addresses unique challenges specific to a large enterprise domain where large number of concurrent access is a requirement. As we shall see, providing low-latency authorization for a large number of concurrent requests can incur a significant performance issue, especially when we confront the realities of accessing the dynamically varying context data from various sources.

The main contributions of this paper are as follows:

- We propose a context-dependent policy model that extends traditional ACL mechanisms with contextual information and usage control
- We identify the problem of possible unintended privacy leakage with context constraint specification, and discuss how to address this while supporting a flexible policy management and low-latency policy evaluation.
- We address the practical reality that some context information for evaluating a privacy policy may be unavailable.
- Finally, we show, through our performance studies, that the retrieval of context information can be expensive (in terms of latency), and suggest a note of caution against expecting all enterprise applications to become "context-aware" overnight.



**Figure 1: Component Architecture of CPE**

Figure 1 illustrates the overall CPE system architecture. It consists of three logically distinct components:

1. The *Policy UI* component for easy policy management
2. The policy database for storing privacy policies

\* This work was completed while John was working at IBM

3. The *Policy Evaluation* component for evaluating access requests based on the policies and makes “Grant/Deny” decisions

Our focus in this paper is to discuss the CPE context-dependent privacy policies and the Policy Evaluation component. The discussion of the GUI component for policy management is relegated to a separate paper. As CPE focuses purely on a single enterprise domain, issues related to privacy in multi-domain or federated environments are beyond the scope of our current discussion.

The rest of the paper is organized as follows. Section 2 reviews the prior work. Section 3 presents the CPE context-dependent privacy policy with the focus on addressing the context-dependent issues. Section 4 presents performance results. Finally, Section 5 concludes the paper.

## 2. Related Work

The Aware Home initiative [3] may be the closest in spirit to our work on CPE. The Aware Home access control mechanism uses the Context Toolkit [7] to extend role-based access control with “environment roles”. Given its focus on residential homes, the Aware Home framework does not concern itself with practical features that are intrinsic to an enterprise environment, such as hierarchical policies and overrides, and low evaluation latency. Moreover, the Aware Home approach does not investigate problems encountered by the unavailability of context data.

The Houdini framework [4] focuses on the expression and enforcement of individual privacy preferences in a cellular network. One of Houdini’s innovations is the use of Web-based forms for defining various generic activity-oriented context attributes, based on a combination of raw location and time data. Whereas the framework focuses on supporting rapid evaluation for a single user, we aim to support  $O(100)$  concurrent policy evaluations issued by different users in an enterprise setting where the needs of individuals and organizations must be balanced and where context data may come from a variety of different (and possibly unavailable) sources.

Attribute-based Access Control (ABAC) extends the RBAC paradigm. Policies are specified in terms of predicates over a set of attributes, and an individual’s access attempt is evaluated on the set of attributes that she possesses. For example, the RT framework [8] defines semantics for localized control over roles and attributes, and techniques for delegating such control, while trust-based privacy preferences (e.g., [9]) allow the policy owner to specify the minimum trustworthiness of a requester to access the specified context data. Such a framework is used by approaches

such as pawS [10], which mediate user/device interaction with those devices, or ContextFabric [11], which supports data sharing between devices and services in an untrusted domain. Although our focus is currently on CPE policies and the evaluation algorithm, the Context field in a CPE policy can easily be extended to adopt the ABAC framework.

There are a variety of other approaches to privacy issues in mobile computing environment. A. Kapadia et al.[21] proposed a privacy language based on the metaphor of physical walls, and assumes users understand and accept the privacy implication of a physical wall. The approaches of collaborative filtering [12] and K-anonymity [13, 20, 22, 23] assume some semantics (e.g., spatial or temporal) on the underlying data to implicitly control the granularity of data exposure. Similarly, approaches such as “faces” [14] or pseudonym [15] focus on implicit specification of user privacy preferences. In contrast, CPE employs an explicit privacy model, where the evaluation algorithm does not understand the semantics or the value of the data.

Other work in supporting explicit fine-grained privacy policies, such as CoPS [19] and [24] are focused on extending the RBAC mechanism for achieving expressiveness. However, these projects did not discuss the issues related to context-dependent policies. CPE on the other hand focuses on context-dependent policy mechanism and proposes solutions to address corresponding issues.

## 3. Context-dependent Privacy Policy

CPE is designed as a policy evaluation engine, where a client (e.g., a context service) issues a request (to CPE) for permission to release context information pertaining to one or more subjects to a specified requester. In this section, we describe the CPE privacy policy mechanism. We specifically focus on the context-dependent privacy policies.

### 3.1 Privacy policy model

To support the twin needs of flexibility and scalability for policy specification and management, a CPE privacy policy mainly contains the following fields:

**Subject:** a user or a group of users whose information is protected by this policy

**Information:** the subject’s information that this policy protects

**Requester:** an individual user or a group to whom this policy applies when requesting for information

**Application:** the applications to which the information may be released

**Context:** a set of context constraints that must be satisfied for the policy to be in effect

**Policy level:** the hierarchy level of the policy to support the need of policy overriding

**Release:** “Grant” or “Deny”, a decision whether to release the information or not

By separating "application" and "requester", we support the flexibility for information to be used to perform a service without necessarily releasing the information to the requester. For instance, a policy that allows Joe to use the "IntelligentDialer" application to telephone Jane without actually giving Joe Jane's telephone number.

The Subject and Requester fields can be either individual users or groups. Groups can be defined hierarchically and usually mirror organizational hierarchies. The advantage of this is to improve the scalability of policy management and to separate the logical privacy policy from concrete deployment.

An example privacy policy is: *President (subject) allows (Release "Grant") white house staff (requester) to know his location (information) when both the president and the requester are in the white house (context).*

We support the ability to override policies by using the policy level field. A policy at a higher level is able to override all policies at a lower level. This can satisfy the enterprise requirement for enforcing a regulation/corporate rule while enabling individualized user policy at the same time.

For a given request, there can be multiple policies that are in effect, and the release decision from those policies could be different. To resolve the conflict, we can first use the policy level to decide which policies are higher in the hierarchy. For policies at the same level, we introduce specificity checking for resolving the conflict. Policies that are defined more specifically are considered more specific. For example, in the information field, specificity follows a hierarchy (e.g., location.address.city is more specific than location.-address). However, there are situations where the specificity is unclear, for example, the specificity for two user-defined groups. In these cases, the "Deny when in doubt" principle is applied and the request is refused if any policy denies the access.

### 3.2 Context-dependent Policies

Having explained the basic privacy policy model, we now focus on discussing context-dependent policies. As described above, a CPE privacy policy has a "Context" field that essentially expresses the condition under which the policy is valid. This field is a collection of context-related predicates that must be satisfied for the policy to be considered active (e.g., Bob.location=home AND Alice.location=office).

#### 3.2.1 Context Field Specification

The Context field is an XML String containing an XQuery-compliant [16] predicate set over these attributes. CPE substitutes two keywords in the XML string during evaluation: "\$subject" and "\$requester", referring respectively to the Subject and Requester values supplied by the external service during an evaluation request. This allows the XQuery-based predicate set to be specified either in terms of these keywords or in terms of a specific user identity (e.g., "Bob"). Using the keywords allows the context predicates to be expressed in terms of the current requester or subject. For example, a Context field with the predicate: `"/user/$requester/location = 'office' AND /user/$requester/location/floor='first' AND /user/joe/location = 'office'"` indicates that the policy is valid only if the requester is in "office.firstfloor" and Joe himself is located in the "office".

#### 3.2.2 Context Field Restrictions

If users are allowed to specify arbitrary context predicates for the context-dependent policies, we can face problems where colluding users glean unauthorized context information from the system. For example, suppose Joe specifies a policy that "Bob is allowed to see my location if Alice is in the office". Then, the two friends Bob and Joe can implicitly deduce that Alice is located at the office by having Bob issue a request for Joe's location and verify if it is granted (even though Alice's policy itself may deny both Bob and Joe access to her location data).

One way to avoid such leakage is for system to prevent Joe from creating such a policy. In order to do that, the system needs to go through a global consistency check for every policy creation, and may need to invoke a revocation process. Therefore, this solution may introduce significant temporal dependencies in the policy creation process. Another alternative would be to have the CPE evaluation engine expressly check the permissions on Alice's location at runtime before considering whether the policy is presently applicable or not. (In this case, Bob would be unable to decide if a Deny response occurred because Alice was not in the office, or because she had prohibited access to her location information.) This approach, however, imposes significant performance bottlenecks on the evaluation process.

To enforce privacy without compromising on evaluation efficiency, CPE policies constrain the context predicates to only refer to attributes belonging to either the Subject or the Requester in the policy, or to the "\$subject" or "\$requester" wildcards that will be replaced by the specific user values during the evaluation process. These mechanisms however do not

prevent all covert data channels. For example, if the subject of a request learns the result of a query, the subject may infer the context of the requester at the time of the query (e.g., if Alice request's Bob's location and Bob has a policy of "grant if \$requester/location= office"). Likewise, if the requester learns (out-of-band) of the details of the subject's policies, the requester may infer the context of the subject at the time of the query. The system need to enforce that all the queries are protected to prevent info leakage.

CPE's restriction on the Context predicates prevent users from specifying potentially legitimate policies based on external context (e.g., user Joe exposing his location in case "fire-alarm=true"). In our system, such exposure may only be realized through required policies specified by an Administrator. Our experience clearly shows that, in practice, building a high-throughput privacy engine requires balancing the expressiveness of context with efficiency.

### 3.2.3 Absence of Context Information

Given the dynamic nature of context data, CPE must always retrieve the "freshest" context information from a context source and evaluate each request independently, instead of using policy or response caching [17]. In any realistic environment, context information will occasionally be unavailable (e.g. due to loss of source network connectivity or sensor failure). This reality must be addressed in the CPE architecture, as otherwise privacy preferences may be subverted erroneously. For example, the predicate "Bob.location= office AND Joe.location= office" will evaluate to "false" if Bob's location cannot be determined. A policy that has a "Deny" associated with this predicate will then not be considered to hold, potentially resulting in an inappropriate grant response. To avoid such situations, context-dependent policies with a "Deny" in the Result field are assumed to apply even if the context data cannot be obtained. However, a policy with a "Grant" in its Result field is considered inapplicable in the absence of context data. This approach ensures that "deny" prevails over "grant" in the absence of verifiable contextual information.

### 3.3 Policy Evaluation Algorithm

Having discussed context-dependent privacy policy model, we now describe how to evaluate a request based on those policies. A request to CPE is from an external service specifies {subject, requester, application, and information}, in other words, it asks "Can the application X being run by user A be granted access to the Information S about the subject B?"

The evaluation engine operates the following steps:

*Evaluate (requester A, subject B, Application X, Information S)*

1. Find set  $p_1$  containing all applicable policies, i.e., policies where the Subject fields contains B and Requester field contains X, and the Application field and Information fields refer to S and X or to less-specific values.
2. For all policies in  $P_1$ , evaluate the context predicate to form a set of policies  $P_2$  with the context field evaluated to be true ("\*" context is always true). //note: in absence of context, "Deny" policies evaluate to true, while "Accept" policies evaluate to false.
3. For all policies in  $p_2$ , get the Controlling (most-specific) policy set  $p_3$  by considering policy level and specificity.
4. Grant access if and only if all the Release values in the controlling policies are Grant, otherwise, deny.

Table 1 and Table 2 illustrate policy evaluation for a set of requests for President's location. Each column in table 1 specifies one privacy policy, and each column in table 2 represent a request.

**Table 1: Example Context-Dependent Policy DB**

ID	1	2	3
Subject	President	President	President
Requester	President.dept	President.friendsnfamily	Advisor1
Application	*	*	*
Context	\$subject.location = whitehouse AND \$requester.location = whitehouse	\$subject.location .room = whitehouse. Livingquarters	\$subject.location = whitehouse
Information	Location	Location	Location
Release	Grant	Grant	Grant

**Table 2: Context-Aware Policy Evaluation Example**

Requester	Spouse	Vicepres	Employee1	Advisor1
President's Location	Living quarters	Oval office	Oval office	Blue Room
Requester Location	NA	Green room	NA	Out of the country
Applicable policies	2	1	1,2	1,3
Controlling policies	2	1	2	3
Result	Grant	Grant	Deny	Grant

To simplify this example, we assume that all policies are at the same level in the policy hierarchy. In addition, the policies shown apply to all applications and a single type of context information (Location). Assuming that the two user-defined groups have the following members: President. friendsNfamily group

contains spouse and employee1, President.dept group contains Vicepres, employee1, and advisor1.

Let's look at each request (column) listed in table 2:

- Spouse's request is governed by policy 2. The request is granted if President is presently in the living quarters, but denied otherwise.
- Vicepres's request is governed by policy 1. If both President and the vice president are in the White House, the request will be granted and denied otherwise.
- Employee1's request is governed by policies 1 and 2. Assuming that Employee1's location is not available, the request will be granted by policy 2 if President is in the living quarters and denied otherwise.
- Advisor1's request is governed by policies 1 and 3. Assuming that Advisor1 is out of the country, the request is governed by 3 and will be granted if President is located in the White House.

#### 4. Performance Evaluation of CPE

We implemented the CPE engine as a Java-based application, running in its own JVM. The policies were stored in a relational database. To support context-dependent policies, we implemented our own XQuery recursive descent parser and predicate evaluator. Our implementation uses a context service described in [18]. We now report on studies used to evaluate the following performance metrics:

- What is the latency of a single CPE evaluation, and what is the impact of the number of policies?
- What is the additional overhead of policies that incorporate context predicates? How does this vary as the number of context predicates increases, or as the number of relevant context-dependent policies changes?
- How does the evaluation latency vary with the number of concurrent evaluation requests (an indicator of the system throughput)?

Our base test setup consists of three distinct servers, the CPE server (for policy evaluation), the context server (for retrieving context data needed for evaluating context predicates), and the directory server (for group membership). Each server machine runs the Windows 2000 Server OS, and had 4 1.5 GHz Intel® Xeon™ processors, each with 512 MB of memory. The basic test method consists of first defining appropriate policies and populating the database, and then computing the mean of 100 consecutive (sequential) evaluation requests of the form "Is <user A>, using <application X>, permitted to obtain <information Q> about <user B>?".

In our first experiment, we study the basic CPE evaluation latency (in the absence of contextual constraints), and the impact of number of applicable policies on the evaluation latency. Figure 3 plots the average evaluation latency (over 100 consecutive requests) as a function of the total number of policies pertaining to the subject in the database.

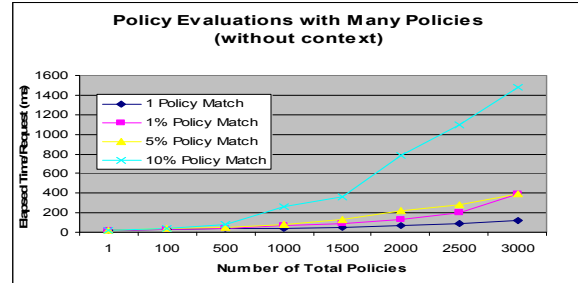


Figure 3: Latency vs. No. of Policies

We consider 4 cases, where the eventual number of *controlling* policies (i.e., those applicable policies that are in the highest policy level and are most specific) is either 1, or 1%, 5% and 10% of the total number of policies. As Figure 3 shows, the evaluation latency essentially depends on the number of controlling policies. CPE's design choice of eliminating policies by level and specificity, keeps the overall latency low, even if the number of applicable policies is quite large.

#### 4.1 Context-Dependent Policies

We now examine performance of context-dependent policies. To evaluate the additional overhead, we repeat the experimental setup of Figure 3, except that all the policies now have a Context field. Figure 4 shows the latency associated with 1, 1%, 5% and 10% most-specific policies. As we can see that introducing context in the policy *significantly* increases the evaluation latency.

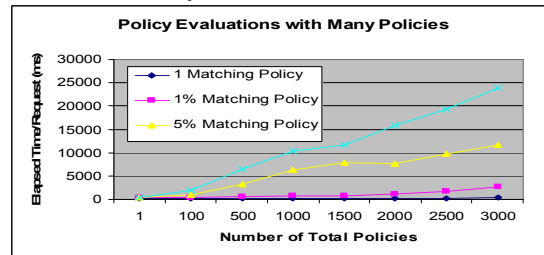


Figure 4: Latency vs. No. of Policies (with context)

It is important to understand that the overhead of actually evaluating the retrieved context data is negligible. To demonstrate this, Figure 5 plots the latency associated with a single context-dependent policy as a function of the complexity of the predicate. The figure demonstrates that the complexity of the predicate itself has little effect on the evaluation

latency (of course, having 0 predicates equals a context-independent policy and is much faster).

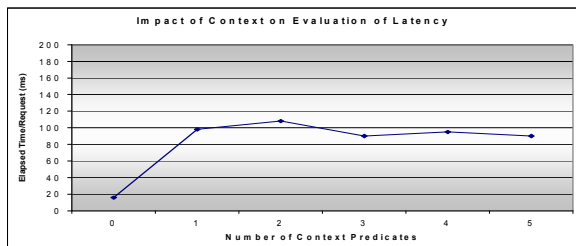


Figure 5: Latency vs. No. of Context Predicates

We argue that the sharp rise in latency with a context-dependent policy is not an artifact of CPE, but a reflection of the reality of today’s enterprise software. Many of the sources of context-data (such as presence, calendar information or location) are embedded within existing enterprise applications (such as instant-messaging or email), which were not designed to provide this information to external entities. Accordingly, the responsiveness of these systems to queries for “context” data quickly degrades in the face of even moderately large query rates. To demonstrate this effect, we ran a stripped-down “thin client” on the same machine as the context server (thereby also eliminating any network delays between CPE and the context server), and measured the average latency of a single request to retrieve different forms of context.

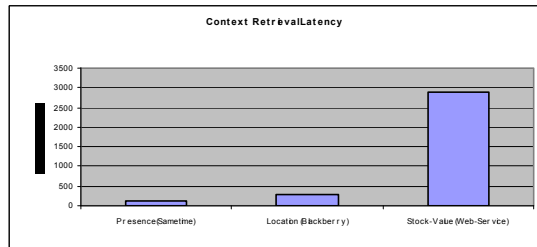


Figure 6: Latency of Context retrieval

Figure 6 shows this average latency for three different types of context—presence (obtained from an Instant Messaging service), location (from a Blackberry device) and real-time stock values (from a Web service). As this figure shows, enterprise-grade context sources show large variability in response times (100ms-3s) to queries for context. While some of this latency can be attributed to the lack of optimizations on the context server, this latency is often unavoidable due to the fact that proactive caching of context data is not trivial. Implementing a useful context caching solution would require a) policies to accommodate variable tolerance in the accuracy of the context data, which would permit the source to perform event-driven updates and largely avoid synchronous context retrieval, and b) making good predictions on the user

request patterns. Clearly, the overhead of context-dependent policies remains a reality in practical pervasive environments, until such an infrastructure for context sources is researched and developed in the future.

## 4.2 Overall System Throughput

To obtain the *system throughput* of CPE, we computed the average evaluation latency against a varying number of concurrent clients. Figure 7 shows the experimental results, for both context-independent and context-dependent policies, with all the clients synchronized to issue their requests at the same time. For the case of policies that are not context-dependent, we observe response times of less than 1 second, as long as the number of clients simultaneously issuing requests does not exceed 100. These numbers demonstrate that the CPE implementation can easily scale to about  $O(10,000)$  users, assuming that at most ~1% of users would simultaneously issue requests for privacy-sensitive data.

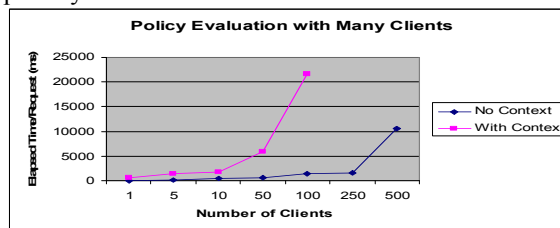


Figure 7: Avg. Latency vs. No. of Concurrent Requesters

However, when the privacy preferences are context-sensitive, the evaluation overhead increases sharply. The principal reason behind this is the additional latency incurred in retrieving data from the context server—as each context request takes longer to process, the number of available threads in the privacy engine decreases quickly. The evaluation latency then sharply increases, as each request has to incur a significantly larger queuing delay. Our performance results thus demonstrate that, in practical enterprise environments, the capacity of a system to deal with context-aware privacy preferences can be an order of magnitude lower than what it would for privacy preferences without any context dependence.

## 5. Conclusions

We have presented the design and evaluation of our CPE middleware, targeted at satisfying the privacy concerns that have often thwarted the deployment of several “much-touted” context-aware computing applications in enterprise environments. CPE accepts requests of the form <requester, subject, application, information> and goes through the set of existing

policies to decide if the access request should be granted or denied. While ACL-based privacy approaches have been presented before, CPE embeds several design features that are critically needed to support enterprise-scale deployments in practice. Overall, the twin notions of policy hierarchy and specificity provide both the requisite degree of control and scalability. In addition, we also saw how a practical implementation of CPE had to deal with the occasional unavailability of context data, and had to restrict the acceptable values of the Context field to prevent privacy leaking.

Our performance studies showed that the CPE evaluation algorithm is indeed able to support reasonably complex privacy policies efficiently. Our studies also show that *retrieving contextual information is quite expensive in current operating environments*. This is a fact that often seems to be neglected in discussions on context-based computing. Our experience suggests that context sources (such as IM or email applications) will also potentially need to be re-engineered to support a higher retrieval load. This observation suggests that context-aware policies may not immediately become as ubiquitous as originally perceived, and that the scope of context-aware policies should be judiciously limited to preserve overall response times for the time being.

## References

- [1] M. Reardon, "Mobile Phones that Track Your Buddies," *CNET.com*, 14 Nov. 2006.
- [2] Sandhu, R.S., et al., "Role-based Access Control Models." *IEEE Computer*, Vol. 29, No. 2, Feb. 1996, 38-47.
- [3] Covington, M., et al., "Securing context-aware applications using environment roles", *Proceedings of the IEEE Symposium on Security and Privacy* (Chantilly, Virginia May 2001), pages 10-20.
- [4] Hull R, et al., "Enabling Context-Aware and Privacy-Conscious Data Sharing", *IEEE International Conference on Mobile Data Management (MDM)*, (Berkeley, USA, January 2004).
- [5] Palen L, Dourish P., "Unpacking Privacy for a Networked World", *Proceedings of ACM Conference on Human Factors in Computing Systems CHI 2004*.
- [6] Howard J., et al., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, Feb. 1988, 51-81.
- [7] Salber, D, et al., "The Context Toolkit: Aiding the Development of Context-Enabled Applications", *Proceedings of CHI '99*, May 1999, ACM, 434-441.
- [8] Li N., et al., "Design of a Role-based Trust Management Framework", *Proceedings of the IEEE Symposium on Security and Privacy*, May 2002, 114-130.
- [9] Wagealla W., Terzis S., English C., "Trust-Based Model for Privacy Control in Context-Aware Systems", In *2<sup>nd</sup> Workshop on Security in Ubiquitous Computing*, October 2003.
- [10] Langheinrich, M., "A Privacy Awareness System for Ubiquitous Computing Environments", In *Proceedings of Ubicomp 2002* (Goteborg, Sweden, September 2002).
- [11] Hong, J., Landay J., "An Architecture for Privacy-Sensitive Ubiquitous Computing", the Proceedings of the International Conference on Mobile Systems, Applications and Services (Mobisys), 2004.
- [12] Canny, J., "Some Techniques for Privacy in Ubicomp and Context-Aware Applications", In *the Proceedings of the Privacy in Ubicomp 2002 Workshop*, September 2002.
- [13] L. Sweeney., "k-anonymity: A Model for Protecting Privacy", *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10 (5), 2002; 557-570.
- [14] Lederer S., "Everyday Privacy in Ubiquitous Computing Environments", *Privacy in Ubicomp'2002*, September 2002.
- [15] Kobsa A. and Schreck J., "Privacy through Pseudonymity in User-Adaptive Systems", *ACM Transactions on Internet Technology*, Vol. 3, No. 2, May 2003, 149-183.
- [16] W3C. XQuery 1.0: An XML Query Language, <http://www.w3.org/TR/xquery/>
- [17] Kaminsky M. et al., "Decentralized User Authentication in a Global File System", *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, (Bolton Landing, NY, 2003).
- [18] J. Black et al., "Pervasive Computing in Health Care: Smart Spaces and Enterprise Information Systems," *Workshop on Context Awareness, MobiSys 2004*, June 2004.
- [19] Sacramento, V. et al., "A Privacy Service for Context-aware Mobile Computing", *Security and Privacy for Emerging Areas in Communications Networks*, 2005, SecureComm 2005.
- [20] C-Y. Chow, and et al., "A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-based Services", In *Proceedings of the ACM Symposium on Advances in Geographic Information Systems, ACMGIS*, 2006.
- [21] A. Kapadia et al., "Virtual Walls: Protecting Digital Privacy in Pervasive Environments", *Proc. 5th Int'l Conf. Pervasive Computing (Pervasive 07)*, Springer, 2007, pp. 162-179.
- [22] C. Bettini, S. Mascetti, X. S. Wang, and S. Jajodia, "Anonymity in Location-based Services: Towards a General Framework," in *Proceedings of the International Conference on Mobile Data Management, MDM*, 2007.
- [23] B. Bamba, L. Liu, P. Pesti, and T. Wang, "Supporting Anonymous Location Queries in Mobile Environments with PrivacyGrid," in *WWW*, 2008.
- [24] A. Mitseva, et al., "Context-aware privacy protection with profile management", *Proceedings of the 4<sup>th</sup> international workshop on wireless mobile applications and services on WLAN hotspots*, LA, CA, 2006.