PlexC: A Policy Language for Exposure Control

Yann Le Gall and Adam J. Lee
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA, USA
{ylegall, adamlee}@cs.pitt.edu

Apu Kapadia
School of Informatics and Computing
Indiana University Bloomington
Bloomington, IN, USA
kapadia@indiana.edu

ABSTRACT

With the widespread use of online social networks and mobile devices, it is not uncommon for people to continuously broadcast contextual information such as their current location or activity. These technologies present both new opportunities for social engagement and new risks to privacy, and traditional static 'write once' disclosure policies are not well suited for controlling aggregate *exposure* risks in the current technological landscape.

Therefore, we present *PlexC*, a new policy language designed for exposure control. We take advantage of several recent user studies to identify a set of language requirements and features, providing the expressive power to accommodate information sharing in dynamic environments. In our evaluation we show that *PlexC* can concisely express common policy idioms drawn from survey responses, in addition to more complex information sharing scenarios.

Categories and Subject Descriptors

H.4.0 [**Information Systems Applications**]: General

Keywords

Privacy, Exposure, Policy Languages

1. INTRODUCTION

The popularity of online social networks has contributed to an unprecedented amount of personal information sharing. Moreover, the widespread use of mobile devices encourages the broadcast of *contextual* information from any location. For example, smart phone users can send their current location to social networks such as Facebook Places [25], Google+ [14], and Foursquare [13]. Furthermore, technologies such as CenceMe [19] can infer the current activity (e.g., running or dancing) from a smart phone's onboard sensors. With so many ways to share personal contextual information, the task of protecting individual privacy is becoming more challenging. One important challenge is to maintain the utility of information sharing without sacrificing personal privacy. To achieve this equilibrium individuals need to do more than simply define a static disclosure policy once and for all. They must be able to specify flexible and adaptive policies that can manage the disclosure of personal information in the face of both typical and atypical access patterns. We refer to such policies as *exposure-aware*.

Motivation. Over the years, a large body of research literature has explored a variety of access control mechanisms and their policy language encodings. Existing policy languages have incorporated powerful features to group principals into functional roles [16, 17, 27], delegate authorization decisions across security domains [2, 4], and even manage state changes during policy evaluation [3, 21]. However, few sharing systems or policy languages have drawn upon large user studies to inform their design. As a consequence, the resulting languages and systems offer a variety of interesting features, yet may not provide users with the functionality needed to address their real-world exposure concerns. By contrast, we carefully consider findings from several recent user studies within the exposure space [5, 24, 28] and leverage a variety of findings from these studies to provide insight into exposure perception and control.

For example, Schlegel et al. highlighted the importance of exposure feedback through an intuitive interface [28]. Additionally, Patil et al. discovered that certain factors, like the frequency with which location requests occur, are more important to users than other common factors, like the current time of the location request [24]. This is quite interesting, as few existing systems allow for controlling the frequency of requests, while several [18, 26, 30] provide policy constructs for controlling disclosures based upon the day of week or time of day. Another important outcome of this study was the identification of several common concerns and policy idioms that are not typically associated with social engagement purposes, such as only sharing location during emergencies or with law enforcement personnel.

Our Contributions. Findings from these recent user studies reveal several ways to address shortcomings in current information sharing systems and their respective policy languages. To summarize a few, location sharing systems and their disclosure policy languages must be flexible enough to support users with diverse privacy concerns [5]. Furthermore, it is important to provide unobtrusive, ambient feedback about how users' data are being shared without necessarily revealing the identity of the requester [28]. Finally, policies should provide the ability to manage disclo-

SACMAT'12, June 20–22, 2012, Newark, New Jersey, USA.

sure based on more than just common factors, such as the identity of the requester, but more complex policies may not be easily expressed [24]. To address these concerns we propose a new policy language *PlexC* whose functionality is based in large part on the needs voiced by the human subjects who participated in these studies. In doing so we make the following contributions:

1. We survey the recent research literature for human subjects' data regarding contextual information sharing and exposure control. Based on these findings we develop policy language and system requirements necessary for servicing the exposure control needs of users;
2. We develop a general system model for contextual information sharing systems that represents the features of existing logically centralized systems and is capable of modeling more user-centric systems that may appear;
3. We design a novel policy language, *PlexC*, that addresses the limitations identified in recent user studies and specify its syntax and semantics. We further discuss the query resolution procedure used by *PlexC*;
4. To evaluate the utility of *PlexC* we demonstrate that it is both capable of expressing a range of common policy idioms and can encode interesting real-world information sharing constraints specified by the subjects of several survey studies.

Paper outline. We start by defining the exposure problem and by identifying a set of language requirements that are motivated by recent user studies in §2. Next we discuss related work in §3. The syntax of *PlexC* is described in §4. In §5 we evaluate the expressivity of our policy language against real user policies, interpret the findings, and discuss future work in §6. Finally, we conclude in §7.

2. BACKGROUND AND REQUIREMENTS

In this section, we first define the concept of *exposure* and introduce the relevant research challenges. Next we highlight the results of several recent user studies that explore aspects of the exposure-control problem space. We conclude this section by enumerating a set of exposure-control policy language features the need for which is highlighted by these studies and other works in the research literature.

2.1 The exposure problem

Before describing our system model and how it addresses the exposure problem, we first explain what we mean by "exposure". Intuitively, a user's ideal policy for controlling access to their personal data is a moving target that is, at best, approximated by the policies and controls that the user puts in place to protect their information. To paraphrase an example by Schlegel et al. [28], a user may initially set a policy allowing her co-workers to access her location during normal work hours to facilitate in-person meetings. However, if she later finds that her boss is accessing her location every 5 minutes to ensure that she remains in-office, she may become uncomfortable. This disconnect between the employee's model of permissible sharing and the level of sharing allowed by the protections that she put in place leave her more *exposed* to external queries and analysis than she had anticipated.

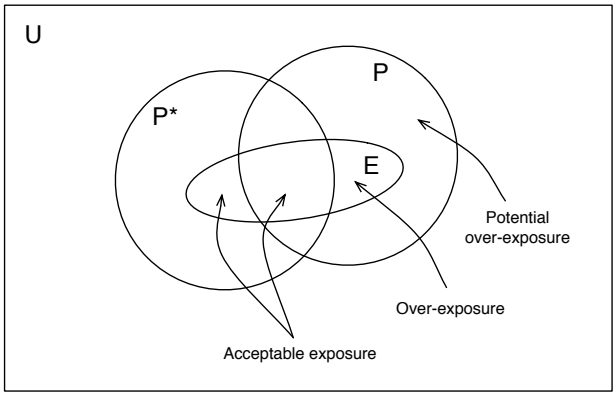

Figure 1: The exposure control problem.

The problem of exposure control is non-trivial, as exposure can be viewed as a function over a multi-dimensional space expressing a human sentiment. Some of these inputs may be unknown a priori as many contextual factors may influence a user's perception of exposure. For example, someone's notion of exposure may be influenced by the time of day, their current location, whom they are with, how many requests they have received, and so on. We propose that trying to control such a complex and dynamic property requires an adaptive process in which disclosure policies are continuously specified, enforced, and revised.

A semi-formal view of the exposure problem is captured in Figure 1. In this depiction U represents the universe of all access traces to a user's personal data. These traces describe sequences of queries to a user's data, which may be dependent on system and user context (location, activity, etc.), as well as other past queries. $P^\star$ represents the user's ideal model of data sharing—which is unlikely to be captured correctly due to the complexities of managing the myriad contextual facets of the exposure control problem—while P represents the access traces permitted by the user's deployed policies. E represents the access traces that have actually been made to the user's data and represents the user's *exposure*. $E \cap P^\star$ represents the user's *acceptable exposure*, whereas $E \cap P \setminus P^\star$ represents the user's *over-exposure*. $P \setminus P^\star \setminus E$ represents the user's *potential over-exposure*.

The complexities of properly capturing $P^\star$ lead us to envision an *exposure control loop* in which a policy is specified and deployed, and feedback on the allowed access traces is periodically provided to the user. This exposure feedback can then be used by the user to revise their policy over time, resulting in a sequence of policies allowing trace sets $P_1, P_2, \ldots, P_n$ that aims to minimize potential over exposure and avoid further over-exposure. In the following sections we describe the design of the *PlexC* system and how it accommodates and encourages the exposure control loop.

2.2 Recent Studies

Our work is grounded in a series of recent user studies relating to exposure control. Here we describe each study in more detail and explain how their findings are relevant to exposure control. The results of these studies contribute to many of the requirements discussed in Section 2.3.

When Privacy and Utility are in Harmony: Towards

Better Design of Presence Technologies. In this study, Biehl et al. explored user sentiment about presence data collection and sharing with an emphasis on workplace settings [5]. Another goal was to explore the utility of receiving various types of presence information. They conducted a survey of 32 participants representing a wide range of ages, professions, and geographic regions across the US.

This study is relevant to exposure control because it measured how comfortable participants felt as a function of many variables, including the type of data collected (e.g., location, activity), the recipient of the data (e.g., boss, coworker, friend), the setting in which the data collection occurred (e.g., office, work event, home), and the format, owner, and location of the data collected. Also, the authors measured how comfort levels changed based on perceived utility for the recipient.

One important finding of this study was that comfort levels across different sensing technologies were bimodal. In other words there is no one-size-fits-all privacy policy that addressed all users' privacy needs. Thus, information sharing systems must be flexible. Another finding was the strong correlation between how comfortable users were with sharing information by granularity and their perceived utility of receiving information from others at that granularity.

Eyeing Your Exposure: Quantifying and Controlling Information Sharing for Improved Privacy. In this study, Schlegel et al. address the problem of location exposure feedback and control in a game-based simulated lab study [28]. They develop and compare two different smart-phone interfaces: (i) a so-called "detailed information interface" that shows the number of requests in the last hour from different categories of people (e.g., friends, family, strangers), and (ii) a so-called "eyes interface" that shows the user a number of cartoon eyes on the main screen of the app in which each eye represents location requests from a single person, and the size of the eye grows depending on the number of requests and the social relationship.

This study is relevant because it quantifies the role of frequency in exposure control and embraces the notion that informative *feedback* is an important part of controlling exposure. However, with too much detail feedback interferes with querier anonymity. Likewise, if feedback is too frequent or obtrusive, then it may annoy the user. The findings of this study suggest that it is possible to benefit from feedback without sacrificing querier anonymity or usability.

My Privacy Policy: Exploring End-User Specification of Free-Form Location Access Rules. In this online study Patil et al. asked over a hundred participants to write location-sharing policy rules using everyday English [24]. In addition participants were asked to rate and rank the importance of a number of factors that might influence location sharing, such as the identity of the recipient, the current location, the frequency of requests, and the like.

The research questions addressed in this work are also very pertinent to the domain of exposure control. The notion of exposure varies across individuals and across many other dimensions. This study measured the preferences of a large sample of individuals and allowed them to freely identify factors that contribute to over-exposure. Furthermore, the ratings and rankings suggest which factors might require more attention than others.

There were a number of interesting findings. Unexpectedly, participants indicated that the 'time of day' and the 'day of the week' of location requests were less important than the 'frequency of receiving requests'. Furthermore, in general people had difficulty expressing coherent policies that controlled for all of the factors that they rated as being important. Finally, the authors identified several common themes in the free-form policies. Some of these include complete manual mediation of requests, temporary blocking, and sharing only for emergencies.

2.3 Language Requirements

Here we describe a series of language features that are important for the efficient and accurate expression of exposure control policies. These requirements are motivated by past work as well as the recent user studies previously described.

Disclosure Negotiation. In open distributed systems it is impossible to specify the trust relationship between all pairs of individuals a priori. Negotiation allows strangers to build trust by exchanging credentials, information, etc. Negotiation has been identified as an important feature and used in several policy languages [4,12,29]. Negotiation is also important for understanding the reasons for which a request was made, and individuals are more likely to feel comfortable sharing their location if they believe it will be useful to the requester [5]. An example of this type of policy idiom is given below: *Share my city-level location with anonymous requesters, but if the requester is willing to reveal his identity then also share my street-level location.*

Polymorphism. Here we use polymorphism to describe a policy whose requirements change based on the user's degree of over-exposure. Schlegel et al. explore ways of estimating and representing this metric [28]. The following policy illustrates the utility of exposure polymorphism: *Share my exact location with family only when my current over-exposure level is low; Otherwise if my over-exposure is high, only share my city-level location.*

Side Effects. Side effects appear in policy rules and specify transactions that modify the authorization state of the system when the rule is satisfied. Side effects are appropriate in large dynamic systems where maintaining ACLs is inefficient [21]. Furthermore, role-based policy languages typically require updates to the authorization state as users activate roles. However, most modern authorization languages do not explicitly provide constructs to express state changes, so management of state changes must be hard-coded into system resource guards [3]. An example of a policy that would be more easily expressed with side-effects might be: *The first 3 location requests from an individual require my explicit approval, but subsequent requests do not.*

Aggregate Operations. Aggregate operators can provide users with summary information about the set of accesses to their personal information (the region labeled "E" in Figure 1). This often includes operations such as SUM, COUNT, or MAX, over sets and multisets of tuples [20]. Frequency-based policies rely on the ability to aggregate records in the audit log, and Patil et al. showed that individuals believe that the frequency of requests is an important factor to consider [24]. Furthermore, it is demonstrated by Dell'Armi et al. that aggregate operators can increase the modeling power of disjunctive logic programming languages and provide concise knowledge representation [11]. A typical use

of this feature would be the use of aggregation to limit the frequency of location disclosures, e.g.: *Do not share my location more than 10 times per day.*

Querier Privacy. "Querier privacy" often refers to anonymous access of resources, but, in general, it is not limited to protecting the identity of the requester. Querier privacy has been identified as an important feature in large online social networks (OSNs), especially those that have been used to organize protests and share sensitive documents [1]. Interestingly, Tsai et al. showed that users of the location-sharing technology *Locyoution* felt more comfortable sharing their location when they were given feedback about who requested their location. The ability to provide users with exposure feedback might seem to be incompatible with querier anonymity, but Schlegel et al. demonstrated intuitive feedback interfaces that accomplish this [28].

Delegation. Delegation allows disclosure decisions to be passed on to a trusted authority. The utility of this feature is apparent in the following policy rule: *Share my location with the same people with whom my friends share their location.* Delegation is an important feature of many authorization languages [2, 4, 10, 12, 16, 17]. In large decentralized systems preexisting trust relationships often do not exist between authorizer and requester. Thus, delegation allows the authorizer to make decisions based on trusted third parties. Delegation also simplifies policies in hierarchical systems.

Groups and Roles. In role-based access control (RBAC), subjects are assigned to one or more roles (or groups), and permissions are assigned based on their roles, e.g.: *Family members can always see my exact location, but colleagues can view my location only during work hours.* RBAC greatly simplifies permissions management, is well suited for large organizations in the commercial and government sectors [27], and is supported in most modern policy languages [5, 24].

Time and Location-based Rules. Time-based rules control disclosure based on the current time. Similarly, location-based rules control disclosure based on the current location of the policy owner or the requester. As an example of a location-based rule, the owner might define a number of 'named regions' as a coordinate pair and a radius, and associate regions with a sharing policy: *Share my location with family only if I am at the hospital.*

Time intervals and named regions are natural ways to specify policies that accommodate daily schedules and routines, and previous work has demonstrated that users of OSN's are comfortable expressing policies using these features [30]. Furthermore the current time and location of an individual influence the type of information that she is willing to share [5], e.g.: *Share my location only between 9am and 5pm.* As previously shown, policy rules that are based on the frequency of requests can also be implemented by combining features that allow access to the current time and audit log. Patil et al. observed that frequency-based rules may have a greater importance than time-based rules [24].

Disclosure Levels. In a policy language that supports multiple disclosure levels, the policy owner can specify the degree of information to disclose. For example, in response to a location request, the policy owner might choose to disclose only the name of the current city. This feature would accommodate many of the challenges identified by Biehl et al. [5]. They found people were more comfortable sharing detailed location information at work and less detailed information outside of work. Therefore, comfort with different disclosure levels is highly influenced by current location.

3. RELATED WORK

The body of literature describing access control policy languages and policy idioms is extensive. Researchers have developed role-based abstractions [17, 31] to simplify the management of user rights, trust management approaches [6, 7] that combine the management of policies and trust relationships, and distributed logic-based approaches [4, 12] that can concisely and compactly manage very complex policies. While some of these approaches have a logical syntax and semantics, others are based on XML [31] or object-oriented paradigms [10]. In this section we survey several recent and feature-rich policy schemes and illustrate that none of these schemes supports the full set of features outlined in Section 2.3 (Table 1 summarizes the features of these schemes).

Li et al. introduced the *RT* framework, which consists of a family of related languages for specifying distributed authorization policies [17]: *RT* is a role-based trust management language in which policies are constructed using four simple rule types that assign users to roles, represent delegations, and structure roles into hierarchical relationships; RT_1 extends this basic framework with support for parameterized roles; RT^T provides syntax for specifying policies that require thresholding and separation of duty; and RT^D introduces constructs for constrained delegation. *RT* has both a set-based semantics and a Datalog-based semantics, and policies can be efficiently evaluated via translation into a Datalog program.

Park and Sandhu introduced $UCON_{ABC}$, a family of models for usage control (UCON) [22, 23]. UCON is a conceptual framework that provides a comprehensive approach to managing access control, Digital Rights Management (DRM), and trust management. It can express a wide variety of policies by applying different combinations of authorizations, obligations, and conditions to digital objects. For example, basic RBAC can be expressed using authorization rules alone, whereas DRM can be expressed using a combination of authorization rules, conditions, and obligations. UCON also explores the complexities that arise when data consumers become data producers, if, for example, a client's personal information is logged during transactions.

Damianou et al. introduced Ponder, an access control language for a variety of applications such as firewalls, operating systems, and databases [10]. In addition to traditional features such as roles and delegation, Ponder supports policies that require actions to be taken after being triggered by a certain event. Unlike many other authorization languages, Ponder is described as a declarative, strongly typed, object-oriented language.

DeTreville presents Binder, a security language for distributed systems, which is based on Datalog [12]. However, unlike basic Datalog, Binder programs can securely communicate with other Binder programs across distributed environments using signed certificates.

Becker et al. developed Cassandra, which is built upon Datalog with constraints (Datalog_C [4]). Cassandra provides role-based trust management in distributed domains with credential retrieval, separation of duty, and role activation/deactivation. Additionally, Cassandra rules may contain a constraint c drawn from a constraint domain C that

	negotiation	exposure polymorphism	side effects	aggregation	tunable querier privacy	roles & delegation	time rules	location rules*	tunable disclosure granularity
RT	no	no	no	no	no	yes	no	no	yes
Cassandra	yes	no	no	yes	no	yes	yes	no	yes
SecPal	no	no	no	yes	no	yes	yes	no	yes
SMP	no	yes	yes	no	no	yes	no	no	no
Ponder	no	no	yes	no	no	yes	yes	no	yes
Binder	no	no	no	no	no	yes	no	no	yes
UCON	yes	no	yes	no	no	roles	yes	yes	no
PlexC	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**

Table 1: Comparison of language features.
*While other languages were not designed with location sharing in mind, they may be able to support location via minor extensions.

can be tuned to provide different tradeoffs between computational complexity and expressivity. Becker et al. then build upon the extensibility of Cassandra in SecPal [2]. SecPal has a high-level natural syntax and its design features include delegation, constraints, and negation in queries. SecPal policies can also be compiled into Datalog$_C$ programs.

The State Modifying Policies framework [3] can be used to extend policy languages based on distributed logics with concepts from Transactional Datalog [8]. This provides support for the use of policies that are capable of adding/retracting facts to/from the policy's logic program at runtime. This is useful, e.g., for supporting policies that can augment and examine their own audit logs.

Recently, Gunter et al. described an idiom called "Experience-Based Access Control" (EBAM) [15]. Briefly, EBAM is a set of models, tools, and techniques to reconcile the differences between ideal policies and the operational policies enforced by the underlying system. A typical approach for realizing EBAM may include maintaining and analyzing an access log to suggest ways to update and improve existing rules. This iterative process is similar to the exposure feedback loop discussed previously; however, *PlexC* focuses on the human perception of exposure.

Collectively, these policy languages introduce an impressive assortment of paradigms, idioms, and features for expressing security policies in different contexts. However, no single policy language provides support for all of the features identified in Section 2.3 as being important to the management of end-user exposure (see Table 1). This is not surprising, as exposure management was not a primary goal during the development of this prior work. In the next section we describe *PlexC*, a policy language for exposure control that was designed not only to meet all of the needs identified in Section 2.3, but also to take into account the findings of recent user studies in the domain of exposure control.

4. PLEXC: SYSTEM AND SYNTAX

We now describe the system model assumed by *PlexC* including its components, interfaces, and assumptions. We then develop the *PlexC* exposure control language, which is based on transactional extensions to Datalog.

4.1 System Model

PlexC is a system that acts as a protection layer around a set of resources. It has four main components: an external query interface, an evaluation engine, a set of local knowledge bases, and a component that manages sending feedback to the user. This high-level structure is shown in Figure 2.

In a typical workflow, an application requests access to some resource, say, Alice's location. This request goes

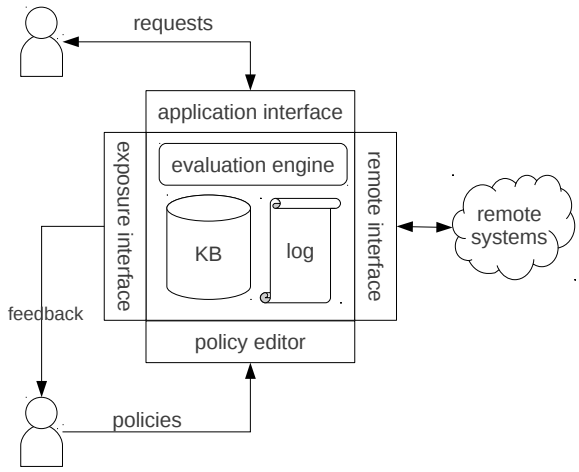

Figure 2: The general *PlexC* system model.

through the external query interface, which limits and controls ways in which external applications can interact with the system. The query is then processed by a policy evaluation engine, which determines if the requester should be granted access to the resource. This decision might be based on information from several sources. In addition to evaluating disclosure policies, the evaluation engine may examine audit logs and the authorization state stored in local knowledge bases, and it may even request information from remote *PlexC* systems. Finally, the disclosure decision is written to the system audit log, and the feedback component may decide to notify Alice about this interaction.

Communication between knowledge bases. When the resource engine evaluates Alice's disclosure policies, communication with other system components may be necessary. For example, Alice may choose to reveal her location only if very few requests have been made in the past hour. This policy rule would require that the evaluation engine communicate with the local database, which maintains a log of transactions. Furthermore, Alice's disclosure policies might depend upon Bob's policies. For instance, Alice might only reveal her location to members of a volunteer group organized by Bob. Consequently, the policy engine would communicate with Bob's knowledge base to verify that the requester is a member of the appropriate group. In order to enable this type of behavior, knowledge bases can communicate with each other via an *exported interface* that allows users to reference each other's disclosure rules.

Storage of disclosure policies. Users store their exposure control policies in a knowledge base composed of facts and rules. Notice that the system model shown in Figure 2 caters to the principles of the exposure control loop by explicitly providing a path by which feedback information can flow from the system to the policy owner. Furthermore, this general system design lays the groundwork to support a rich set of features that are important for controlling exposure. We discuss these features in the next section.

4.2 Datalog Overview

Datalog is a logic programming language for deductive databases. Because of its well-defined declarative semantics and efficient query evaluation algorithms, it provides a nice environment within which to express authorization policies. Indeed, many policy languages—including *PlexC*—are based on Datalog or can be translated into Datalog programs (e.g., [2–4, 12, 17]). We now provide a brief review of the terminology, syntax, and semantics of Datalog.

Datalog is a syntactic subset of Prolog, and programs are composed of *facts* and *rules*. A *rule* is a statement of the form $q :- p_1, p_2, ..., p_n$, where q and each p_i for $1 \leq i \leq n$ are *literals*. Intuitively, this rule can be read as "p_1 and p_2 and ... and p_n imply q". q is referred to as the *head* of the rule, and the *body* is composed of each p_i. A *fact* is a rule that contains only a head and no body.

A *literal* has the form $P(x_1, x_2, ..., x_m)$ where P is a predicate name followed by a tuple with arity m, and each x_i for $1 \leq i \leq m$ is a *variable* or *constant*.

Consider the following example that demonstrates a simple Datalog program:

```
parent('alice','bob').
parent('carol','alice').
ancestor(?X,?Y) :- parent(?X,?Y).
ancestor(?X,?Z) :- ancestor(?X,?Y),ancestor(?Y,?Z).
?-ancestor(?X,'bob').
```

Here we denote variables as strings prefaced by a question mark, "?", and string literals are surrounded by quotes. In this example, the first two statements are ground facts. These are sometimes stored in a physically separate database called the *Extensional Database* or EDB. The next two statements are rules, which are stored in the *Intensional Database* or IDB. The EDB and IDB contain disjoint sets of predicates; as such, predicates defined in the EDB may only appear in the body of rules, and may not appear in the head of any rule. The last statement above is a query that seeks to find all bindings of `?X` such that `?X` is an ancestor of `'bob'`. In the above program, the tuples `ancestor('alice','bob')` and `ancestor('carol','bob')` satisfy this query.

4.3 PlexC

We now describe the set of extensions distinguishing *PlexC* from pure Datalog. First, we discuss the interface across which external applications communicate with our system. We then explain how transactional updates to the system state can be expressed and how policy authors can both create rules that are based on changes in the system audit log and rules that are sensitive to user feedback. Finally, we list additional built-in predicates and functions.

External Interface. External applications, such as location- or presence-sharing applications, communicate with the *PlexC* system to determine if a certain resource of a user should be disclosed to the requester. This communication occurs through an *external interface* exposed to these types of applications that is composed of a set of predicates described below:

System defined:.

- `get(?Q,?U,?R,?L) :- U.canAccess(?Q,?R,?L)` $\otimes$ `+log(?Q,?U,?R,?L,NOW);` `get()` is a system predicate that is invoked when user Q requests resource R at granularity L belonging to user U. Successful evaluation of this predicate inserts a record into the audit log.
- `over-exposure(?U,?E)` binds the current exposure of user U to the free variable E. The current exposure could be estimated by a user-defined function that runs over the audit log. For example, a user's exposure might be HIGH if the audit log shows a large number of accesses in the past hour.

User defined:.

- `canAccess(?Q,?R [, ?L])` is true if the requester, Q, can access the desired resource, R, at the (optional) level of granularity, L.
- `isMember(?U,?R)` is true if user U is currently a member of role R.
- `canQuery(?U,?P)` is true if the policy author allows another user U to reference predicate P in her policy rules.

By creating a set of policy rules, the policy author is free to define the conditions satisfying the predicates in the external interface. However, the system is responsible for defining the `get` predicate, which retrieves the user's personal information, such as the current location or activity.

When personal information is disclosed via the `get` predicate, a transaction occurs in which a record of the access is inserted into the system audit log. The record contains information about the requester, the resource disclosed, the time of disclosure, and the level of detail (granularity) of the disclosure. Prior research has explored incorporating transactions into Datalog. Transaction Datalog (TD) is an extension to Datalog for executing transactions that modify the database as rules are evaluated. TD supports the classical ACID properties as well as other properties like transaction hierarchies, concurrency, and cooperation [8].

PlexC also supports the notion of *state effects* as introduced by Becker and Nanz [3]. Effects can be composed by the sequential transaction operator "$\otimes$" from Transactional Datalog [8]. This feature allows users to express policies that require role activation, separation of duty, or other state-dependent operations. For example, the following rule allows requesters to access an individual's location only once:

Example 4.1

```
canAccess(?X,LOCATION) :- not seen(?X,LOCATION)
⊗ +seen(?X,LOCATION);
```

Here, the sequential transaction operator, $\otimes$, is used to specify the facts to be inserted to or deleted from the database (denoted by a "+" or "−" respectively), if all conditions in the body are satisfied.

Built-in Constants. *PlexC* also includes a number of constants that refer to resources that the user is not responsible for defining:

- NOW refers to the current time;
- TODAY is a string representing the current date;
- MYLOC represents the current location of the user, stored as a point (coordinate pair), and a radius;
- LOCATION is a constant used to identify location resources;
- CITY is a constant used to specify the city-level of granularity for location resources;
- ANYONE is a placeholder that matches any user when scanning the audit log;
- TRUE represents the positive truth value;
- FALSE represents the negative truth value.

User Policies. Users can define facts and rules to control disclosure. As with basic Datalog this allows the easy creation of groups and roles. Example 4.2 demonstrates a set of basic facts and rules that a user might create.

Example 4.2
```
canAccess('bob', LOCATION);
canAccess(?X, LOCATION) :- isMember(?X,
'friend');
```

The first statement is a fact that explicitly gives Bob access to Alice's location information. In the next statement, Alice also allows her friends to view her location. Thus we see that with basic Datalog syntax we can easily implement a simple, static role-based access control model.

Remote predicates. Users can also specify *remote predicates* by providing an identifier as a prefix before the predicate name. With this feature we can encode policies that require delegation. In the following example, Alice delegates disclosure decisions to Bob:

Example 4.3
```
canAccess(?X, LOCATION) :- bob.canAccess(?X,
LOCATION);
```

Similarly, with these features we can express basic forms of disclosure negotiation and other rules that are *quid pro quo*. In the following example, Alice only allows another user to access her location if she can access his location:

Example 4.4
```
canAccess(?X, LOCATION) :- ?X.canAccess('alice',
LOCATION);
canAccess(?X, LOCATION, CITY) :-
?X.canAccess('alice', LOCATION, CITY);
```

Additionally, *PlexC* allows policy authors to constrain the information in the knowledge base that is visible to other users. This is achieved with the built-in predicate, `canQuery(?U,?P)`, which allows another user U to query the predicate, P. For example, the registrar at a university might allow a teacher, T, to query the list of students enrolled in courses that he teaches.

Example 4.5
```
canQuery(?T, enrolled) :- teaches(?T, ?C),
enrolled(?U, ?C);
```

Handling Exposure. Users have the ability to write policies that depend on the current exposure conditions. Aggregation is an important prerequisite for achieving this behavior [28], and Mumick et al. investigate extending Datalog with aggregate operators [20]. They show that Datalog can be efficiently extended with aggregate operators using magic sets and semi-naive evaluation algorithms, which provide good heuristics over the naive, bottom-up approach. In order to ensure the termination of Datalog programs, aggregate operators are subject to restrictions such as stratification [11]. In our case we provide special built-in predicates that are restricted to aggregating over a logically separated set of facts and predicates. This restriction is sufficient for our purposes, as it allows *PlexC* policies to aggregate over the audit log, for example. The following demonstrates how aggregation over the audit log can be used to limit the frequency of location sharing to no more than 5 times per day:

Example 4.6
```
canAccess(?X,LOCATION) :- accessCount(?X, ?N, 1,
00:00, 23:59), ?N <= 5;
```

Here, `accessCount(?X, ?N, ?D, ?`T_1`, ?`T_2`)` invokes a search of the audit log for the number of accesses N by requester X between times T_1 and T_2 over the past D days.

In addition to aggregation over the audit log, users can write rules that depend on their current exposure. Prior research suggests that several factors contribute highly to an individual's notion of exposure, such as the social relation of the requester [28], the frequency of requests [24], and the surroundings at the time of request [5]. *PlexC* provides the access to this information through built-in functions, predicates, and language features, making it easy to define custom exposure functions. For example, a user might define exposure levels to be HIGH if the number of requests by strangers in the audit log exceeds a certain threshold.

Example 4.7
```
canAccess(?X,LOCATION) :- exposure(?E), ?E <
MEDIUM;
canAccess(?X,LOCATION,CITY) :- exposure(?E), ?E
>= MEDIUM;
```

Keeping the User in the Loop. In the study by Patil et al. many participants expressed the desire to mediate all requests for their personal information [24]. To this end we introduce a built-in function `prompt(?X,?R)` that prompts the current user to give requester X permission to access resource R. Other participants simply wanted to be notified for each request, so we define a similar function `notify(?X,?R)`, which notifies the user that requester X has accessed resource R.

Additional Features. There have been a number of extensions to pure Datalog, some of which *PlexC* incorporates. These include built-in predicates, functions, and negation [9]. *PlexC* includes support for basic equality, comparison, and arithmetic operators. These can be viewed as infix predicates except that the operands correspond to terms, and the result of the atom is evaluated by the underlying implementation and does not depend on facts in the local knowledgebase. The following rule demonstrates both a built-in function to test if the current day is a weekday, as well as the built-in greater-than operator, and stipulates that location requests are only permitted on weekdays between 9am–5pm:

Example 4.8
```
canAccess(?X, LOCATION) :- weekday(TODAY), NOW >
9:00, NOW < 17:00;
```

```
program    : statement*
statement  : (fact | rule | query) ';'
rule       : literal ':-' literals ['⊗' effects]
fact       : atom
query      : '?' literals
literals   : atom (',' atom)*
literal    : 'not'? atom
effects    : effect (',' effect)*
effect     : ('+'|'-') atom
atom       : [identifier '.'] identifier tuple
tuple      : '(' terms ')'
terms      : term (',' term)*
term       : function | variable | constant |
             string | number
variable   : '?' constant
function   : identifier tuple
```

Figure 3: The formal grammar of *PlexC*.

PlexC also supports several predicates to create *named regions*, which are essentially locations on a map with an associated radius. The `region(?NAME,?LAT,?LON,?R)` predicate defines a region $NAME$ centered at the coordinate (LAT,LON) with radius R. The predicate `inRegion(?L,?NAME)` tests if the location L is within the region, $NAME$. Example 4.9 only allows members of a *student* group to access location when the user is on campus:

Example 4.9
```
region('campus', 40.2, -100.2, 1km);
canAccess(?X,LOCATION) :- inRegion(MYLOC,
'campus'), member(?X, 'student');
```

Furthermore, *PlexC* supports a limited form of negation. Pure Datalog does not allow negation, which can threaten the evaluation safety of programs. Typically, negation is handled using *stratification* or the *closed world assumption* (CWA). Stratification imposes an evaluation order on rules where negated body predicates must be evaluated before predicates in the rule head. CWA allows the inference of negative ground facts if they do not appear in the EDB [9]. *PlexC* uses the CWA to handle negation.

Example 4.10 demonstrates a rule that uses negation to implement an exclusion policy:

Example 4.10
```
canAccess(?X,LOCATION):- not
member(?X,'enemies');
```

Finally, we support a set of built-in functional-symbols that may depend on the deployment environment. For example, a location-sharing application might contain a set of functional-symbols to perform distance calculations, e.g. `within(?L1,?L2,?D)` would return true if $L1$ and $L2$ are within distance D. Similarly, functions that provide reverse geocoding would be useful, such as `cityOf(?L)`, which would return the city of the location coordinate L.

Figure 3 shows the grammar production rules for *PlexC*.

4.4 Rule Evaluation

In typical Datalog systems extensional facts are applied to rules in the intensional database to generate new facts until no new facts can be generated (a fixed point). This bottom-up approach is straightforward and can occur before handling queries. However, more expressive languages do not take this approach to evaluate queries. One of the reasons is that certain special predicates and function symbols cannot be computed prior to receiving queries. For example, some predicates and constants depend on current time and location (e.g., `accessCount`, many types of rules defining the `canAccess` relation), while others require the user's interaction (e.g., `prompt`).

Therefore, instead of using bottom-up strategies, modern expressive policy languages employ memoized, top-down evaluation algorithms that combine the efficiency of goal-oriented approaches while avoiding the non-termination issues of standard SLD resolution used in Prolog [4].

Consider the following example:

Example 4.11
```
canAccess(?X, LOCATION) :- weekday(TODAY),
   bob.member(?X,'friend'),
   accessCount(?X,?N,1,00:00,23:59), ?N <= 5
```

Access to the current user's location is contingent upon several factors. First, the date is obtained and tested as a parameter of `weekday`. Next, the second literal is a remote predicate indicating that the requester needs to be a friend of Bob. A query is therefore sent to Bob's exported predicates interface, and if Bob allows the current user to query this predicate, and the requester belongs to the `friend` role, then a positive result is returned. The `accessCount` predicate invokes a query on the audit log and binds the number of accesses by the requester (in the current day) to the free variable N, and the last item tests that N is no more than 5.

5. EVALUATION

In this section we give an informal evaluation of the expressiveness of *PlexC*. First, we show how *PlexC* meets all of the language requirements outlined in Section 2.3. We then show how a variety of common policy idioms can be represented in *PlexC*. Finally, we use *PlexC* to encode some of the more interesting policies gathered from free-response questions in the study conducted in [24].

5.1 Meeting All Language Requirements

Here we show how *PlexC* achieves all of the requirements outlined in Section 2.3.

- **Groups and Roles:** Example 4.2 demonstrates how to define roles and limit access based on group membership. Policy authors can create different roles and assign membership relations using the natural Datalog syntax.
- **Delegation:** Example 4.3 shows how delegation is possible in *PlexC*. Delegation requires the evaluation of a relation whose records are not contained in the local knowledge base.
- **Disclosure Negotiation:** Example 4.4 relies upon the evaluation of remote predicates to exchange information between the policy author and the requester until the conditions for disclosure are satisfied.
- **Side Effects:** Example 4.1 shows how *PlexC* draws upon existing syntax [3, 8] to specify changes to the authorization state during the evaluation of rules.

- **Disclosure Levels:** The amount of detail in a disclosure can be controlled by specifying the appropriate resource identifier. For instance, Example 4.7 shows how the granularity of information to be disclosed can be adjusted.
- **Time and Location rules:** Time-based rules can be expressed using the built-in constants that represent the current time and day as shown in Example 4.8 and Example 4.9, respectively.
- **Aggregate Operations:** *PlexC* provides support for aggregation over the system audit log via special predicates. This functionality can be seen in Example 4.6.
- **Polymorphism:** In Example 4.7 we encode a policy whose behavior is dependent on the target user's current level of over-exposure.

5.2 Encoding Free-form Policies

Here we demonstrate the expressiveness and utility of *PlexC* by encoding some interesting and complex policies taken from participant free responses gathered during the study detailed by Patil et al. [24].

A number of participants expressed the desire for complete mediation of all requests for their location. For example, one such policy was: *Keep my location private and ask every time someone wants to know my location.* This policy would have the following implementation in *PlexC*:

Example 5.1
```
canAccess(?X,LOCATION) :- prompt(?X,LOCATION);
```

Another common response was that users wanted to be notified after every location disclosure, but didn't necessarily need to know who had accessed their location. For example, one user stated *"Any time anyone views my location, I must get a notification."* This rule is similar to the previous example, but does not require direct interaction from the user. One interpretation of this policy is that access should be allowed to everyone as long as there is a notification:

Example 5.2
```
canAccess(?X,LOCATION) :- TRUE ⊗
notify(?X,LOCATION);
```

Another interpretation might be to modify any existing rules by adding a notification upon success, in which case the `notify()` predicate can be appended to the existing rules.

Location-based rules were also among the more unique responses. For example, one participant wrote, *"Allow users to see when I am within a particular radius of them."* An interesting implication of this is that the user wants to share her location only when it could possibly be of use to the recipient. This would be implemented in the following way:

Example 5.3
```
canAccess(?X, LOCATION) :- get(?X, LOCATION,
?L1), within(MYLOC, ?L1, 1km);
```

Many participants also indicated that they would not want to share their location for social engagement purposes. However, many of these otherwise unwilling users of location-based services indicated that they *would* share their location during emergency situations, e.g.: *I would only want someone to know my physical location in an emergency situation.* This introduces the difficult problem of determining when the user is experiencing an emergency. However, one response provided some intuition: *"only if I'm missing for 24+ hours".* This policy could be approximated in the following way:

Example 5.4
```
canAccess(?X,LOCATION) :- member(?X,'emergency'),
accessCount(ANYONE, ?N, 1, NOW-24:00, NOW), ?N
= 0;
```

By this encoding access is granted only if the requester belongs to the 'emergency' role and the audit log shows that nobody has received the user's location in the past 24 hours.

6. DISCUSSION AND FUTURE WORK

Implementation. We are currently implementing *PlexC* as part of a larger location-sharing application. We have already implemented several major components of the system, including a mobile application to track current location and view the locations of others, a web interface for managing policies and carrying out more complex queries, and a server application to store data and manage social relations. *PlexC* will be used as the fundamental access control component to manage the information flow between the other system components. We plan to use this testbed to better understand the utility of *PlexC*, as well as to explore the system design tradeoffs present in this space.

Validation of User Studies. The design of *PlexC* is motivated by the collection of recent user studies described in Section 2.2. However, these studies share a common limitation: because these studies are based on user surveys, they reveal only the *perceived* preferences and needs of participants in an artificial environment. In other words, although the participants in these studies are likely to have given truthful answers to the surveys, there is a chance that they would behave differently in a real-world scenario. While it is not possible to completely account for all sources of response bias in a lab setting, a field study of a fully functional system would be able to mitigate these effects and support or challenge the findings upon which *PlexC* is based.

Usability of Policy Creation. *PlexC* allows users to create concise policies for exposure management, and it inherits many desirable traits from Datalog (e.g., unambiguous semantics and tractable evaluation). However, we do not expect the average user to write rules in *PlexC* directly. *PlexC* was developed, instead, to represent a formal semantics for exposure-aware policies. While it is possible to write *PlexC* policies directly, we envision that most users will interact with their policies via some form of structured policy editor. We believe that a form-based rule editor would simplify the creation of *PlexC* rules that are easy for users to understand, while still taking advantage of the power of *PlexC*. Striking a good balance between usability and expressive power will be an interesting research challenge.

7. SUMMARY

In this paper we address the development of *PlexC*: a policy language for exposure control. The concept of *exposure* denotes the extent to which an individual's personal data is shared, and addresses the individual's resulting concern for privacy. Given the complexity of this design space, we first articulate requirements for policy languages for exposure control by analyzing the findings of

several recent survey studies addressing various facets of the exposure problem. Not surprisingly, existing *access control* policy languages are shown to be insufficient for meeting the *exposure control* needs voiced by participants in these studies. We present *PlexC* as a solution to meet the needs of these participants. After describing the details of *PlexC*, we show that it is both suitable for meeting the needs of users in modern context-sharing systems, as well as capable of encoding a variety of historically useful policy idioms. Although *PlexC* was derived by examining surveys of users' perceived exposure-control needs, further evaluation work is still required. In particular our team plans to explore the development of usable policy-management interfaces and user studies of *PlexC*-based contextual sharing systems.

Acknowledgements. This research is supported by NSF grants CCF-0916015, CNS-1016603, & CNS-1017229, and US DHS grant 2006-CS-001-000001, under the auspices of the Institute for Information Infrastructure Protection (I3P). The contents of this paper do not necessarily reflect the views of the sponsors.

8. REFERENCES

[1] M. Backes, M. Maffei, and K. Pecina. A security api for distributed social networks. In *NDSS*, Feb. 2011.

[2] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 2009.

[3] M. Y. Becker and S. Nanz. A Logic for State-Modifying Authorization Policies. *ACM TISSEC*, 13:20:1–20:28, July 2010.

[4] M. Y. Becker and P. Sewell. Cassandra: Distributed Access Control Policies with Tunable Expressiveness. In *POLICY*, pages 159–168, June 2004.

[5] J. T. Biehl, E. Rieffel, and A. J. Lee. When Privacy and Utility are in Harmony: Towards Better Design of Presence Technologies. *Personal Ubiquitous Computing*, in press, Feb. 2012.

[6] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. In *Infrastructures (Position Paper). LNCS 1550*, pages 59–63, 1998.

[7] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, May 1996.

[8] A. J. Bonner. Transaction Datalog: a Compositional Language for Transaction Programming. In *In Proceedings of the International Workshop on Database Programming Languages*, 1997.

[9] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE TKDE*, 1:146–166, March 1989.

[10] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *POLICY*, pages 18–38, 2001.

[11] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate functions in disjunctive logic programming: semantics, complexity, and implementation in dlv. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 847–852, 2003.

[12] J. DeTreville. Binder, a logic-based security language. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 105–113, May 2002.

[13] Foursquare. `http://www.foursquare.com/`.

[14] Google+. `https://plus.google.com/`.

[15] C. A. Gunter, D. M. Liebovitz, and B. Malin. Experience-based access management: A life-cycle framework for identity and access management systems. *IEEE Security & Privacy Magazine*, 9(5), September/October 2011.

[16] A. J. Lee, T. Yu, and Y. L. Gall. Effective trust management through a hybrid logical and relational approach. In *ASIACCS*, Apr. 2010.

[17] N. Li and J. C. Mitchell. RT: A role-based trust-management framework. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–212, Apr. 2003.

[18] Locaccino. `http://locaccino.org/`.

[19] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In *SenSys*, pages 337–350, 2008.

[20] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.

[21] L. E. Olson, C. A. Gunter, and P. Madhusudan. A formal framework for reflective database access control policies. In *CCS*, pages 289–298, 2008.

[22] J. Park and R. Sandhu. Towards usage control models: beyond traditional access control. In *SACMAT*, pages 57–64, 2002.

[23] J. Park and R. Sandhu. The uconabc usage control model. *ACM TISSEC*, 7(1):128–174, Feb. 2004.

[24] S. Patil, Y. L. Gall, A. J. Lee, and A. Kapadia. My Privacy Policy: Exploring End-user Specification of Freeform Location Access Rules. In *Proceedings of the Workshop on Usable Security (USEC)*, Mar. 2012.

[25] Facebook places. `http://www.facebook.com/places/`.

[26] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao. Understanding and Capturing People's Privacy Policies in a Mobile Social Networking Application. *Personal and Ubiquitous Computing*, 13:401–412, August 2009.

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

[28] R. Schlegel, A. Kapadia, and A. J. Lee. Eyeing your Exposure: Quantifying and Controlling Information Sharing for Improved Privacy. In *SOUPS*, July 2011.

[29] K. E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation. In *POLICY*, pages 68–79, 2002.

[30] J. Y. Tsai, P. Kelley, P. Drielsma, L. F. Cranor, J. Hong, and N. Sadeh. Who's viewed you?: the impact of feedback in a mobile location-sharing application. In *ACM CHI*, pages 2003–2012, 2009.

[31] W. Yao, K. Moody, and J. Bacon. A Model of OASIS Role-Based Access Control and its Support for Active Security. In *SACMAT*, pages 171–181, 2001.