A Context-Aware Privacy Policy Language for Controlling Access to Context Information of Mobile Users

Alireza Behrooz[1] and Alisa Devlic[2]

[1] Appear Networks, Kista Science Tower
164 51 Kista, Sweden
alireza.behrooz@appearnetworks.com
[2] Ericsson Research, Färögatan 6
164 80 Stockholm, Sweden
alisa.devlic@ericsson.com

Abstract. This paper introduces a Context-aware Privacy Policy Language (CPPL) that enables mobile users to control who can access their context information, at what detail, and in which situation by specifying their context-aware privacy rules. *Context-aware* privacy rules map a set of privacy rules to one or more user's situations, in which these rules are valid. Each time a user's situation changes, a list of valid rules is updated, leaving only a *subset* of the specified rules to be evaluated by a privacy framework upon arrival of a context query. In the existing *context-dependent* privacy policy languages a user's context is used as an additional condition parameter in a privacy rule, thus *all* the specified privacy rules have to be evaluated when a request to access a user's context arrives. Keeping the number of rules that need to be evaluated small is important because evaluation of a large number of privacy rules can potentially increase the response time to a context query. CPPL also enables rules to be defined based on a user's social relationship with a context requestor, which reduces the number of rules that need to be defined by a user and that consequently need to be evaluated by a privacy mechanism. This paper shows that when compared to the existing *context-dependent* privacy policy languages, this number of rules (that are encoded using CPPL) decreases with an increasing number of user-defined situations and requestors that are represented by a small number of social relationship groups.

Keywords: Context-aware privacy rules, social relationships, mobile users.

1 Introduction

Advances in context-aware technologies are making peoples' lives easier by sensing and collecting information from their surroundings and using this context to assist people in performing their daily tasks. In most scenarios, preserving privacy and integrity of users' personal data is a major issue. People would like to control who can access their context information, at what level of detail, when, and in which

R. Prasad et al. (Eds.): MOBISEC 2011, LNICST 94, pp. 25–39, 2012.

situations. Therefore, it is important that people can grant restricted access to their context information or deny access to it, depending on their current situation. There is a need to specify a user's *context-aware* privacy preferences, enabling a user to define different sets of privacy rules and when they are applicable. A user's situation is defined by a set of context values that are obtained through some automated means (i.e., via sensors). Note that we distinguish between context values whose access is controlled by a privacy mechanism (i.e., ***sensitive context***) and context values which are used to determine a user's situation (i.e., ***situational context***).

We tried to express context-aware privacy policies using eXtensible Access Control Markup Language (XACML) [1] in our earlier work [2]. However, for each privacy rule in XACML, a logical combination of conditions can be specified that determines whether the rule should be applied or not. Therefore, we specified a user's situational context in the condition part of the privacy rule along with the authorized requestors' condition and the logical function that should be applied to all condition parts in the rule when the condition is evaluated. Since the logical function can be an AND operation, for a condition to be valid *all* the condition parts have to be evaluated to TRUE. Since at the time when a user situation changes the requestor information is not necessarily available, the requestor condition part cannot be evaluated. Therefore, privacy rules cannot be filtered upon a situational context change. Consequently, upon receiving a context request, *all* the specified privacy rules must be evaluated, *regardless* of the fact that only some of these rules might be valid in the user's current situation. We refer to the described privacy policy languages, in which a user's situational context is used as an additional condition based upon which the decision for granting or denying access to the requested context is made, as *context-dependent* privacy policy languages.

A user's social relation with the requestor has been identified in several studies [3][4] as one of the most important factors influencing a person's willingness to disclose their context information. However, in most of the existing privacy policy languages, social relationship is not used to define privacy preferences. Hence, the ability to define privacy preferences based on a user's social relationships with potential requestors can reduce the number of privacy rules that need to be specified by a user, since potentially a large number of requesters can be represented by a social relation. Consequently, fewer rules need to be evaluated by a privacy mechanism.

In order to address these problems, we propose a *context-aware* privacy policy language (CPPL) based upon the following two design considerations: (1) a user's situations are defined separately from their privacy rules and (2) a context requestor can be specified using its identity or its social relationship to a user. The CPPL design assumes a privacy mechanism that will, when a user's situation changes, select the privacy rules that are valid in this situation. Since a user's situation is defined by a set of values assigned to context parameters, a context change does not necessarily imply a situation change. In fact, we assume that a user's situation will change at least an order of magnitude less frequently than each of its context parameters. Consequently, the selection of a set of privacy rules will be in frequent (as compared to the rate of context changes). When a context request arrives, the privacy mechanism will check (based on a context requestor) only the rules that are valid in the current situation that have been updated at the latest situation change.

We describe architecture of the privacy framework that provides the described context-aware privacy mechanism. Additionally, we create an analytical model to compute the reduction in the number of rules that have to be evaluated by the privacy mechanism when they are specified using CPPL as opposed to when they would be specified using other context-dependent policy languages. We show that this reduction linearly increases with the number of situations and requestors that are represented by a few social relationship groups.

The rest of this paper is organized in 6 sections. Section 2 describes our motivation scenarios, leading to requirements for a *context-aware* privacy policy language. Section 3 reviews the related works according to the identified requirements. Section 4 describes the syntax of the proposed privacy language. Section 5 describes the privacy framework architecture. Section 6 provides the analytical model, while section 7 concludes the paper with plans for future work.

2 Motivation Scenarios and Requirements

This section presents two scenarios that motivate the need for context-aware privacy in the daily lives of average mobile users and derives requirements for a context-aware privacy language from these scenarios.

Bob uses a mobile application that collects his body temperature, heart rate, and blood pressure. A *Healthcare Institution (HCI)* collects *all* these health information of the application users. When a health emergency occurs (i.e., one or more health indicators exceeds a predefined threshold), the application will detect it, then it will send this information along with the Bob's current location to the HCI. HCI will in turn identify the nearest available nurse to Bob and ask her to visit Bob.

Alice uses a mobile application to share her activities with her family and friends on specific occasions. She agrees to allow her husband to see her current activity while she is in Paris, but not otherwise. When Alice is on vacation (this context information can be inferred from her current location and calendar), she wants to inform her friends about the city she is visiting. She is also willing to share her location at the street level with her friends on weekends so they can find each other and go out together.

From these scenarios we identified the requirements that should be considered when selecting an existing or designing a new context-aware privacy policy language:

- *User-defined situation:* A privacy policy language should enable users to define privacy preferences that are valid in specific situations. A situation should be specified by users based on their available context (via a tool with a graphical user interface), using parameters defined in the context model.
- *Rich context model:* A context model should be rich enough to allow users to define any situation, while at the same time it should be customized for use by applications in a particular domain.
- *Periodic-time:* A privacy policy language should enable the definition of privacy rules that are valid during periodic time intervals (e.g., on weekends, on work days from 12:00 to 13:00, etc.).

- *Social relationship:* Users should be able to define their privacy rules based on the social relationship that they have with other people. Maintaining social contacts with a small number of groups and using these groups to specify privacy rules can significantly reduce the number of rules that users need to specify.
- *Fine-grained access:* Users should be able to specify the granularity of context information that they want to disclose to others in a privacy rule.
- *Context-awareness:* Privacy policy rules should be context-aware, thus they should be evaluated when a user's situational context changes (rather than when a request for sensitive information arrives). As a result, only a subset of privacy rules that are valid in the user's current situation will be checked by the privacy mechanism when a context request arrives.
- *Conflict-handling***:** There is a potential risk that more than one privacy rule is valid and can be applied in a particular situation. In some cases, these rules can indicate different actions. For example, one rule might grant access to the requested context, while another rule denies it. A privacy policy language must provide a mechanism to handle such conflicts.

3 Related Work

This section reviews the state-of-the-art context-dependent privacy languages according to the requirements identified in the previous section.

3.1 Houdini

Houdini [5] is a context-aware privacy framework that enables users to specify their privacy preferences through web-based forms. Privacy preferences can be defined based on the users' current situation, their social relationship with the requestor or the requestor's identity, and the relation of the requestor's current situation with respect to their own current situation (e.g., if they are located on the same street).

Users can define potential situations through web-based forms. However, situations cannot be defined or modified using the privacy policy language, since the privacy language is decoupled from the context model. Instead, a user's situation is a single variable that is used in the privacy policy rules and whose value must be calculated before evaluating the corresponding rule.

Conflict handling is supported by assigning priorities to rules. If there is a conflict between different rules actions, the rule with the highest priority will be considered.

There is no support in privacy policies for periodic time conditions. Additionally, granularity of disclosed context information cannot be controlled.

3.2 UbiCOSM

The Ubiquitous Context-based Security Middleware (UbiCOSM) [6] represents privacy policies as tuples of one or more contexts that are associated with a set of privacy permissions. A privacy permission determines what kind of operation can or

cannot be performed on a particular resource. Privacy permissions are not directly assigned to particular users. Instead, when a user enters a particular context (e.g., a physical location), the associated permission becomes applicable to this user. Permissions have a property that can be assigned either a positive or negative value indicating that access to the requested data is granted or denied, respectively. Additionally, it is not possible to control the granularity of disclosed information.

The UbiCOSM middleware allows mobile users to define situations based on their context and map their privacy permissions to these situations. It updates the set of valid permissions whenever the user's situation changes, which decreases the policy evaluation time when a context request arrives.

There is no explicit support in UbiCOSM for defining a user's situation based on a periodic time interval. Additionally, conflict handling is not supported. A user's social relationship cannot be used (rather than the requestor's identity) to define privacy rules.

3.3 CoPS

The Context Privacy Service (CoPS) [7] enables mobile users to control who can access their context data, when, and at what granularity. CoPS does not enable specification of a user's situations and rules based on a user's context. Instead, CoPS uses an optimistic or pessimistic approach to define a default policy in which all requests are granted or denied, except those that match one of the rules specified by a policy maker. By defining only the rules that specify under which circumstances context should be disclosed or not (depending on the chosen default policy) the number of rules that need to be specified and evaluated is reduced.

Access to context can be granted for the restricted time (e.g., only once, for 2 hours, always allow, or never allow). A context model in CoPS is limited to context variables provided by the middleware. A hierarchical syntax (e.g., "campus.building.room") can be used by a user to control the granularity of the disclosed context information. Context granularity can be specified using a spatial precision (e.g., "Room 123"), temporal restriction, or freshness of the disclosed context information (e.g., to disclose the user's location 15 minutes ago).

CoPS implements definition of groups and access control based on the membership in the specified groups, which decreases the effort of specifying and evaluating the policy rules. Groups can reflect an organization structure or can be defined by a user.

If more than one rule matches the request, conflict handling is performed using the CoPS specificity algorithm that identifies the most specific rule from the matching rules set by comparing their structure fields in the specified order of priority.

3.4 Context Privacy Engine

The Context Privacy Engine (CPE) [8] extends the traditional Access Control List (ACL) mechanism with a set of context constraints that have to be evaluated to validate a particular privacy policy. Context constraints are used to define context conditions that are associated to either the context owner or the context requestor,

using XQuery expressions. However, a user's situation defined in the policy is not reusable in other rules. Therefore, if more than one policy should use a particular situation, then this situation definition must be repeated in each of these policies.

A subject and a requestor in CPE policies can be individuals or groups of people. However, a group (e.g., defining a user's social relationships) is expected to be created by an application.

CPE supports conflict handling by considering a policy level, which is an optional field in the policy. A policy at a higher level overrides all policies at lower levels. If there are multiple policies with the same level, the most specific one will be applied.

The CPE policy language does not provide a means to control the granularity of disclosed context information. This policy language is *context-dependant*, thus when a context request arrives, *all* the privacy policies have to be evaluated *regardless* of the user's current situation. Additionally, evaluating *context-dependent* privacy policies requires retrieving context data upon arrival of a context query, which can be time consuming.

3.5 SenTry

The SenTry language [9] is designed as a combination of a user-centric privacy ontology (called SeT Ontology, written in Web Ontology Language (OWL) [10]) and the Semantic Web Rules Language (SWRL) [11] predicates. For each context entity a policy instance is defined which contains the associated privacy rules (defined as SWRL predicates). The SenTry language supports two categories of rules: Positive Authorization Rules (PAR) and Negative Authorization Rules (NAR). NAR rules can only have a "deny" effect, while PAR rules can either allow access to the requested context or transform the context information according to the specified granularity.

Privacy rules are *context-dependent*, thus the context-awareness requirement is not met. Situations can be defined using the SWRL predicates, however SWRL does not support more complex logical combinations of predicates than the conjunction, which makes is difficult to define arbitrarily complex situations.

SenTry provides the "grant override" combination algorithm to handle conflicts among different rules, which is an optimistic algorithm that grants access if at least one rule grants access to the requested context information. Different combination algorithms can be defined to handle conflicts, but it is up to the privacy framework to decide what algorithm should be applied for *all* the policies in the system.

3.6 Summary

Table 1 shows that none of the existing privacy languages fully meets the identified requirements. Most of these languages enable definition of situations and support a rich context model, but none of them enables definition of situations based on periodic time intervals. Languages that satisfy the context-awareness and/or the social relationship requirement enable a small number of privacy rules that have to be evaluated by a privacy mechanism upon a context query arrival.

Table 1. Summary and comparison of context-dependent privacy policy languages

	Houdini	UbiCOSM	CoPS	CPE	SenTry
User-defined situation	+/-	+	-	+/-	+/-
Rich context model	+/-	+/-	-	+	+/-
Periodic-time	-	-	-	-	-
Social relationship	-	-	+/-	+	-
Fine-grained access	-	-	+	-	+
Context-awareness	-	+	-	-	-
Conflict-handling	+	-	+	+	+/-

4 CPPL Model

This section introduces a novel CPPL language that enables mobile users to define their context-aware privacy preferences in a specific granularity based upon the social relationship of a user with a context requestor. By defining different parts of the language, we explain how CPPL meets all the identified requirements.

CPPL specifies *context-aware* privacy rules by mapping a set of *privacy rules* to one or more user *situations*, in which these privacy rules are valid (as shown in Figure 1). A CPPL policy contains one or more context-aware privacy rules.

```
<xs:complexType name="ContextPrivacyRuleType">
  <xs:sequence>
    <xs:element minOccurs="0" ref="cppl:Description" />
    <xs:element type="cppl:Situations" />
    <xs:element type="cppl:RuleSet" />
  </xs:sequence>
  <xs:attribute name="contextPrivacyRuleId" type="xs:ID" />
</xs:complexType>
```

Fig. 1. XML schema representation of a Context Privacy Rule

The *Situations* element contains one or more *Situation* elements each of which is defined by one or more context conditions that must apply to *an* entity (as depicted in Figure 2). To determine if an entity is in a particular situation, all the conditions in the *Conds* element have to evaluate to true. The *Entity* element represents a context owner, which can be an *environment*, a *device*, or a *user*. For example "*<Entity >user|Bob</Entity>*" refers to Bob as a user. The entities in CPPL are expressed using the MUSIC context model [12]. The MUSIC context model enables context parameters to be specified in a hierarchical manner by noting all the parent concepts in the inheritance chain to which the context parameter is assigned to (e.g., "#healthInfo.bloodPressure.systolic"). CPPL uses entities and context parameters to define situations, thus meeting the rich context model requirement.

```
<Situation situationId="Emergency">
  <Entity>#user|Bob</Entity><Conds><CondOp op="OR"><Cond>
    <Logical op="OR"><Constraint param="#healthInfo.bloodPressure.systolic"
                                        op="NEQ" value="105" delta="15.0"/>
                    <Constraint param="#healthInfo.bloodPressure.distolic"
                                        op="NEQ" value="70" delta="10.0"/>
    </Logical></Cond><Cond><Constraint  param="#healthInfo.heartRate"
                                        op="NEQ" value="75" delta="25.0"/>
    </Cond></CondOp></Conds></Situation>
```

Fig. 2. A Situation element representing Bob's health emergency

Each time a user's situation changes, the list of valid privacy rules are updated; it is this set of rules that will be checked upon receiving a context query. This design makes the CPPL context-aware as was elaborated in section 2.

The *Conds* element contains either a single condition (*Cond*) or a condition operator (*CondOp*) that performs a logical operation on two or more single conditions. Operators provided in the current version of CPPL are logical *AND* and *OR*. The *Cond* element can be defined as a single constraint or a logical combination of constraints. In the example in Figure 2, an "OR" logical combination of the abnormal blood pressure and abnormal heart rate is defined to indicate an emergency situation. The former is a logical combination of two constraints while the latter is a single constraint. Note that two kinds of operators are used in this example. One is applied to constraints or conditions, while another operator is applied to context parameters to define a constraint.

The *Constraint* element (illustrated in Figure 2) is used to specify the set of context parameters that define a condition. It specifies five attributes:

- *entity:* An optional attribute that is used to specify an entity to which the context parameters belong to. If it is not specified, the default entity of parent situation element will be used.
- *param:* It refers to the context parameter that can be assigned a value (e.g., "#location.civilAddress.city").
- *op:* The operator applied to one or two context parameters for constraint verification. Table 2 shows the constraint operators that are supported in CPPL. The definition of these operators is adopted from [13].
- *value:* The value of the context parameter.
- *delta:* This attribute is used for continuous parameters. It shows the acceptable range of context parameters values for a given constraint.

Table 2. Constraint operators

GT	Greater than	NGT	Not greater than	STW	Starts with
LT	Lower than	NLT	Not lower than	ENW	Ends with
EQ	Equals	NEQ	Not equals	NSTW	Not starts with
CONT	Contains	NCONT	Not contains	NENW	Not ends with

The CPPL time constraint element is used to specify any (periodic) time constraint. It contains either a *DateRange* element or an *Interval* element. The former defines a time range that *begins* and *ends* at the specified date and time. The latter specifies an interval that has the following attributes:

- *daysOfWeek*: denotes week days in the form of numbers or words, representing one or more days or a range of days (e.g., notations "mon,wed,fri" or "1-3" can both be used to represent Monday, Wednesday, and Friday).
- *months*: denotes months of the year by their names or numbers. Months can enumerated or represented by a range (e.g., "2,4,7" or "may-aug").
- *daysOfMonth*: numbers between 1 and 31 that indicate days in a month.

The *Interval* element contains an optional "*TimeRange*" element for specifying time periods with "*startTime*" and "*endTime*" attributes. An example of using the *TimeConstraint* element to represent working hours is shown in Figure 3.

```
<TimeConstraint ><Interval daysOfWeek="MON-FRI">
   <TimeRange startTime="08:00:00" endTime="16:00:00" />
</Interval></TimeConstraint>
```

Fig. 3. An example of the use of the *TimeConstraint* element

Figure 4 shows the definition of *Ruleset* element, a collection of *Rule* elements that are mapped to one or more situations. The *Rule* element (see an example in Figure 7) describes who (*Identity* element) can access what kind of context information (*ContextParams* element). The effect of a privacy rule is to *permit* or *deny* access to the requested context information. When there is more than one rule in a rule set, a combination algorithm should be used to evaluate the final effect and resolve potential conflicts of the RuleSet. This algorithm uses a "denyOverrides" or "permitOverrides" policy to *deny* or *permit* access to the requested context if at least one rule from the set evaluates to *deny* or *permit*, respectively. This algorithm can be used to determine the final effect of the multiple rule sets in a user's privacy policy.

```
<xs:complexType name="RuleSet">
   <xs:attribute name="combinationAlg" type="xs:string" use="optional" />
   <xs:sequence><xs:element type="cppl:RuleType" /></xs:sequence>
</xs:complexType>
<xs:complexType name="RuleType"><xs:sequence>
    <xs:element type="cppl:IdentityType" /> <xs:element type="cppl:ContextParamsType" />
   </xs:sequence>
   <xs:attribute name="effect" type="cppl:EffectType" />
</xs:complexType>
```

Fig. 4. XML schema representation of Ruleset element

The *ContextParams* element specifies sensitive context parameters assigned to an entity. Using an *AnyContextParam* element within the *ContextParams* element indicates that all context parameters can be accessed by all potential requestors.

Privacy rules can be defined based on a user's social relationship with a context requestor, enabling a user to specify a rule for a *class* of requestors instead for each requestor individually. The *Identity* element enables different ways of representing a context requestor in a privacy rule, using the following elements:

- *One:* an individual represented by an id.
  - e.g. <one id="sip:admin@HCI.com"/>
- *Many:* a group of users in the same administrative domain.
  - e.g. <Many domain="HCI.org"/>
- *Relation:* a group of people having a specific relation to the context owner.
  - e.g. <Relation relation="spouseOf"/>
- *AnyIdentity:* is used for rules that should be applied to all the requestors.

Figure 5 shows an example of *RuleSet* element that permits Alice's friend to access her current location at the city level.

```
<Rule effect="Permit">
   <Identity><Relation relation="friendOf"/></Identity>
    <ContextParams><ContextParam>
     <Entity>#user|Alice</Entity>
     <Param>#location.civilAddress.city</Param>
    </ContextParam></ContextParams>
</Rule>
```

Fig. 5. Privacy rule allowing Alice's friends to access her location

When a group of people are selected using the *Many* or *Relation* elements, it is possible to exclude one ore more individuals from the selected group using the *Except* element (as depicted in Figure 6).

```
<Identity><Many domain="HCI.org">
   <Except id="sip:Alice@example.com"/>
</Many></Identity>
```

Fig. 6. Using Except element to exclude an individual from a group

5 Privacy Policy Management System Architecture

A *Privacy Policy Management System (PPMS)* provides a context-aware privacy mechanism supporting the CPPL language that enables a user to control access to his/her sensitive context depending on this user's current situation. We refer to the user controlling access to his/her sensitive context information as the *context owner* and to the user requesting access to the context owner's sensitive context as the

context requestor. Note that we assume that a context owner and a context employ different devices to control and request access to a particular context, respectively.

As illustrated in Figure 7, the PPMS executes at the context owner's device and consists of four components: *Policy Administration Point (PAP), Policy Refinement Point (PRP), Policy Enforcement Point (PEP), and Policy Decision Point (PDP).* A context owner specifies his/her privacy preferences (step 1). These preferences are transformed by the PAP into the CPPL policies and stored into the *CPPL Repository* (step 2). The PRP reads the privacy rules from the policies stored in the CPPL repository (step 3), extracts the *situational context parameters* that are used in conditions of these rules, and sends the request to the *Context Provider* to retrieve these context parameters (step 4). The *Context Provider* obtains the requested context from the sensors that can provide this information and fires the context changed event containing the sensed context data. This event is received by all the components that have previously requested this information (steps 5 and 11).

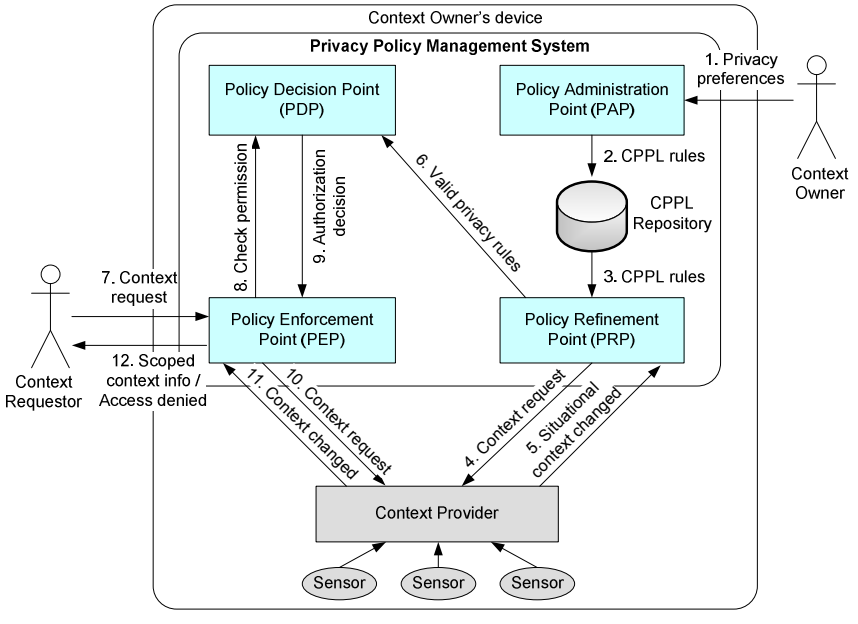

Fig. 7. The overall architecture of privacy policy management system

The PEP controls access to the context owner's sensitive context. After receiving a context request from the context requestor (step 7), the PEP asks the PDP to check if this requestor is permitted to access the requested context (step 8). The PDP checks the list of valid rules in the owner's current situation in order to determine is the requestor is permitted to access the requested *sensitive context*. It returns the authorization decision to the PEP, which is either to *permit* or to *deny* access to the requested context (step 9). If the request is permitted, the PEP will forward the context request to the context provider (step 10). Otherwise, the context request will

be rejected. After obtaining the context information from sensors, the PEP will receive the updated context (step 11), format this context in the granted scope (if the access to it has been permitted), and send this scoped value to the requestor (step 12).

If the updated context sent in the changed context event is the situational context, it will be received by the PRP (step 5). The PRP will determine if this situational context update has caused the change of a user's situation and if so it will update the list of valid privacy rules in this new situation & send this list to the PDP (step 6). This process is called *policy refinement* and must be performed before determining if the context requestors are authorized to access the requested context. However, if the changed context is not a situational context, the policy refinement is not necessary because the valid rules are already up to date.

The PPMS has been implemented in Java and integrated as a component in the MUSIC context middleware [14]. We plan, as part of the future work, to evaluate the performance of the proposed privacy mechanism in terms of the latency that this mechanism adds to the context response time.

6 Analytical Model

As earlier elaborated, a small number of privacy rules encoded using a context-aware privacy policy language should be evaluated by a privacy mechanism upon arrival of a context query. To fulfill this goal, CPPL introduces two approaches: (1) filtering the context-aware privacy rules based on the context owner's current situation and (2) representing context requestors using their social relationship with the context owner.

This section describes an analytical model that is used to compute and compare the number of privacy rules that need to be evaluated by a privacy mechanism when they are specified using a context-dependent privacy policy language vs. when they are specified using the CPPL.

In order to simplify the model we made the following three assumptions:

- Each privacy rule specifies access of a single requestor (i.e., an individual or a group of people) to a single context parameter.
- For any requestor and the owner's sensitive context parameter, a separate privacy rule has to be defined.
- The number of the sensitive context parameters whose access is controlled is the same in each situation.

In *context-dependent* privacy policy languages, the number of privacy rules that need to be evaluated by a privacy mechanism (N_{CB}) is equal to the number of requestors used in the privacy rules (R) multiplied by the sum of the number of sensitive context parameters (C_{Si}) in all the user's defined situations (S):

$$N_{CB} = \sum_{i=1}^{S} R * C_{S_i} = R * \sum_{i=1}^{S} C_{S_i} \quad (1)$$

Since the number of sensitive context parameters in different situations is the same (i.e., $\forall i, j \in [1,S], S \in N, C_{S_i} = C_{Sj} = C_S$) (1) becomes:

$$N_{CB} = R * S * C_S \tag{2}$$

In CPPL, the number of privacy rules that need to be evaluated by the privacy mechanism (N_{CPPL}) is equal to the number of valid privacy rules in a user's current situation. Since the privacy rules in CPPL can be specified for both the individual requestors and the groups, N_{CPPL} can be calculated as the sum of the number of groups (G) and number of individual requestors that are identified by their identity (R_{IND}) multiplied with C_S:

$$N_{CPPL} = (G + R_{IND}) * C_S \tag{3}$$

In order to compare the number of rules that need to be evaluated by a privacy mechanism when they are encoded using a context-dependent privacy language vs. when they are encoded using CPPL, we compute the ratio of N_{CB} and N_{CPPL}:

$$\frac{N_{CB}}{N_{CPPL}} = \frac{R * S}{G + R_{IND}} \tag{4}$$

The result of (4) can be interpreted as the reduction in the number of the rules achieved by CPPL design of context-aware privacy rules. Since each potential requestor either belongs to a group or is considered as an individual requestor, and assuming that there are no empty groups, the sum of groups and individual requestors is *less than or equal to* the number of all requestors. Furthermore, there is always at least one situation even if a user has not defined any situation, to which the privacy rules are mapped (i.e., an *"Always"* situation). Thus, it can be concluded that N_{CPPL} will always be *lower than or equal to* N_{CB}. Additionally, the more user-defined situations and the larger the number of requestors that are represented by social relationship groups, the more effective CPPL becomes compared to the existing context-dependent privacy policy languages. If all the rules are defined for individual requestors and there is only one situation, the number of rules in CPPL will be equal to the number of rules in context-dependent privacy languages.

7 Conclusion

This paper introduces a context-aware privacy policy language (CPPL), which can be used to represent context-aware privacy preferences of mobile users in order to control access to their context information. A user's context is used in context-aware privacy rules to specify which of these rules are valid in particular situation(s). CPPL enables a user to define situations using a set of context parameters that are defined in a context model. When a user's current situation changes, a list of valid rules is updated by a privacy mechanism (that implements support for CPPL), thus leaving only relevant rules to be evaluated upon arrival of a context query.

CPPL enables a user to specify privacy rules based on a social relationship with a context requestor, thus reducing the number of rules that need to be specified by the user and that consequently need to be evaluated by the privacy mechanism.

In the existing context-dependant privacy policy languages a user's context is used as an additional condition in a rule, along with a requestor's identity. In order to

process a context request, the privacy mechanisms supporting these languages need to process *all* the specified rules and to retrieve all context values that are used to define privacy preferences, which can in the case of evaluation of a large number of rules, significantly increase the privacy policy evaluation time.

We provide an analytical model that calculates a reduction in the number of rules that need to be evaluated by a privacy mechanism when they are encoded using CPPL (vs. when they are compared to context-dependent privacy languages), showing that effectiveness of CPPL increases with an increasing number of defined situations and requestors that are represented by a small number of social relationship groups.

As part of the future work, we plan to perform a performance evaluation of the proposed privacy framework in terms of the latency it adds to the context response time. Moreover, a usability study will be done to make sure that our approach can be easily employed by average mobile users.

References

[1] Moses, T.: eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS (February 2005)

[2] Devlic, A., et al.: Context inference of users' social relationships and distributed policy management. In: Proc. of the 7th IEEE International Conference on Pervasive Computing and Communication (PerCom 2009), 6th Workshop on Context Modeling and Reasoning (CoMoRea 2009), Galveston, Texas, USA, pp. 755–762 (March 2009)

[3] Consolvo, S., et al.: Location Disclosure to Social Relations: Why, When, and What People Want to Share. In: 11th International Conference on Human-Computer Interaction (CHI 2005), pp. 81–90. ACM Press, Las Vegas (2005)

[4] Olson, J.S., et al.: Preferences for Privacy Sharing: Results & Directions CREW Technical Report (2004)

[5] Hull, R., et al.: Enabling context aware and privacy-conscious user data sharing. In: 5th IEEE International Conference on Mobile Data Management (MDM 2004), Berkley, CA, USA, pp. 187–198 (January 2004)

[6] Corradi, A., Montanari, R., Tibaldi, D.: Context-based Access Control Management in Ubiquitous Environments. In: Third IEEE International Symposium on Network Computing and Applications (NCA 2004), Cambridge, MA, USA, pp. 253–260 (August 2004)

[7] Sacramento, V., Endler, M., Nascimento, F.N.: A Privacy Service for Context-aware Mobile Computing. In: First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005), Athens, Greece, pp. 182–193 (September 2005)

[8] Blount, M., Davis, J., et al.: Privacy Engine for Context-Aware Enterprise Application Services. In: IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Shanghai, China, vol. 2, pp. 94–100 (December 2008)

[9] Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, pp. 181–184 (2001)

[10] McGuinness, D.L., Harmelen, F.: OWL web ontology language overview. W3C submission, W3C Recommendation (2003), `http://www.w3.org/TR/owl-features/`

[11] Horrocks, I., et al.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C submission, `http://www.w3.org/Submission/SWRL/`

[12] Reichle, R., Wagner, M., Khan, M.U., Geihs, K., Lorenzo, J., Valla, M., Fra, C., Paspallis, N., Papadopoulos, G.A.: A Comprehensive Context Modeling Framework for Pervasive Computing Systems. In: Meier, R., Terzis, S. (eds.) DAIS 2008. LNCS, vol. 5053, pp. 281–295. Springer, Heidelberg (2008)

[13] Reichle, R., et al.: A Context Query Language for Pervasive Computing Environments. In: Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008), Hong Kong (March 2008)

[14] IST project MUSIC, Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environment project, http://www.ist-music.eu