Policy-based Access Control in Mobile Social Ecosystems

Sara Hachem, Alessandra Toninelli, Animesh Pathak, and Valérie Issarny
{sara.hachem, alessandra.toninelli, animesh.pathak, valerie.issarny}@inria.fr
INRIA Paris-Rocquencourt, France

***Abstract*—The novel scenarios enabled by emerging mobile social applications raise serious concerns regarding access control of users' contextual and social data. Given the variety of existing and upcoming social applications, it is important to provide (i) generic yet flexible policy models that combine expressivity with personalization, (ii) actual running infrastructures to enforce policy-based access control on heterogenous devices with minimal development/deployment effort, and (iii) user-interfaces to allow the easy specification of policies without dealing with the complexity of the underlying policy and data models. Toward this goal, in this paper we make three contributions. First, we present a novel policy framework for controlling access to social data in mobile applications. The framework allows the representation of expressive policies based on users' social interactions, which can be easily extended with new domain data models, while keeping policy model compatibility intact. Secondly, we demonstrate how we integrated the policy framework as part of Yarta, a middleware for managing mobile users' social ecosystems, implemented and deployed on laptops and smart phones. Third, we show the graphical policy editor provided with the policy framework to allow non-technology savvy users to easily specify and manage their access control policies.**

I. Introduction

The popularity of social applications has been steadily increasing on the Web over the past few years. Advances in wireless network technologies and the widespread diffusion of smart phones equipped with sensing capabilities offer promising chances to enhance social applications and make them truly pervasive in everyday life. In addition, the formation of ad hoc networks enables social encounters between proximate users with common interests, anywhere and anytime [1], [2].

The novel scenarios enabled by these emerging *mobile social applications*, however, raise serious concerns regarding privacy and access control of users' data. The most critical aspect is that mobile social applications manage contextual data, such as personal contents, user interests and activities, as well as human relationships, which are sensitive per se and can be further used to infer sensitive information. It is therefore crucial to ensure an adequate level of control on information describing users' social environments and interactions.

Given the variety of existing and upcoming social applications, it is important to design a policy framework to be as generic and flexible as possible, so that several different applications can make use of it. This raises several requirements: on one hand, it calls for (i) generic yet flexible policy models that combine expressivity with personalization. On the other hand, it raises the need for (ii) actual running infrastructures to enforce policy-based access control on heterogenous devices with a minimal development and deployment effort, as well as (iii) user-interfaces to allow the easy specification of policies without dealing with the complexity of the underlying policy and data models.

Most policy models, even those designed for ubiquitous applications, are not able to represent access control choices within the complex dynamics of social interactions. Some recent efforts propose policy-based approaches to control access to shared resources in social networking applications [3]–[5]. Nevertheless, all these solutions rely on a rather simple modeling of social networks, which only includes the base "friendship" relationship, often considered as bi-directional (which is not the case in most real life social relationships, e.g., the "know of" relationship). In addition, current access control policy-based frameworks for social applications are generally designed to protect specific resources, such as media contents [3], [5] or medical information [4], and do not provide a generic model for any type of social information as an accessible resource.

This lack of generality, both in modeling access conditions and accessed resource, hinders the reuse of existing policy frameworks and their adaptation to new social application scenarios. Policies, based on various social constraints, should be designed to allow the specification of access control directives on different resources. Furthermore, they should be defined by relying on an interoperable data model to allow policy reuse across different user applications and policy exchange between different users of the same application. At the same time, policies should be customized for a specific application and its domain data model, to avoid imprecise and/or incorrect security directives. For example, for a user wishing to share different types of content (such as pictures and notes) via a social application, the generic concept for "content" would not be suitable as he would need to specify distinct policies for pictures and notes, respectively.

As far as the need for running infrastructures is concerned, not only are actual components needed to enforce access control, but those components should also be able to execute on a variety of mobile devices (e.g., smart phones) with minimal configuration effort. To the best of our knowledge, only a few solutions have been actually implemented as running systems [3], [5] but they may be difficult to reuse out of their proof-of-concept scenario, whereas no complete

The work is this article is partially supported by EC FP7 ICT NOE NESSOS project.

policy infrastructure exists for access control in mobile social applications. Note that the availability of fully implemented policy architectures has been crucial in the past towards the adoption of policies in real application scenarios for distributed systems, such as QoS and network management [6] or autonomous agent coordination [7].

A further requirement for the policy architecture is the availability of a user-interface to allow both application developers and end-users to specify their own security policies without dealing with the complexity of the underlying policy model. At present, policy frameworks for social applications often extend the interface of the target application [8], which are by definition application-specific, or they present policies according to their internal representation, thus requiring significant technical expertise to deal with the interface [3]. This is not suitable for social scenarios, where mobile users become the security administrator of their devices and applications, managing their own data and applications, rather than simply relying on external centralized or outsourced security management services [9].

Based on the above discussion, we claim the need for a flexible policy architecture to enforce access control in mobile social applications. Toward this goal, we make the following contributions. First, we have designed a novel policy framework for controlling access to social data in mobile applications. The framework is based on the semantic policy model presented in [10] and allows the representation of expressive policies based on users' *mobile social ecosystem* (MSE), that is, the rich set of social interactions occuring between people in mobile environments, according to various social relationships (e.g., friends, family or co-workers), different activities performed on content (commenting, tagging, etc.), as well as formation of groups and organization of events. The adoption of semantic technologies makes the extension of the policy model with new domain data models straightforward, while keeping policy model compatibility intact. Secondly, we have integrated the policy framework as part of Yarta, a middleware for managing mobile users' social ecosystems [11]. Third, we have provided a graphical policy editor to allow non-technology savvy users to easily specify and manage their access control policies.

The paper is structured as follows. Section II presents our policy framework, and in Section III we describe how we integrated it within the Yarta middleware architecture. Then, we provide details about the graphical policy editor we implemented (Section IV). To assess our contribution, in Section V we provide extensive evaluation of the current prototype implementation for smart phones. Finally, in Section VI we discuss related work. Conclusions and future work follow.

II. A Policy Framework for Mobile Social Ecosystems

Our policy model allows the specification of access control policies to protect users' social data, based on social data themselves. Each policy defines under which conditions (*i.e., the social context)* a given *resource* is accessible via a certain *action*. The accessible resource is any information within the

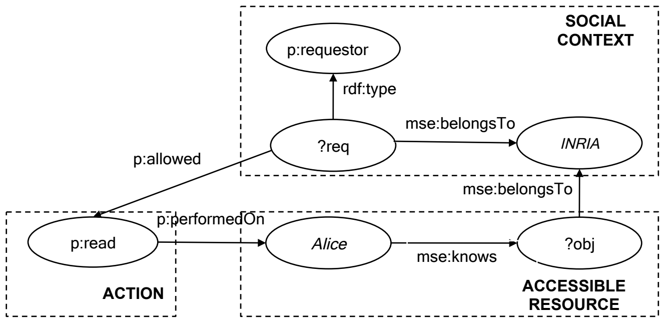


Fig. 1. Example Policy Graph

user's mobile social ecosystem. The action can either be `read`, `add`, or `remove` (a set of) triple(s) from/to the user's MSE data. The social context represents any socially meaningful information that constrains access to a resource. Note that the social context itself is expressed in terms of the user's MSE. For example, one may define the following policy: *Any member of INRIA can read the list of the friends of Alice who are also members of INRIA* Figure 1. In this case, the list of Alice's friends is the accessed resource (all nodes linked to alice via the *knows* relationship and are members of INRIA), while the social context is people who are members INRIA. Thus, the model supports the definition of access control policies based on users' social relationships and activities.

A. Policy Representation Details

We use the representational model presented in [10] to specify policies. In particular, a policy is represented as a set of attributes with predetermined values, either constant or variable with constraints over the range of values. The current knowledge describing the mobile social ecosystem where the access request takes place (and which is the target of the request as well) is also modeled in terms of attribute/value pairs. For a policy to be "in effect" the attribute values that define the MSE knowledge have to match the definition of the policy attributes with constrained values (i.e., policy constraints).

Both the base MSE model and the policy model are represented using the Resource Description Framework (RDF)[1], a base Semantic Web standard, and the RDF query language SPARQL[2]. In particular, the MSE is represented as a graph of RDF triples, i.e., subject-predicate-object triples, with each statement describing an attribute and its value. A policy is a set of RDF statements or SPARQL triple patterns. The set of RDF statements and SPARQL triple patterns defining a policy is linked as a graph of nodes and arcs. Figure 1 shows the graph-based representation of the example policy introduced above. The prefixes `p:` and `mse:` represent the namespace of the policy model and the MSE model, respectively, while terms preceded by question mark represent variables. For the sake of simplicity we omit the namespace for instances.

By relying on a graph representation, our policy model allows the definition of any type of logical relation between the

[1]http://www.w3.org/TR/rdf-primer/
[2]http://www.w3.org/TR/rdf-sparql-query/

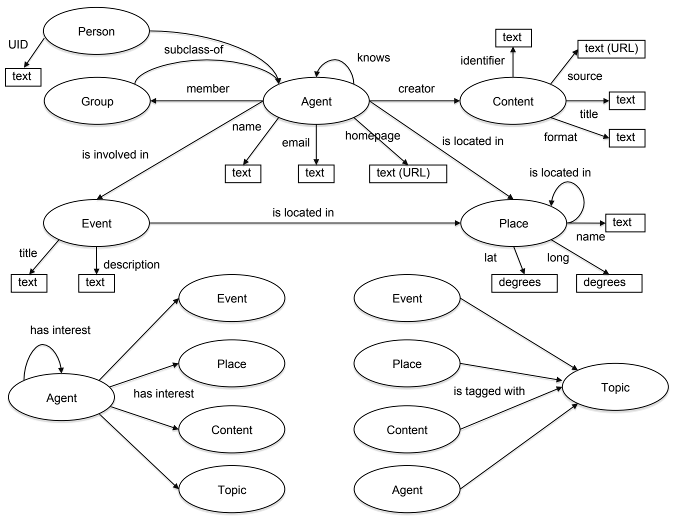


Fig. 2. MSE Data Model

base elements defined above, e.g., between the data requestor and the data owner, the requestor and the resource, the requestor and the environment, or any other relation the user might wish to specify. All these constraints can be specified by simply drawing new arcs between the policy nodes. Therefore, the model is able to represent existing policy models, such as role-based, identity-based or attribute-based, with enhanced expressive capabilities.

Finally, accessible resources can be defined in a flexible way by simply building appropriate graph patterns. For example, instead of controlling access to Alice's friends, we might control access to any information about her only by replacing the triple `(Alice, knows, ?obj)` with the less constrained `(Alice, ?pred, ?obj)`. Similarly, several different policies may be defined, possibly using multi-hop graph patterns.

We provide a rich representation of MSEs and the interactions possible in them based on the expressive and extensible model defined in [11]. For the sake of clarity we recall in Figure 2 the graph of first-class entities and relationships. Beside the availability of a base MSE model, access control policies for a specific mobile social application should be defined in terms of the domain knowledge characterizing that application. Towards this goal, our base MSE model is designed to be easily augmentable thanks to the extensibility features of RDF: this allows the easy specification of access control policies for different mobile social application scenarios.

On the other hand, RDF allows the association of a formal semantics to policy and MSE data models, and this supports simple reasoning over them. Because all mobile social applications share a common MSE data and policy model, they rely on a shared foundation of common meaning, which they can further extend based on specific requirements. By means of automated reasoning, classes and properties defined in application-specific extensions are put in clear semantic relation with base classes, thus enabling the reuse of existing policies in different applications. In summary, RDF reasoning capabilities, based on explicit semantics, and extensibility features are the key enablers for the reuse and exchange of policies between mobile social applications.

B. Policy Evaluation

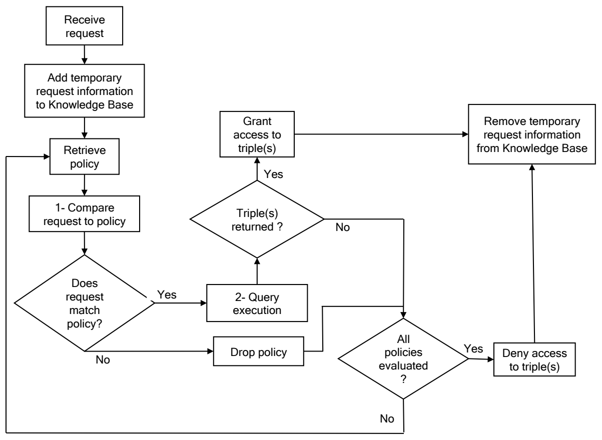

Receive request
Add temporary request information to Knowledge Base
Retrieve policy
1- Compare request to policy
Does request match policy?
Yes
No
2- Query execution
Triple(s) returned ?
Yes
No
Grant access to triple(s)
Remove temporary request information from Knowledge Base
Drop policy
All policies evaluated ?
Yes
No
Deny access to triple(s)

Fig. 3. Policy evaluation process

We now show how a policy is evaluated to answer a request for accessing data, given an MSE knowledge. Whenever a request is received, each policy currently enforced by the user is evaluated to determine whether the requested resource is accessible. In the following we describe the matching process for a single policy. The same process can be repeated for all policies. Note that, since the default behavior is to prohibit access unless differently stated, evaluation is performed until a policy granting access to the requested resource is found.

In general, access will take place within certain circumstances, i.e., a specific requestor asking to perform a given action on a resource. The resource can be known or unknown, depending on the action: in case of add/remove, the resource is known, while in case of a read action, the resource is in general unknown (because the requestor is typically asking for a type of information, but does not exactly know which information will be returned).

In particular, each access request on MSE data is represented as a SPARQL query, which always includes a set of RDF statements describing the requestor and the action. To avoid useless query execution, we perform an initial matching between the SPARQL queries representing the access request and the policy, respectively. Policy evaluation resolves then to the execution of two main steps as shown in Figure 3:

1. Matching the access request against the policy: This step is needed to verify if their graph patterns are compatible. If this is the case, the two queries are merged to create a single query, which combines the policy with information describing the current request (i.e., who is the requestor, what is the action, which resource is currently requested). If not, the policy does not apply to the current request and the request for access fails (for the considered policy).

2. Executing the resulting query over the knowledge base: Temporary information about the requestor and the action is added to the knowledge base, which also contains user's MSE data. Then, the combined query, obtained at the previous step, is executed over the knowledge base; if any result is returned (i.e., at least one RDF triple), access is granted to that result. If not, the request for access fails (again, for the considered policy).

Note that when the requested resource is a specific (set of) RDF triple(s), access is granted if and only if each triple is accessible. This case applies to `add`/`remove` actions only, where the requestor knows exactly which triples to access. On the other hand, if the requested resource is a SPARQL pattern (i.e., a `read` action), access is granted to only those triples that are allowed by current policies, while others are simply filtered out during the evaluation process.

C. An Example of Policy Specification & Evaluation

Let us recall the example policy represented in Figure 1: *Any member of INRIA can read the list of the friends of Alice who are also members of INRIA*. This policy can be formalized as the following SPARQL query:

```
CONSTRUCT {Alice mse:knows ?obj. }
where {     ?req rdf:type p:requestor
            ?req mse:belongsTo INRIA
            ?req p:allowed p:read.
            p:read p:performedOn Alice.
            Alice mse:knows ?obj.
            ?obj mse:belongsTo INRIA.}
```

Let us now consider the following access request: *Bob would like to see Alice's friends*. The corresponding SPARQL formalization reads as follows:

```
CONSTRUCT {Alice mse:knows ?obj. }
where {     Bob rdf:type p:requestor.
            Bob p:allowed p:read.
            p:read p:performedOn Alice.
            Alice mse:knows ?obj.
}
```

Step 1: The following elements are checked, to ensure that the element defined in the policy matches with the one defined in the request.

- Action. In this case, the action is read, as shown by the pattern `(p:read, p:performedOn, ?obj)`.
- Requestor. In this case, since the policy does not refer to a specific requestor's identity, the variable `?req` matches with `Bob`.
- Resource. In this case, the resource triple `(Alice, mse:knows, ?obj)` matches for the policy and the request, while the triple `(?obj, mse:belongsTo, INRIA)` needs to checked at the next step.

Step 2: The output of step 1 is the following SPARQL query:

```
CONSTRUCT {Alice mse:knows ?obj. }
where {     Bob rdf:type p:requestor          (1)
            Bob mse:belongsTo INRIA           (2)
            Bob p:allowed p:read.             (3)
            p:read p:performedOn Alice.       (4)
            Alice mse:knows ?obj.             (5)
            ?obj mse:belongsTo INRIA.         (6)
}
```

Temporary data, i.e., RDF statements #1, #3 and #4, are added to the Knowledge Base. The query is executed over the knowledge base, which contains both MSE data and temporary data. From the example we can see that the above query will succeed only if the MSE knowledge includes triple #2: `(Bob, mse:belongsTo, INRIA)`, and it will return all RDF triples (if any) matching the pattern constrained by statements #5 and #6. In other words, only if Bob is a member of INRIA, he will be returned the list of her friends who are also members of INRIA.

III. Integrating the Policy Framework within Yarta Middleware Architecture

We have integrated our policy framework within Yarta [11], a middleware architecture providing mobile social application developers with a set of functionalities that allow them to easily create, manage and securely exchange MSE data. The middleware consists of two layers, as shown in Figure 4: the *MSE Management Middleware* layer, which is responsible for storing, managing and allowing access to MSE data, and the *Mobile Middleware* layer, which takes care of low level communication/coordination issues.

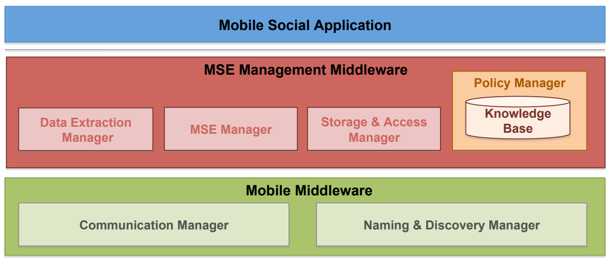


Fig. 4. Yarta Middleware Architecture

In this paper we only focus on components that, by integrating with the Yarta middleware, implement the policy model described above, namely the Knowledge Base and the Policy Manager. More details about the middleware architecture and its components can be found in [11].

Each user's social graph is managed by the *Knowledge Base* (KB) middleware component. The KB offers a set of interfaces describing the base concepts needed to manipulate RDF graphs, i.e., nodes, triples and graphs, as well as rich application programming interfaces to allow the retrieval, insertion and removal of data to/from the KB. The KB is also able to handle the merging of MSE graphs coming from different users.

To ensure access control enforcement according to the policies defined in the system, the KB is wrapped by a *Policy Manager* (PM). The Policy Manager intercepts any tentative access action on the KB, and performs reasoning on defined policies and the access request's context to determine whether the action is permitted. In particular, the PM evaluates all applicable policies with respect to current access conditions and, if no valid policy to allow access is found, it denies access to the requested resource (negative default).

In more detail, the PM middleware component, shown in Figure 5, consists of different subcomponents: the *Policy*

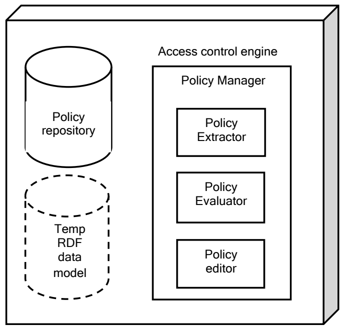


Fig. 5. Policy Manager components

Extractor, the *Policy Evaluator*, the *Policy Repository* and the *Policy Editor*. The Policy Extractor is in charge of extracting all user policies from the Policy Repository when the application is launched or when policies are edited. Extracted policies will then be ready for evaluation whenever a request for access is received. User policies are stored in the Policy Repository as a set of SPARQL queries, with each query representing a policy. This repository is accessible to the policy manager only.

The Policy Evaluator is in charge of performing policy evaluation upon access request. In particular, when a request is received, the Policy Evaluator receives information including the requested resource, the requestors identity, the requested action and a local copy of the user's MSE knowledge base. It adds temporary information about the request to its local knowledge base and combines it with extracted policies, as described in Section II. By executing the so obtained query on the local knowledge base, the Policy Evaluator builds a filtered data model, used to answer the access request.

In particular, the Policy Evaluator provides two main filtering functionalities. In case the input is a specific triple or set of triples (to be read, added or deleted from the graph), it returns a positive or negative response that allows or deny, respectively, access to that triple. In particular, for a set of triples, access is permitted only if all triples are accessible based on access control policies. In case the input is a query on the user's graph (only for read actions), the PM retrieves query results and filters them out based on current policies, thus returning to the KB only those triples that satisfy both the access request and the policies.

IV. Policy Editing

A powerful policy framework such as ours may be of little use unless the (non-technical) end-user is able to employ it to easily specify access control policies over his data. As seen in Section II-C, the specification of even a simple policy may need several lines of SPARQL, a language that itself can be quite intimidating to the end-user. Also, the end-user might not be familiar with the RDF representation of the MSE data model used by the social application in question. Instead, one would like to use a more natural interface, such as those used to set filters for incoming messages in email clients. For this reason, we have designed and implemented a policy editor that hides the complexity of low level SPARQL queries and RDF graphs. Users can add new policies, view, edit or delete existing ones, all via an intuitive graphical interface. The various categories of resources (e.g., Person, Group, etc.) are automatically made available in the editor, as are the instances currently in the knowledge base.

We illustrate the use of the policy editor by using the policy introduced earlier in the paper. As shown in Figure 6, Alice, the owner of the data, can specify this policy using our policy editor to identify the following 4 properties of the policy:

1. Requestor: This can be specified either intensionally – i.e., by selecting a specific person as the eligible requestor, or extensionally – i.e., in terms of specific sets that group people together based on common properties or criteria, such as members of a group (e.g., "INRIA") or friends of a person. The latter method allows the owner to grant access to a set of users using a single policy.

2. Action: This can be `read`, `add`, or `remove`.

3. Resource to be Accessed: The editor allows the owners to grant access either to all information in the KB, or very specific information (e.g., "the list of friends of a person") by selecting one of the resource categories in the policy editor.

4. Additional Conditions: Additionally, the owner can specify resource-related conditions that the result of the query should satisfy (e.g. "returned set of users must also be members of INRIA").

Although the policy editor makes the specification of policies easy, its content/options differ from application to application due to the different MSE data models used by them. Implementing the code for it for each application manually can therefore be tedious and error-prone. To alleviate the above problem, the options in the UI of the policy editor are generated dynamically, based on the MSE data model used by the application, as well as the current state of the owner's KB. In particular, we use the RDF description of the former to auto-generate the categories in the policy editor, while the individual members of the categories are populated dynamically by querying the KB at policy creation time. Finally, to allow users to define more complex policies, the editor also provides them with the raw SPARQL file which they can edit before it is loaded by the policy manager.

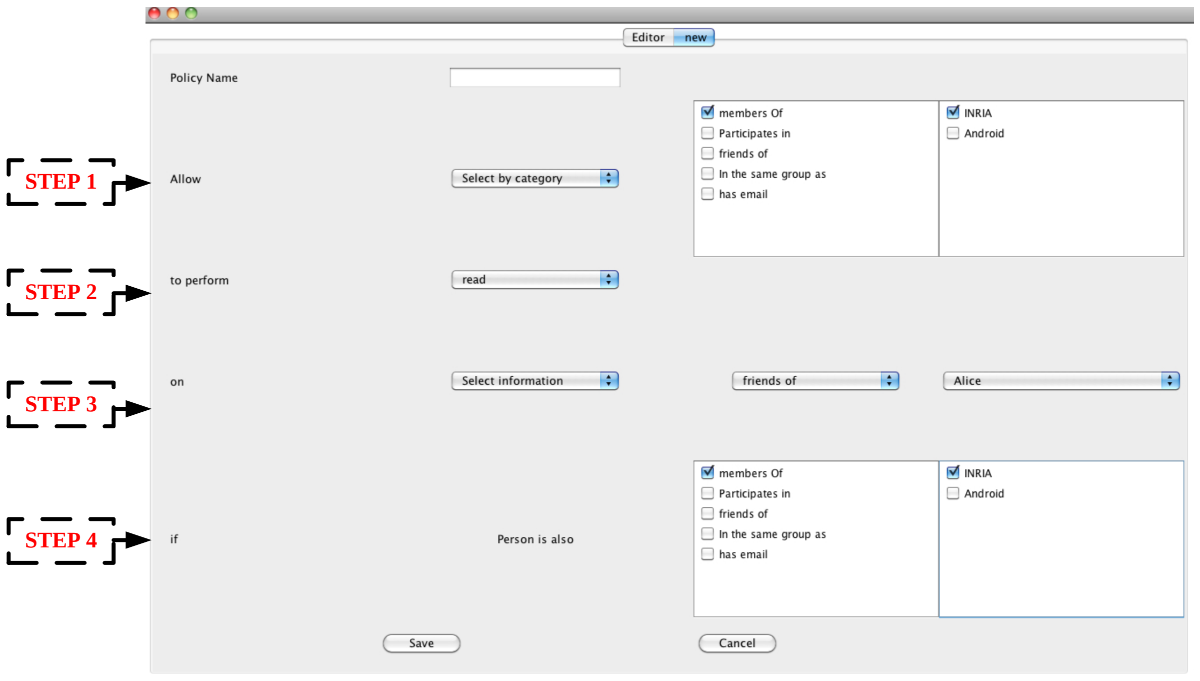
V. Implementation and Evaluation

A. Prototype Implementation

The Yarta middleware prototype is written in Java 2 SE and has been deployed both on laptops running Windows/Mac OS, and on smart phones running Android. Similarly to Section III, we mainly focus on Knowledge Base and Policy Manager components here.

In the laptop prototype, both the Knowledge Base and the Policy Manager rely on capabilities offered by the Jena Semantic Web Framework [12], which is at present the most comprehensive framework to manage RDF and Web Ontology Language (OWL)[3] data in Java applications. Jena provides a set of native APIs for data manipulation and also supports the

[3]http://www.w3.org/TR/owl-features/



Policy Name

Allow — Select by category — [✓ members Of] Participates in, friends of, In the same group as, has email — [✓ INRIA] Android

to perform — read

on — Select information — friends of — Alice

if — Person is also — [✓ members Of] Participates in, friends of, In the same group as, has email — [✓ INRIA] Android

Fig. 6. Policy editor, with annotations for each step in the policy specification process.

SPARQL query language to retrieve data according to graph patterns. We also rely on Jena features to handle duplicates due to the merging of multiple graphs. The KB currently uses the filesystem as a backing store. For the Android prototype we exploit Androjena[4], an Android-compatible port of the Jena framework that has been recently released. This allows us to transparently run Yarta on the laptop and on the mobile phone by replacing Jena libraries with Androjena ones.

The Policy Editor was written in J2SE using the Swing graphical framework. We use the RDF description of the data model to auto-generate the categories in the policy editor using the Apache Velocity template engine[5] and RDFReactor[6].

B. *Experimental Results on Smart Phones*

Ideally, the policy management subsystem should be responsive and scale well with KB size, and not depend on the type of the policy being evaluated. To evaluate the performance and scalability of the Policy Manager component, we executed several tests based on the following four parameters.

- size of KB: We increased the size of the knowledge base for social networks ranging from 400 to 2500 people.
- type of policy: We created three different policy sets. Each set presents a policy category - *Set 1* for requestor-related policies, *Set 2* for resource-related policies, and *Set 3* for policies with cross-related conditions, i.e., constraints binding different elements of policies (e.g., the requestor and the resource).
- type of response : Both `grant access` and `deny access` responses were generated
- type of action: We performed tests for `read`, `add`, and `remove` operations.

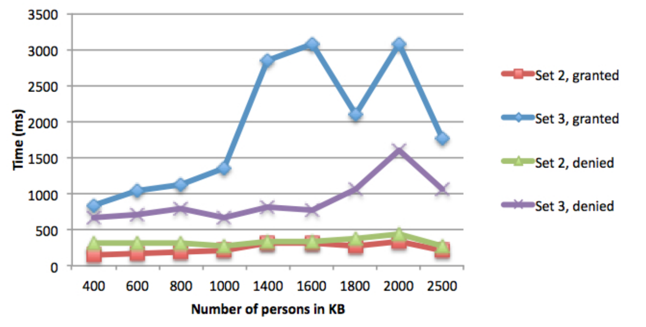
[4]http://code.google.com/p/androjena/
[5]http://velocity.apache.org/
[6]http://semanticweb.org/wiki/RDFReactor

Testbed Setup: The tests were run on a Google Nexus One mobile phone running Android 2.2 OS, equipped with a 1 GHz processor and 512 MB of RAM. To provide test cases, we exploited an anonymized data set that was gathered from Facebook [13]. We selected 10 sizes ranging from 400 to 2500 people, and for each size value we randomly extracted 10 sub-graphs from the data set using the snowball sampling algorithm [14], which has been shown to preserve the topological structure of graphs. For each extracted sub-graph, we created a KB (in RDF) containing nodes of type `Person` and `mse:knows` relationships between them. For testing purposes we also added `mse:email` and `mse:homepage` attributes for each person. Then, for each RDF file and each policy category defined above, we evaluated the following requests:

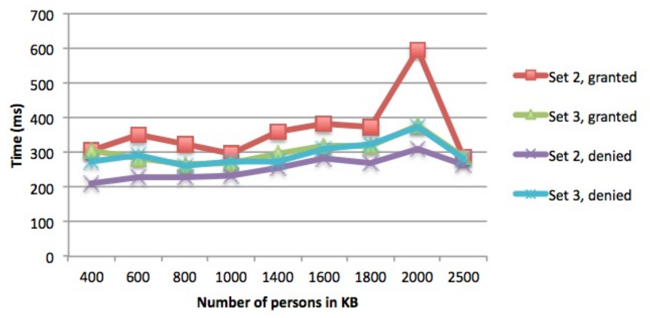
1) `add email`
2) `remove email`
3) `read email`
4) `read friends`
5) `read graph` with nodes about a person

and calculated the average evaluation time for all graphs per request for each policy set.

Experimental Results: The performance results of the Policy Manager are summarized in Figures 7 and 8. Due to lack of space, not all results are shown. The time needed to evaluate requests increases linearly with the size of the knowledge base. All types of action and policy were seen to show similar evaluation times (at most 600 milliseconds), except for the `read` requests evaluated with policies with cross-related elements, which took up to 3000 milliseconds for 2500 people, as shown in Figure 7(a). This is understandable given that

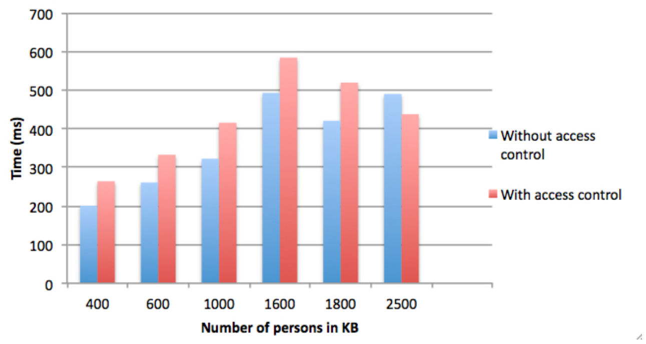


(a)

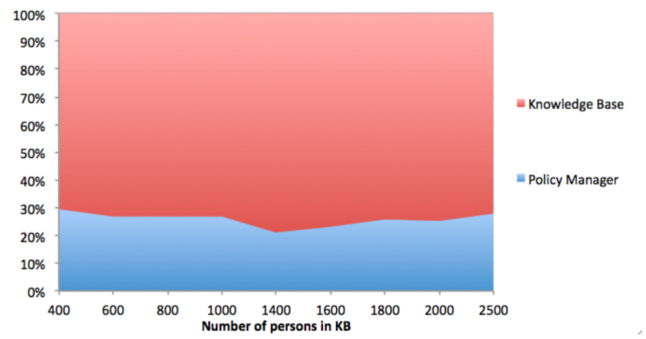


(b)

Fig. 7. Time taken by the Policy Manager to a) `read graph` b) `remove triple`



(a)



(b)

Fig. 8. a) `total time` and b) `time ratio` for add action and policy Set 3

cross-related conditions are more complex than strictly subject or resource-related ones.

Additionally, the response time was seen to be independent of the outcome of the evaluation, again with the exception of the `read graph` requests evaluated with policies with cross-related elements. Also, as shown in Figure 7(a), denying read access to a graph requires less time than allowing to do so.

Regarding the overhead introduced by the PM, we tested the evaluation times needed to execute requests with and without policy enforcement in place. The outcome showed that the overhead due to access control depends on the requested operation. As shown in Figure 8(a), enforcing access control on `add` requests results in a minor overhead unlike `read` requests results. This is mainly due to performance limitations of currently available implementation tools, which in case of a `read` have a major impact.

As for the share of work done by the PM, our experiments showed different behaviors for `read`/`remove` requests and `add` requests. In the former case, the PM requires more than 90% of the total time needed to execute a request issued to the KB, while in the latter it takes no more than 30% of the time. The reason is that, in the current implementation of `read`/`remove` operations, the PM performs time-consuming filtering operations on the knowledge base, while the KB only returns or remove triples. On the contrary, to answer `add` requests, the Knowledge Base performs more time consuming operations, such as range and domain checking. Note that `add` operations are generally faster than `read`/`remove` as they do not require retrieval of triples from the knowledge base.

VI. Related Work

With the increasing popularity of social applications, different solutions have been proposed in recent years to control access to personal data and content shared via social networking applications. In this section we discuss access control models and frameworks that are specifically targeted to social applications, possibly mobile.

A first category of solutions extends popular social applications (e.g., Facebook) to provide enhanced access control over personal data or added functionalities. For example, [4] proposes fine-grained access policies to medical data, based on the purpose of the requested access. The policy model is SecPAL [15], which however does not provide support for MSE data modeling nor semantic inference. Authors of [3] complement Facebook-like applications with support for collaborative policy editing, which can be useful when multiple users have interest in restricting access to a resource (e.g., a picture). To automatically infer user policies based on user behavior, authors of [5] propose machine learning techniques. These are all interesting efforts towards the provisioning of new functionalities for policy-based access control in social applications. However, they do not focus on providing expressive policy models nor enforcement infrastructures like Yarta.

Semantic technologies have also emerged as a promising choice to represent and reason about policies. In particular, recent work proposed to adopt semantic policies to control access to resources in social networking applications, such as Facebook [16] [17]. Both works propose, with some differences, to represent policies as Semantic Web Rule Language (SWRL) rules[7]: by reasoning on the social knowledge base, represented in OWL, they derive the set of active permissions. Similarly to us, these approaches provide rich descriptions of social interactions via semantic modeling. In [16] authors also introduce the concept of trust in a relationship, which we do not currently handle. However, since reasoning is performed on the whole knowledge base to infer the current list of permissions, this rule-based approach has two main limitations: first, forward reasoning requires the whole social knowledge to be stored in the same node in order to obtain all valid permissions, thus making the system inherently centralized. Furthermore, whenever a change in the social knowledge occurs, all permissions must be re-computed, which may raise efficiency concerns. Also, existing literature on rule-based systems reports of extremely critical situations, where the rule-base size made rule management such a complicated task to require a team of expert administrators [18].

Authors of [19] propose a relation-based access control model to support data sharing among large groups of users. The main idea is to represent permissions as relations between users and data, thus decoupling them from roles as in role-based access control. Authors provide a formal representation of their model, which includes E-R diagrams and their mapping to Description Logic, to allow reasoning. They do not, however, provide any specific social model as this is not their primary focus. In addition, the framework adopts hierarchies whose semantics may not always be clearly defined, which might make the model not easily manageable.

To the best of our knowledge, none of the systems presented above provide a full policy infrastructure like Yarta does. Some systems do provide proof-of-concept implementations, which however are designed for a specific application or resource, and we are not aware of their availability for reuse. On the contrary, our policy framework is generic and expressive enough to be used for any mobile social application, and it is integrated in a reusable middleware architecture whose components allow the enforcement of access control. Finally, only few solutions provide user-interfaces. In particular, [3] provides a user-interface, which however requires to deal with concepts like strong/weak conditions that might not be intuitive to define in practice, while [17] requires to deal directly with a SWRL editor. Our GUI allows non technical users to specify policies without dealing with the underlying SPARQL model.

VII. Conclusion

The novel scenarios enabled by emerging mobile social applications raise serious concerns regarding access control of users' contextual and social data. In this paper we presented a novel semantic policy framework for controlling access to social data in mobile applications. The framework allows the representation of expressive policies based on users' (MSE) and makes the extension of the policy model with new domain data models straightforward, while keeping policy model compatibility intact. We have integrated the policy framework as part of the Yarta middleware architecture and evaluated it via extensive testing. Finally, we provide a graphical policy editor to allow easy specification and management of users' access control policies. Our evaluation shows that our initial implementation of this novel policy framework scales well, although the response times have room for improvement in some cases.

Our current and future work include the validation of our policy framework in emerging application scenarios, such as social gatherings and smart city applications. We are also working on the optimization of the policy evaluation process and we are planning to test the usability of our interface via field testing with non-technical users.

[7]http://www.w3.org/Submission/SWRL/

References

[1] E. F. Churchill and C. A. Halverson, "Guest editors' introduction: Social networks and social networking," *IEEE Internet Computing*, 2005.

[2] Q. Jones and S. A. Grandhi, "P3 systems: Putting the place back into social networks," *IEEE Internet Computing*, vol. 9, no. 5, 2005.

[3] R. Wishart, D. Corapi, S. Marinovic, and M. Sloman, "Collaborative privacy policy authoring in a social networking context," 2010.

[4] P. Kodeswaran and E. Viegas, "A policy based infrastructure for social data access with privacy guarantees," *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, 2010.

[5] M. Shehab, G. Cheek, H. Touati, A. C. Squicciarini, and P.-C. Cheng, "User centric policy management in online social networks," *Policies for Distributed Systems and Networks, 2010 IEEE Intl. Symp. on*, 2010.

[6] L. Lymberopoulos, E. Lupu, and M. Sloman, "An adaptive policy-based framework for network services management," *Journal of Network and Systems Management*, vol. 11, no. 3, 2003.

[7] A. Uszok, J. M. Bradshaw, R. Jeffers, N. Suri, P. J. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, "Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement," *POLICY*, 2003.

[8] A. Besmer, H. R. Lipford, M. Shehab, and G. Cheek, "Social applications: exploring a more secure framework," *5th Symposium on Usable Privacy and Security*, ser. SOUPS '09.

[9] D. Smetters and R. E. Grinter, "Moving from the design of usable security technologies to the design of useful secure applications," *New Security Paradigms Workshop*, 2002.

[10] A. Toninelli, R. Montanari, O. Lassila, and D. Khushraj, "What's on users' minds? toward a usable smart phone security model," *Pervasive Computing, IEEE*, vol. 8, no. 2, pp. 32–39, April-June 2009.

[11] A. Toninelli, A. Pathak, and V. Issarny, "Yarta: A middleware for managing mobile social ecosystems " *International Conference on Grid and Pervasive Computing (GPC 2011)*, 2011.

[12] "Jena," last visited: May 2010, http://jena.sourceforge.net/.

[13] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," *4th ACM European conference on Computer systems*, 2009.

[14] J. Illenberger, "Estimating properties of snowball-sampled (social) networks," "http://matsim.org/uploads/Seminar2008_Illenberger _SnowballSampling.pdf", 2008.

[15] A. D. G. Moritz Y. Becker, Cédric Fournet, "Secpal: Design and semantics of a decentralized authorization language," *Journal of Computer Security*, pp. 619–665, june 2010.

[16] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham, "A semantic web based framework for social network access control," *14th ACM symposium on Access control models and technologies*, ser. SACMAT '09., 2009.

[17] N. Elahi, M. Chowdhury, and J. Noll, "Semantic access control in web based communities," *Computing in the Global Information Technology*, 2008. ICCGI '08.

[18] J. Bachant and J. McDermott, "Readings from the ai magazine," R. Engelmore, Ed. Menlo Park, CA, USA: American Association for A.I., 1988, ch. R1 Revisited: four years in the trenches.

[19] F. Giunchiglia, R. Zhang, and B. Crispo, "RelBac: Relation based access control," *Semantics, Knowledge and Grid, Intl. Conf. on*, 2008.