A Privacy Service for Context-aware Mobile Computing

Vagner Sacramento, Markus Endler and Fernando Ney Nascimento *
Departamento de Informática, PUC-Rio
R. Marquês de São Vicente 225
22453-900, Rio de Janeiro, Brazil

E-mail: {vagner,endler,ney}@inf.puc-rio.br

Abstract

*Privacy issues related to the access of context information are becoming increasingly important as we move toward ubiquitous and mobile computing environments. In this article, we describe the design and implementation of a privacy service, called **Co**ntext **P**rivacy **S**ervice (CoPS), to control how, when and to whom disclose a user's context information. Based on the results of an end-user survey and experience reported by other research groups, we identified the main service requirements and designed CoPS aiming flexibility, generality, simplicity and fine-grained privacy control. CoPS is an optional service of our context-provisioning middleware MoCA and allows users of context- and location-aware applications to define and manage their privacy policies regarding disclosure of their context information. The main features supported by CoPS are group-based access control, pessimistic and optimistic approaches for access control, hierarchical privacy rules, mixed-initiative interaction, and rule specificity analysis.*

Keywords:
Privacy, Context Service, Context-awareness, Middleware, Collaboration.

1 Introduction

Location-awareness, and context-awareness in general, have been identified as key elements of the mobile computing paradigm [12]. Consequently, many context-provisioning middleware infra-structures have been developed for mobile and wireless networks. But surprisingly only a few works have dealt with the implications that access to context data has for personal information privacy.

In our work, we are particularly interested in middleware support for distributed collaborative applications, where access to the computational context [13] and location information opens a wide range of new forms of end-user communication. For example, location information can be displayed by instant communication services, physical proximity among users can be used to select collaborating peers, and information about the connectivity status can enhance mutual collaboration awareness. This motivated us to design and implement a service-oriented middleware architecture called ***Mo**bile **C**ollaboration **A**rchitecture* - MoCA [10, 8] for collecting and processing context data from mobile devices in 802.11 networks, and making it available to applications. We are currently using this middleware to implement context-aware applications for mobile collaboration.

While some users may want to use such applications to facilitate group coordination or convey a sense of presence with friends or co-workers, there are also some serious concerns about the risk of disclosing personal context information. Hence, there is a huge demand for tools which provide end-users with the ability to define specific privacy control over context data. But since privacy is a very broad concept that entails very different interpretations and requirements, a context privacy solution should be flexible and adaptive to the specific needs of individuals, organizations, user communities, and applications.

In a recent survey we did with approximately 120 students we assessed the most frequent privacy concerns related to context and location data and identified the main requirements for context privacy [9]. This guided our design of a flexible and generic privacy control mechanism for context information, which we named "**Co**ntext **P**rivacy **S**ervice" (CoPS). This service allows the end-users to share their context data with the right people, at the right level and at the right moment through the following features: group-based access control, hierarchical privacy rules, rule specificity analysis (based on the requester ID, spacial and temporal precision, and information freshness), optional user notification, logging, and plausible deniability mechanisms.

*Partially supported by CNPq research grants no. 552.068/02-0 (Project ESSMA).

The remainder of this paper is structured as follows. Section 2 presents a discussion about some related work. Section 3 presents a short overview of MoCA and the type of context data that it makes available. Section 4 introduces CoPS' main features and describes the typical pattern of interaction among CoPS, a context service and an application. Then, Section 5 presents the main components of CoPS, giving emphasis on the structure of the privacy rules, the policy hierarchy, group definitions, the algorithm for rule specificity analysis, and some concrete scenarios illustrating how the algorithm works. We then discuss some details of our current prototype in Section 6 and show preliminary performance results in Section 7. Finally, in Sections 8 and 9 we discuss future work and give concluding remarks.

2 Related Work

Recently, research about mechanisms for privacy of control of context information has received increased attention. In the following we discuss the work most related to this field.

In Confab [4], people, places, things, and services are assigned to *info-spaces*, which are tuple-spaces storing static or dynamic context data about any of the info-space's entities. The context data stored in *info-spaces* are *contexts tuples*, and are populated by sources of information, such as sensors. Similar to other tuple spaces, an *info-space* supports *in-* and *out-methods*. *In-methods* affect what data is stored within an *info-space*, and include add and remove methods. *Out-methods* govern any data leaving an info-space, and include query, subscribe, unsubscribe, and notify. Privacy mechanisms are enforced through *info-space operators* (*in-* and *out-operators*), which govern what data can enter or leave the *info-space*. *In-operators* and *out-operators* are run on all tuples coming in/out. These operators can apply, for example, the info-space's access control policies to ensure that a tuple is allowed to be added/removed, or that this tuple should be blocked for reading or removal.

Project Aware Home [1] focuses only on home environments, where a variety of data about home residents and their activities is captured (by sensors), processed and stored. The access control mechanism uses an extension for Role-based Access Control (RBAC) [11]. Similar to the subject roles of RBAC, the authors defined *environment roles*, which can be used to capture security-relevant aspects of the environment in which an application executes.

In [14] privacy of location information is described and controlled by simple rules based on set theory. Each rule establishes a list of users who are allowed (or disallowed) to know the location of a user for a given period. The rule specifies the authorizations based on one of four visibility modes: *Visible to All, Invisible to Some, Visible to Some*, and *Invisible to All*, and using boolean operators *AND* or *OR*. When conflicting rules (e.g. R1 grants but R2 denies access), are combined using *AND*, the location information is not made available, and when using *OR*, it is made available to the requester.

Most of the approaches exclusively focus on location context, and propose specific mechanisms for controlling access and disclosure of this information. We however, take a broader approach considering that *any* context data, specially computational context that can be automatically collected and which apparently does not reveal relevant information, should be subject of access control.

Most of the related work adopts a centralized approach for storing and controlling access to context data, even though this surely gives end-users less control of their context information. Only [4] proposes a fully decentralized approach. Although this seems more reasonable from the end-user perspective, which does not have to trust a centralized context-provisioning infrastructure, it entails some problems when this context information has to be shared (albeit in a controlled way) among many users. Due to the huge amount of resources necessary to store, process and distribute this information, and the intrinsic limitation of mobile devices, it turns out that, at least with our current technology, only centralized approaches are feasible from a software engineering point of view.

Moreover, network use experience of the past decades has shown that despite the real threats of using unknown and remote services, users have largely trusted network infrastructures because of the obvious benefits that they gain. In fact, in most of our daily activities we, *de facto*, rely on social protocols and law enforcement, and expect that other people will indeed obey the rules and follow the social norms.

3 MoCA's Context Provisioning Services

The *Mobile Collaboration Architecture* (MoCA) [10, 8] consists of client and server APIs, basic services supporting context acquisition, storage and processing, and a framework for implementing application proxies (*ProxyFramework*). The APIs and the basic services have been designed to be generic and flexible, so as to be useful for different types of context-aware collaborative applications, e.g. synchronous or asynchronous interaction, message-oriented or artifact-sharing-oriented. MoCA is intended for use in an infra-structured wireless LAN (such as 802.11), and the current version runs on WinXP/CE and is based on TCP/IP.

In MoCA the following are the core services and components responsible for probing, storing and inferring computational and location context.

- *Monitor*: is a daemon executing on each mobile device that is in charge of collecting data concerning the device's execution state/connectivity, and sending this data to the *CIS* (*Context Information Service*) executing on one (or more) node(s) of the wired network. The collected data includes the quality of the wireless connection, remaining energy, CPU usage, free memory, current Access Point (AP), list of all APs and their signal strengths that are within the range of the mobile device.

- *Context Information Service (CIS)*: is a distributed service where each CIS server receives and processes devices' context data, sent by the corresponding *Monitor*s. It also receives requests for notifications (aka subscriptions) with SQL-like, context-based *interest expressions* from applications, and delivers notifications to the applications whenever the corresponding interest expression matches a new state of the context variables. Alternatively, applications may also query *CIS* directly about the current value of a particular context variable of a specific device.

- *Location Inference Service (LIS)*: infers the approximate *symbolic* location of a device. It does this by comparing the device's current pattern of RF signals received from *CIS* (from all "audible" 802.11 Access Points) with the signal patterns previously measured at pre-defined *Reference Points* in a Building or Campus. For this, *LIS* periodically queries *CIS* to update the device's pattern of RF signals. Since the RF signal is subject to much variation and interference, the location inference is only approximate: its precision depends on the number of access points and the number of the reference points. *LIS* allows the administrator to define symbolic regions of arbitrary size and shape, and a hierarchical description of regions with its nested sub-regions. Similarly to *CIS*, this service also supports both direct queries about a device's location, and subscriptions for notifications of location changes.

So far, we have used MoCA's API and services to develop some context- and location-aware prototypes of collaborative applications, such as NITA, WhoAreYou?(WAY), BuddySpaceLive, WirelessMarketingService, and others [7]. When we demonstrated these applications to other students and faculty, we realized the importance of privacy issues related to context information, i.e. while context data is useful for implementing adaptable and context-aware applications, it can also be used to derive information of where and how a user is using his device. Table 1 shows all computational and location context variables made available by MoCA's services, and some privacy risks of disclosing each such information.

Although our original goal was to develop a privacy mechanism for MoCA's context-provisioning services *CIS* and *LIS*, we later realized that it would be better to design CoPS as an independent and generic service that could be easily integrated with other context-provisioning services.

Table 1. Context data collected by MoCA

Context Variable	Privacy Risk
CPU usage (%)	gives a clue about the device processing load
Free Memory (in kB)	gives a clue whether the user's device is short of resources
Battery Power (%)	gives an estimate for how long the device will be available
IP Addr/Mask	network point of attachment, owner's network access rights, and rough information about device location
Current AP's MAC-Addr, RSSI	connectivity status and rough information about device location
List of all APs in the range	gives a clue about device's approximate location
LIS' Symbolic location	device's approximate location

4 CoPS Overview

CoPS is in charge of controlling when, how and to whom context data will be disclosed. This service implements an engine that evaluates privacy policies and checks whether access to context data from one *subject* (i.e. user) will be granted to a specific *requester* (i.e. a user or application). A privacy policy is set up by a *policy maker*, which may or not be the subject himself. By using a policy management GUI, the policy maker specifies the rules that dictates the access restrictions to the subject's context information.

The proposed service implements fine-grained access control, feedback and logging mechanisms, which give the subject different means of avoiding abuse of his context information usage. For instance, by setting the notification option in his rules (in addition to the appropriate access control) Bob would probably prevent others, for example his boss, from trying to periodically query his location. The feedback mechanism may use any appropriate form of notification, such as e-mail, SMS, IM, etc. In addition, having access to CoPS' log, Bob would be able to check who accessed (or could not access) his context data, when and how many times it occurred, etc. Feedback and logs have also been identified elsewhere [4] as a simple, yet effective, means of controlling access abuse through social visibility. For example, it is less likely that a boss will repeatedly query an employee's location if he knows that the employee gets notified at every request, and moreover can use the log to prove the abuse, and blame him of this action.

In CoPS, privacy policies are organized in a three-level hierarchy: organization-specific, individual-specific, and default policies. In this hierarchy, the organization-specific policy overrides the individual-specific policy, which in turn, overrides the default one. Hence, for organization-specific policies the policy maker may not be the same as the subject (e.g. the employee). For example, a policy maker responsible for a university may define that the lo-

cation of each member of a department staff must be made available to the director to facilitate delegation of tasks or finding a nearby member of the network support team to fix a problem of the secretary's desktop.

CoPS also supports two general approaches to specify a default access policy, an optimistic and a pessimistic one. With the pessimistic approach, by default all requests are denied, except those that match some rule specified by the policy maker. In contrast, with the optimistic approach, by default all requests are granted, except those matching some rule specified by the policy maker. Thus, end-users need only define the rules specifying under which circumstances their personal information should be disclosed or not, depending on default approach used. For example, the user may define a rule denying access to his location data to any requester which is not affiliated with his department. Furthermore, end-users could set up rules that define side-effect actions (e.g. logging, notifying), related to a specific context variable, (group of) requester(s), time of the day, etc.

We believe that most of CoPS users will be interested in disclose their context information to take off advantage of applications and services. In this case, the optimistic approach is easier to use, because the users can hardly predict all possible scenarios for which he wants to grant access in the pessimistic default policy.

For each access policy the policy maker can set up a set of candidate privacy rules to evaluate the requests. Hence, the set of applicable rules to a given request will be selected according to the subject's current access policy. The subject may change his current access policy through the policy management GUI.

By supporting these two approaches for default policies CoPS gives end-users a convenient, simple and flexible means of defining their rules according to their individual privacy preferences. Unlike others works [4, 3], CoPS' dual approach helps to reduce the number of rules necessary to define a privacy policy. After choosing either the pessimistic or the optimistic approach the policy maker will have only to specify a few rules, each of them producing the following results: "Grant" or "Deny" (but not both), "Not Available" or "Ask Me".

The policy maker may use the "Not Available" result when he wants to take advantage of plausible deniability, since this is also the default result for a request when context information is in fact not available. Returning "Not Available" as the result of a request thus enables a subject to make "white lies" where he in fact denies access, but does not make this explicit to the requester, who will not know whether the context information could not be obtained due any technical failure, due to access restriction, or lack of the data.

The result "Ask Me" is used when the user wants to be asked, *on the fly*, about the request. This feature will be discussed with more detail in Section 4.2.

4.1 Typical Interaction Pattern

The Context Privacy Service comprises a server (CoPS) and two clients APIs (CoPS Client APIs). The first one, *(Context Access Authorization (CAA) API)*, is used by the Context Service to communicating with CoPS server. The second one, *(User and Policy Management (UPM) API)*, is used by client applications at the subject's and requester's side for authentication and session token generation, for checking the consistency of policy rules, for accessing and analyzing the log, and for receiving queries asking the subject for a final decision regarding an access request.

Figure 1 illustrates the general architecture, and shows how its components interact with each other and with the context provisioning service (Context Service). In the case of MoCA, both *CIS* and *LIS* act as the Context Service, while the *Monitor* would be executing on the subject's device.

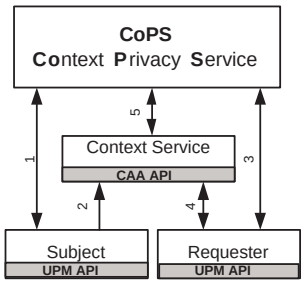

Figure 1. CoPS **general Architecture**

Initially, (1) the policy maker (e.g. the subject) has to define the privacy policy to be applied to a subject's context data: he chooses the default access policy (optimistic or pessimistic) and uses the policy management GUI to write the corresponding privacy rules. In parallel, (2) context data from the subject's device will be periodically received by the Context Service but will only be disclosed upon evaluation of the appropriate privacy policy. (3) Before a requester is able to submit an access request, he has to authenticate himself with CoPS. This authentication will produce a session token which will be used to create an **U**ser **I**dentification **T**oken (UIT) for future requests. The UIT is the hash of the session token. The generation and distribution of user session tokens will be explained in Section 6. (4) When the access request arrives at the Context Service, it will forward the request and the UIT to CoPS and wait for the result. (5) If the requester is successfully authenticated and the request is granted, CoPS replies with a "Grant", otherwise with a "Deny" or "Not Available" result.

4.2 Controllable Properties

According to the results of our survey about requirements for privacy control [9] and related work [4], most users demand means of interactively deciding when requests should be granted or not. In other words, in this approach of interaction, called mixed-initiative, end-users are interrupted and are asked *on the fly* to decide whether to grant or deny the request. If the policy maker set up the privacy rule with the result value "Ask Me", the CoPS server will forward the request to the subject's application. The application may then present to the user (the subject) a human-readable representation of the request and ask him for a decision, such as: "Can requester A be granted access to the context information I?". And the answer options could be, for example, "Always allow", "Just this time", "Only for the next 2 hours", "Never Allow", etc. The CoPS server waits some time for the reply, and if no reply is received, CoPS simply denies access to the context information returning the default reply value "Not Available".

CoPS also supports adjusting the precision of the dynamic contextual information being disclosed. It does so by allowing the policy maker to specify a *spatial precision*, *temporal restriction* and *freshness* of the contextual information in the privacy rules. For example, consider a scenario where some service provides location information. Alice is sharing her location with classmates, but maybe does not feel comfortable letting them know precisely where she is. In this case, she would be able to adjust the level of disclosure by defining the spatial precision of her location information (e.g. "PUC-Rio" or "Department of Computer Science Building" instead of "Room 205"). She could also set some temporal restriction, by defining, for example, the time interval (e.g. "9:00 to 11:30 am AND Monday to Thursday") when the information should be made available. Moreover, she could also specify the freshness of the disclosed information, determining that instead of her current location, only her location 30 minutes ago shall be disclosed.

5 CoPS' Architecture

The service has been designed to offer fine-grained and flexible control over privacy policy evaluation, using the following components: the Privacy Policy Engine, the Dynamic User Management and Access Control (DUMAC), a notification dispatcher and the client APIs. Figure 2 illustrates the main components of the CoPS architecture.

As mentioned in Section 4, the client APIs hides from the Context Service and application developer many details related to CoPS-specific interaction and processing. By designing CoPS independently from the Context Service, we obtain more flexibility and reduced complexity. Flexibility, in the sense that CoPS becomes independent of a specific Context Service, and that Privacy Management can be incorporated as an optional and complementary feature of a context provisioning middleware. Thus, the Client API is very important for enabling a simple and transparent integration of CoPS with both the Context Service and the application clients.

In order to provide support to the mixed-initiative interaction, the CoPS server and the application client API use event-based asynchronous communication. The client API subscribes itself at the CoPS server informing the address in which it is supposed to receive requests for the subject's final decision (Grant or Deny). After receiving a request from the CoPS server, the client API forwards it to the application client and waits some time interval to send a reply. If the CoPS server receives a reply from the client API before the timeout, the subject's decision is sent to the context service, otherwise, the default reply ("Not Available") is dispatched.

The Dynamic User Management and Access Control (DUMAC) component is in charge of implementing user authentication and management of groups and users. Although CoPS offers its own authentication method, in principle it can be integrated with any other similar authentication system, such as NIS, SAMBA or Windows Domain Controller, facilitating the deployment in different administrative domains.

The Policy Evaluation is the central component within the CoPS server. It processes the access request taking into account all privacy policies related to a subject. It first selects the rules of the default access policy chosen by the policy maker, and then evaluates policy specificity, by selecting the most specific rules that match a given request. Based on the set of selected rules, it then checks and resolves possible conflicts in order to compute the final result ("Not Available", "Ask Me","Grant" or "Deny"). The result is then returned to the client API at the Context Service.

5.1 Structure of the Privacy Rules

The structure of a CoPS privacy rule is composed of several fields, which are also present in the requests. Any privacy rule is associated with a default access policy (optimistic or pessimistic). This must be chosen in beforehand by the policy maker, and it will determine the basic evaluation algorithm for each request. The proposed rule fields and their semantics are described as follows.

- *Policy Maker*: Individual who defined/created the privacy rule (may or may not be the same as the subject).
- *Subject*: User or entity whose context data is controlled by this rule.

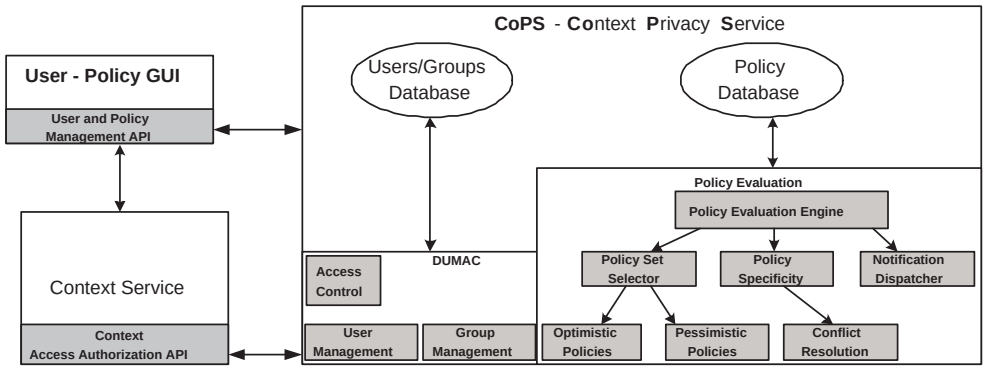

Figure 2. CoPS **Architecture**

- *Requester*: User or software component requesting access the subject's context data.
- *Context Variable*: The specific type of context data being requested (e.g. location, energy level, IP address, etc.).
- *Application*: List of application names that can be used by the requester to access the context variable. The wildcard '*' represents any application.
- *Precision*: Specifies the value precision of the context variable (e.g. for location information, this could be the spatial precision like state, city, ZIP code, building, room, etc.).
- *Temporal Restriction*: Date and time interval restrictions for disclosing the context information (e.g., weekdays, from 9 am to 6 pm).
- *Freshness*: Specifies the freshness (in milliseconds) of the disclosed context information (e.g. location 15 minutes ago, or current location). The default value is 0 ms.
- *Timestamp*: Specifies the time in which the privacy rule has been created. This field is used by the result specificity to resolve possible conflicts among the rules.
- *AccessPolicy*: Represents the access policy (Optimistic or Pessimistic) that this privacy rule is associated with.
- *Policy Level*: Hierarchy level of this rule. Initially, CoPS will support only following three possible values "organization", "individual" or "default".
- *Result*: Outcome of applying this rule to a request. Possible values: "Not Available", "Ask Me", "Grant" and "Deny".
- *Notify Me*: If the policy maker wants to be notified when the rule is applied. The options available are "NoNotification", "E-Mail", "ICQ", "MSN" or "SMS".

5.2 Group Definitions

Groups provide an additional facility for the management of privacy rules and also decrease the processing effort during evaluation of the requests. The *Subject* or *Requester* fields of a privacy rule can be either individual users or groups.

There are two general categories of groups: *administrator* and *user-defined* groups. The first ones are structured hierarchically to reflect the organizational structure, and define the corresponding user roles, similar to RBAC [11]. Groups in a higher level of the hierarchy include all of its descendant groups at a lower level, e.g. the group "puc.employee" comprises the group "puc.employee.prof", which in turn comprises the group "puc.employee.prof.cs". *User-defined* groups, are not hierarchical for the sake of efficient evaluation and maintenance.

Initially, all users in CoPS belong to the group "Anonymous", which facilitates the specification of access rules for unknown users, i.e. the policy maker is able to set up a privacy policy for unknown ("Anonymous") requesters. Moreover, this group can also be used for anonymity, i.e. users can send a request as an "anonymous user" if they want to hide their real identity.

5.3 Policy specificity

During the evaluation process, more than one rule may match the request, for many reasons. For instance, when the requester belongs to several groups mentioned in field "Requester" in some rules (e.g. "Alice" belongs to groups "Coworker" and "MyFriend"), then all these rules match the request. CoPS' specificity algorithm aims to determine

the *most specific* privacy rule that applies to a request and, if necessary, resolve possible conflicts among the rules.

The specificity algorithm works as follows: Given a set of rules previously selected (by the engine) to evaluate a request, the algorithm identifies the most specific rule of the set by comparing their structure fields in the following order of priority: *Subject*, *Requester*, *Context*, *Temporal Restriction*, *Precision*, *Application* and *Result*. When comparing rules with respect to a field, only the ones with the most specific value in this field are selected for the further specificity analysis, while all other rules are not considered for selection. This way, even if two or more rules have different relative specificity (i.e. they differ in two or more fields) the algorithm can identify the most specific rule analyzing these fields according to their priorities. For all fields, wildcard "*" means least specific.

For the specificity of the *Subject* and *Requester* fields, privacy rules mentioning an individual user (e.g. "Alice") are more specific than rules containing a user-defined group (e.g. "MyFriend"), which in turn is more specific than the ones mentioning an administrator-defined group. The administrator-defined group specificity follows the usual interpretation of a hierarchy: groups at a lower hierarchy level are more specific than groups at a higher level (e.g. "puc.employee.prof.cs" is more specific than group "puc.employee.prof").

The same hierarchy-induced specificity applied to the administrator-defined group is used also for the *Precision* field[1]. For example, when comparing rules concerning location information, the most specific ones are those where field *Precision* mentions the lowest level in the location-hierarchy, e.g. "country.state.city.zip" (level 4) is more specific than "country.state.city" (level 3). Two or more privacy rules can be at the highest level of specificity with regard to their *Precision* field if they have the most specific value, and are at the same level in the hierarchy. When this happens, the next field (according to the priority) of these rules is compared to identify the most specific rule. In order to allow for such specificity analysis the developer of the Context Service has to define the syntax (e.g. campus.building.floor.room) of the name hierarchy for this specific field. It will be a configuration parameter of CoPS.

The field *Temporal Restriction* represents the time interval and date at which the requester is granted or denied access to the context information, depending on the access policy approach used (optimistic or pessimistic). This field is very useful when the user wants to restrict the access in some special situations (e.g. at lunchtime or at working hours). Even though the policy maker specifies a time interval (e.g. "9:00am-11:30am"), CoPS represents it in milliseconds, to allow for an accurate rule selection. The specificity for this field is evaluated in three phases: (1) select the rule(s) that match the time and date of the request; (2) identify the rule with the largest time interval and check whether the time interval of the other rules are its proper subsets (e.g. Temporal Restriction "Feb 5, 10:30am-2:00pm" is a proper subset of restriction "Feb 5, 10:00am-6:00pm"). Rules are considered to be at the same level of specificity either if they have identical time intervals, or if the time interval is not a proper subset of the largest time interval; (3) select the rule with the smallest time interval, when they are not at the same level of specificity.

With regard to the field *Application*, specificity has only two possible levels: any application (represented by "*") and a list of applications. Finally, if all previously considered fields are at the same level of specificity, the *Result* field is the one used to select the most specific rule to evaluate the request. The possible values for this field are: "Not Available", "Ask Me" and "Grant" (or "Deny"). The "Not Available" result has precedence over "Ask Me", which in turn has precedence over the others (i.e. result "Not Available" is more specific than "Ask Me", which in turn is more specific than "Grant" and "Deny"). The reason is that "Not Available" implicitly means "Deny"and "don't let requester know it", while "Ask Me" may be interpreted as "Deny" or "Grant", depending on my mood. A conflict is detected when there is more than one rule with a result "Not Available" or "Ask Me", or when all rules have either a "Grant" or "Deny" result. In this case, the last rule with greatest specificity created by the policy maker will be selected. It is necessary to define a deterministic choice for these situations because the conflicting rules may have different notification methods and only a single rule must be chosen to evaluate the request.

5.4 Privacy Policy Evaluation Example

In this section, we show an example of possible privacy rules for user Bob, assuming that the pessimistic default policy has been chosen, i.e. whenever a request does not match any rule, it will be denied. These rules (shown in Table 2) determine how and when Bob's location and energy context variables will be disclosed. In this example, we also assume the existence of some *user-* and *administrator-defined* groups (Bob's and PUC's groups are shown in Table 3), which are mentioned in some of the rules.

Table 3. Assumptions about User Groups

Assumptions		
	Group	**Members**
user-defined	Bob.MyFriend	Bob, Alice, John
	Bob.Coworker	Alice, Jane, John
administrator-defined	Puc.Student	Bob, Alice, Jane, John
	Puc.Manager	Jane, Paul

[1] Although a hierarchy-induced notion of precision is more easily understood in terms of location information, it can be applied also to other context information, such as sub-domains of an IP address.

Table 2. Example rules.

Rules	Access Policy	Subject	Requester	Context Variable	Temporal Restriction	Precision	Application	Result	Freshness	Policy Level	Notify Me
R1	Pessimistic	Puc.Student	Puc.Manager	Location	*	puc	Ap1	G	0	O	e-mail
R2	Pessimistic	Bob	Puc.Student	Energy	09:00am to 06:00pm	*	*	G	5	U	ICQ
R3	Pessimistic	Bob	MyFriend	Energy	09:30am to 12:30am	*	*	G	0	U	ICQ
R4	Pessimistic	Bob	Coworker	Energy	11:00am to 02:00pm	*	*	NA	0	U	NoNotify
R5	Pessimistic	Bob	Coworker	Location	09:00am to 12:00am	*	*	G	0	U	NoNotify
R6	Pessimistic	Bob	Alice	Location	09:00am to 11:00am	campus.building	*	G	0	U	MSN
R7	Pessimistic	Bob	Alice	Location	10:00am to 04:00pm	campus.building.floor.room	*	G	15	U	e-mail

Through some scenarios, we will now explain how the privacy rules are selected and used to evaluate a request, using the algorithm explained in Section 5.3.

As already mentioned, the rule to be applied to the request is always the most specific one, and comparison of the rule's specificity takes into account the fields *Subject*, *Requester*, *Context*, *Temporal Restriction*, *Precision*, *Application* and *Result*, in this order. Thus, intuitively, the algorithm compares the values in the corresponding columns (from left to right), and as soon as one (or several) rules have a more specific value in one of the columns, they are candidate for further comparison.

Scenario1: If Jane makes a request for Bob's location, both R1 and R5 would apply. However, the request would be granted by R1, because this rule belongs to a higher level than rule R5 and, consequently, the first rule overrides the others.

Scenario2: Consider a request from John to get the energy level of Bob's device. In this case, R2, R3 and R4 are the related rules. But among those, rules R3 and R4 are selected because the user-defined groups mentioned in these rules are more specific than the administrator-defined group of R2. Finally, the request will be evaluated by R4 because, despite their fields *Requester*, *Temporal Restriction*, *Precision* and *Application* having the same level of specificity, their *Result* value differs, and "Not Available" has precedence over "Granted".

Scenario3: For Alice's request to get Bob's location rules R5, R6 and R7 should be examined. Among those, R6 and R7 take precedence over R5 because they apply to an individual user, "Alice", rather than to a group, as specified by R5. Although the R6 and R7 are at the same level of specificity in the *Temporal Restriction* field, R7 is more specific than R6 in the *Precision* field, and therefore will be applied to grant the request.

6 Implementation

So far, we have developed a first version of CoPS [6], which includes the Privacy Policy Engine that implements the full-featured specificity algorithm, the DUMAC component supporting symmetric authentication, administrator- and user-defined groups, the client APIs and a Policy Management GUI.

CoPS has been implemented in Java and used MoCA's communication API and Event service for the interaction between the CoPS server and the client APIs. We carefully structured and coded the Privacy Policy Engine so as to maximize the efficiency of the privacy rules evaluation.

6.1 Implementation Issues

CoPS follows the client/server paradigm where the interaction is either through synchronous or asynchronous, encoded or non-encoded communication. The requester's and subject's applications employ the User and Policy Management API (UPM) to authenticate the user and add/remove/query users, groups and privacy rules, while the context service utilizes the Context Access Authorization API (CAA) to control access to the context information. The client APIs also provide interfaces for secure message exchange with the CoPS server via TLSv1 channels.

Initially, the user authenticates himself with CoPS through UPM API, which in turn establishes a secure TLSv1 channel to send the authentication request. This authentication will produce a session token, which will be used to create an **U**ser **I**dentification **T**oken (UIT) for future requests. The UIT is a hash[2] of the session token shared among the client and CoPS during authentication. From this point on, CoPS will perform symmetric authentication using this token shared with the client, i.e. each client request must contain the UIT so that CoPS can authenticate it by comparing the UIT with the hash of the corresponding session token shared with the client. For guaranteeing the client authenticity, at each request the client increments the session token by one and regenerates the UIT from the new value, and the same is done by the server. This protocol obviously requires the server and client to be synchronized with regard to the the current session token. If some synchronization problem occurs (e.g. a message is lost), the client APIs will try to synchronize the session token such

[2]Currently, we are using the SHA-1 to generate the hash of the token.

that the communication is not interrupted by an authentication failure.

The authentication process is implemented by DUMAC, which is responsible for managing all operations (add/remove/query) concerning users and groups. In order to optimize such group and user management operations, DUMAC stores the corresponding information both in memory (as a hash table) and in persistent storage, and keeps these repositories synchronized.

The Policy Specificity component uses DUMAC for identifying to which groups created by the subject the requester belongs. This information is necessary to check which rules match a context access request, as explained in Section 5.3. To evaluate a request the Policy Evaluation Engine component (illustrated in Figure 2) invokes the `runPrivacyPolicySpecificity()` method from the Policy Specificity component, which in turn returns the most specific rule which applies to the request, or NULL if none of the subject's rules applies. The pseudo-code of the algorithm used to evaluate the specificity of the rules is outlined in Algorithm 1, where variables $Xi, i = 2, .., 4$ denote group identifiers, and $Ni, i = 1, .., 5$ are sets of rules with increasing specificity.

6.2 Privacy Rules Specificity Algorithm

After receiving a request with the arguments Subject, Requester, Context Variable, Precision and Application, the specificity algorithm evaluates the rules that apply to the request and returns either an empty set, or a set with the most specific rule. An empty set means that no rule applies to the request, and hence the subject's default access policy (Optimistic or Pessimistic) will be used to evaluate the request. If some rule is returned, the request will be evaluated according to the value specified in the rule's result field.

Initially (step 2 in Algorithm 1) the rule specificity is analyzed at the three policy levels (Organization, Individual and Default), in this order. At each policy level, it will evaluate the matching with respect to the five possible associations between the Subject and Requester fields in a privacy rule. For example, in step 2.1 the method `evaluateRulesSpecificity()` will evaluate the rules specificity that match the subject and the requester; in the step 2.3 this same method will evaluate the rules specificity that matches the subject and subject's groups to which the requester belongs, and similar evaluation will be done with the other groups in steps 2.5, 2.7 and 2.8.

The `evaluateRulesSpecificity()` method implements the specificity analysis based on subject and requester sets, and the other request parameters received as argument. In step 4, it makes a SQL query for retrieving the applicable privacy rules for a particular request. This step implements the user and context specificity criteria, by selecting, for further analysis, all rules that match the subject or groups in SubjectSet and requester or groups in RequesterSet, the Context Variable and the Subject's Access Policy. In the following steps, it performs the rule selection according to the temporal restriction, precision, application and result specificity. Every specificity phase returns an empty set if no rule is selected for the further specificity analysis. As mentioned in Section 5.3, the specificity analysis of the result field is necessary in the case that at the end two or more rules remain at the same level of specificity.

Algorithm 1 Privacy Rules Specificity Algorithm

RUNPRIVACYPOLICYSPECIFICITY(Subject S, Requester R,
Context Variable C, Precision P, Application A)
1. Set X1, X2, X3 and X4 = Ø
1.1. Let AP = Subject's Current Access Policy

2. Loop over the 3 policy levels, starting at the highest level (Organization):

2.1. X1 = evaluateRulesSpecificity (S, R, C,P, A, AP)
if (X1 != Ø) return X1

2.2. Let X2 = subject's groups which the requester belongs to.
2.3. X1 = evaluateRulesSpecificity (S, X2, C,P, A,AP)
if (X1 != Ø) return X1

2.4. Let X3 = administrator's groups which the requester belongs to.
2.5. X1 = evaluateRulesSpecificity (S, X3, C,P, A,AP)
if (X1 != Ø) return X1

2.6. Let X4 = administrator's groups which the subject belongs to.
2.7. X1 = evaluateRulesSpecificity (X4, requester, C,P,A,AP)
if (X1 != Ø) return X1

2.8. X1 = evaluateRulesSpecificity (X3, X4, C,P,A,AP)
2.9. if (X1 = Ø& level != Default) go back to Step 2
else return X1

EVALUATERULESSPECIFICITY(Set SubjectSet, Set RequesterSet,
ContextVariable C, Precision P, Application A, AccessPolicy AP)
3. Set N1,N2, N3, N4 and N5 = Ø

4. Let N1 = all privacy rules for which the subject(s) in SubjectSet is identical to the Subject field; And for which the requester(s) in RequesterSet is identical to the Requester field; And the Context Variable C is identical to the ContextVariable field; And the AccessPolicy AP is identical to the AccessPolicy Field.
4.1. if (N1 == Ø) return Ø

5. N2 = runTemporalRestrictionSpecificity(N1)
if (N2 == Ø) return Ø

6. N3 = runPrecisionSpecificity(N2, P)
if (N3 == Ø) return Ø

7. N4 = runApplicationSpecificity(N3, A)
if (N4 == Ø) return Ø

8. N5 = runResultSpecificity(N4)
9. return N5 containing one (the most specific) or none rule

6.3 Access Authorizations Caching

In its usual mode of operation the Context Service forwards the request and the UIT to CoPS whenever it receives an access request for the subject's context information. If the requester is successfully authenticated and the request is granted, CoPS replies with a "Grant", otherwise with a "Deny" or "Not Available" result. In order to reduce the response time of a context access request, we have implemented a cache holding CoPS' results of recent requests in the CAA API. This way, once a request from a requester R, to a given subject S, concerning a specific context variable C, to an application A with precision P has been evaluated, the context service can evaluate subsequent queries concerning (R, S, C, A, P) from the local cache.

The local cache managed by the CAA is completely transparent to the Context Service, and the developer of this service can decide whether he wants to use it, or not. When the CAA processes a new access authorization request from a context service that uses cache, it forwards the request to CoPS and subscribes itself at CoPS' event server as interested in being notified whenever the result of the evaluated request changes. This way, whenever there is a change of a privacy rule, the event service will evaluate if this modification invalidates the result of any of the subscriber's request. The event server will only analyze the requests that match the subject of the updated privacy rule. If the result value changed, the server will immediately notify the subscriber(s), such that the corresponding CAA's can update the cache with the new result of a given request.

After the subject has defined his privacy policy, we believe that the privacy rules will be updated only sporadically, and consequently, the number of notifications of cache updates will decrease proportionally. From the results of preliminary tests we could perceive that caching significantly reduces the number of queries to CoPS and also reduces the response time of the context access authorization.

The main problem of using a cache for the access authorization results is that there may be a short time interval between the update of a specific privacy rule result at CoPS and the corresponding delivery of an invalidation notification at the context service[3] leading to a potential breach in the subject's access control. However, it seems to us that the gain in performance of using the cache out-weights the relatively small risk.

7 Performance Evaluation

In this section, we describe some preliminary performance tests that we did with the purpose of measuring CoPS' scalability and throughput. In these tests, we used two machines for running the CoPS server and clients respectively, both of which were 2.4Mhz PentiumIV, 512MB of RAM, running WindowsXP Professional in a fast-ethernet local network.

In order to facilitate the implementation of our performance tests we used AspectJ [5] to instrument CoPS' code with instructions for logging the processing time of several parts within CoPS. In our experiments, we did not use CAA caching, and measured the response time of the rules' evaluation process with and without network latency. In addition, in our tests we used the symmetric authentication (using UITs) explained in Section 6 in order to get realistic performance results.

In essence, we wanted to analyze three questions. First, we wanted to measure how the response time increases as a function of the number of applicable privacy rules at all specificity phases. Second, we wanted to identify how response time increases as a function of the number of concurrent clients having a pre-defined amount of privacy rules analyzed at all specificity phases. Third, we wanted to measure the latency of specificity evaluation algorithm disregarding the network latency.

In our first experiment, we populated the CoPS' database with a selected set of privacy rules, in such a way that the same amount of privacy rules would be selected at each specificity phase. This experiment aimed to identify how the increase of the most-specific rule set (i.e. applicable privacy rules analyzed at each specificity phase, including temporal restriction, precision, application and result) influences the latency. Figure 3 outlines the results of this experiment. In this test, we ran one client that made 100 consecutive requests and measured the average elapsed response time to evaluate the requests. We carefully set up the rules' fields so that each different test could select a specific amount of most-specific rules to be analyzed at each specificity phase.

From this test we identified that the total number of rules in CoPS' database do not have direct impact on the response time latency, because the SQL queries for retrieving the rules depending on the subject, requester, context variable and access policy already eliminate all the non-applicable privacy rules with low delay. In addition, the algorithm described in section 6 shows that each phase of the specificity analysis may eliminate some rules for the further analysis. Hence, it is important to note that the main bottleneck of the specificity evaluation algorithm is not the amount of applicable privacy rules selected via SQL query, but the number of privacy rules processed at each specificity phase. As shown in Figure 3 the response latency has linear increase with the number of most-specific privacy rules. From this experiment, we can see that when the most-specific set is large (about 200 most-specific rules at all specificity phases for a single request) the response

[3] In most cases, network latency is less than 2 seconds.

time is about 20ms. However, we believe this to be unlikely scenario, and that in practice, in worst-case, the specificity algorithm will not select and evaluate, at each specificity phase, more than 15 privacy rules for each request. Hence, we realized the following tests (shown in Figure 4) using a pre-defined set of 15 most-specific applicable rules.

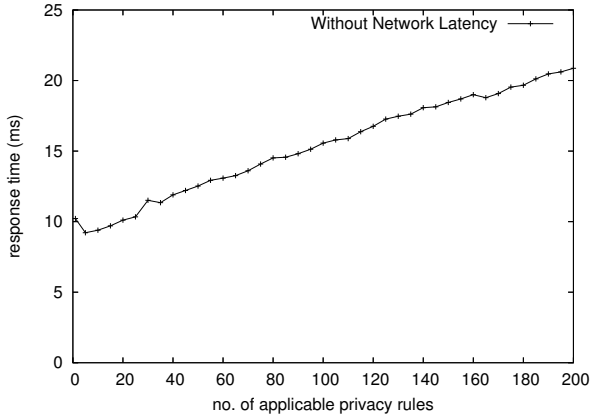

Figure 3. Response Time *versus* Number of Applicable Privacy Rules

Figure 4 shows the results of the second and third experiments, where we analyzed the average response time varying the number of concurrent clients. For these tests, we populated CoPS' user database with 301 users: 300 possible requesters and one subject user 'S1'. In order to reduce the amount of privacy rules, we joined the 300 possible requesters to 'MyFriend' group and created 15 privacy rules with the Subject and Requester fields holding 'S1' and 'MyFriend' values respectively. All these rules were at the INDIVIDUAL level and we assumed that the optimistic default access policy had been chosen. We also carefully set up the rules' fields so that all of them would be always selected/analyzed in all specificity phases for each request. We then ran an increasing number of concurrent requesting clients, where each client made the same request 100 times. Next we measured the response times with and without the network latency.

The only difference between these two experiments is that the last one evaluate the specificity evaluation algorithm latency and it do not consider the network latency.

The results (Figure 4) show a linear increase of the response time and algorithm latency when the number of concurrent clients is increased. Furthermore, the results show that the specificity evaluation algorithm latency has little influence on the total response time of a context access authorization request.

These preliminary tests seem to indicated that the overhead caused by the access control evaluation via CoPS represents only a small portion of the total round-trip-delay of context data access and processing at a Context Service.

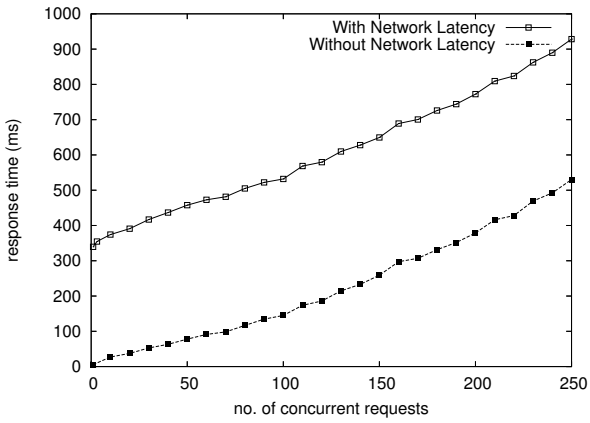

Figure 4. Response Time vs. No. of Concurrent Requests

8 Future Work

As a next step, we plan to extend CoPS' engine to handle also *context-dependent* privacy policies, allowing the policy maker to set privacy rules which depend on dynamic context data. For example, a policy could specify that access to some context data is granted only when the requester is within the university campus, or even in a specific building. In addition, we plan to develop a privacy policy management GUI, which supports end-users to define their privacy rules more intuitively. This GUI ought to be simple and effective to motivate end-user to adopt CoPS.

Furthermore, we are studying the "Platform for Privacy Preferences (P3P)" specification [2] with the purposes of using it to represent the privacy policy structure. P3P supports the encoding of privacy policies into machine-readable XML, making it easier to interpret these policies and execute the corresponding actions.

We also intend to develop a trust model [15, 16] for context-aware computing, exploring some properties of trust evaluation (e.g. diversity, transitivity, and combination), in order to facilitate the definition of privacy policies. For example, assuming the transitive property of trust we could have the following scenario "if Alice trusts Bob who trusts Jane, then Alice will also trust Jane". This way, Alice would not need to explicitly set up privacy rules to handle Jane's request. Instead, the system could be able to infer the Alice's risk level of disclosing her information to Jane, and it would be able to apply the appropriate privacy policy.

9 Conclusion

Since context-awareness has been recognized as a key element for the development of adaptive applications in mobile environments, many efforts have been made to design and implement context-provisioning middleware infrastructures. We have implemented such a middleware, called MoCA which we are now using to implement context- and location-aware applications for mobile and spontaneous collaboration.

Results of a recent end-user survey, where we assessed the acceptance of such applications and privacy concerns, helped us to identify the main requirements for a context privacy service. Then we have designed and implemented the Context Privacy Service (CoPS), trying to address all these requirements.

CoPS is intended as an optional, generic service to enforce the controlled access to context information. Prior to releasing any context information requested by a user or application, CoPS would be queried to decide if access to a subject's context is granted or denied. One of the most interesting feature of this service is its support for a rich set of options for privacy policies, such as user and organization-level rules, both optimistic and pessimistic default access policy, group-based rules and group management, specificity analysis considering the subject, the requester, spatial and temporal restrictions, information freshness, as well as the allowed applications.

References

[1] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 10–20. ACM Press, 2001.

[2] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. Platform for privacy preferences 1.0 (p3p) specification, April 2002. W3C Recomendation, HTML version at http://www.w3.org/TR/P3P/ (Last visited December 2004).

[3] J. Grudin and E. Horvitz. Presenting choices in context: approaches to information sharing. In *Workshop on Ubicomp communities: Privacy as Boundary Negotiation*, 2003.

[4] J. I. Hong and J. A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189. ACM Press, 2004.

[5] R. Laddad. *Aspectj in action: Practical Aspect-Oriented Programming*. Manning Publications Co, 2003.

[6] MoCA Team. COPS/MoCA Home Page, 2005. http://www.lac.inf.puc-rio.br/moca/cops.html (Last visited: February 2005).

[7] MoCA Team. MoCA Applications Home Page, 2005. http://www.lac.inf.puc-rio.br/moca/applications.html (Last visited: February 2005).

[8] MoCA Team. MoCA Home Page, 2005. http://www.lac.inf.puc-rio.br/moca (Last visited: February 2005).

[9] V. S. Rodrigues, M. Endler, and F. N. Nascimento. Design of a Context Privacy Service for Mobile Collaboration. In *Proc. of the Brazilian Symposium on Computer Networks (SBRC'05), Fortaleza*, May 2005. (to appear).

[10] V. Sacramento, M. Endler, H. K. Rubinsztejn, L. S. Lima, K. Goncalves, F. N. Nascimento, and G. A. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10):2, 2004.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[12] J. Schiller and A. Voisard. *Location-Based Services*. Morgan Kaufmann, 2004.

[13] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location. *Computers and Graphics*, 23(6):893–901, 1999.

[14] A. Smailagic, D. P. Siewiorek, J. Anhalt, D. Kogan, and Y. Wang. Location sensing and privacy in a context aware computing environment. In *Pervasive Computing*, 2001.

[15] W. Wagealla, S. Terzis, and C. English. Trust-based model for privacy control in context aware systems. In *Second Workshop on Security in Ubiquitous Computing at the Fifth Annual Conference on Ubiquitous Computing (UbiComp2003)*, October 2003.

[16] W. Wagealla, S. Terzis, C. English, and P. Nixon. On trust and privacy in context-aware systems. In *Second iTrust Workshop*, September 2003.