Enabling Privacy-Preserving Semantic Presence in Instant Messaging Systems

Anca Dumitrache[1,4], Alessandra Mileo[2], Antoine Zimmermann[3], Axel Polleres[2], Philipp Obermeier[2], and Owen Friel[4]

[1] DERI, NUIG, Galway, Ireland, {alessandra.mileo, axel.polleres, philipp.obermeier}@deri.org
[2] INSA-Lyon, LIRIS, France, Antoine.Zimmermann@insa-lyon.fr
[3] Cisco Systems, Galway, Ireland, ofriel@cisco.com
[4] Jacobs University, Bremen, Germany, a.dumitrache@jacobs-university.de

Abstract. In pervasive environments, presence-based application development via Presence Management Systems (PMSs) is a key factor to optimise the management of communication channels, driving productivity increase. Solutions for presence management should satisfy the interoperability requirements, in turn providing context-centric presence analysis and privacy management. In order to push PMSs towards flexible, open and context-aware presence management, we propose some adaptation of two extensions to standard XML-based XMPP for message exchange in online communication systems. The contribution allows for more complex specification and management of nested group and privacy lists, where semantic technologies are used to map all messages into RDF vocabularies and pave the way for a broader semantic integration of heterogeneous and distributed presence information sources in the standard PMSs framework.

Keywords: Presence Management Systems, XMPP, XML, Nested Groups, Privacy List, Context-awareness, Rule-based Policies

1 Motivations

Presence, also known as "presence information", conveys the ability and willingness of a user to communicate across a set of devices [4]. Presence has become an essential building block for many applications, both on the Web and in enterprise information systems, where being able to communicate efficiently with colleagues, customers, partners, suppliers and peers is essential. The concept of *presence* is a key ingredient for efficient communication, but it should take users' privacy into account, and the presence status of partners and resources should be provided to the other users according to such privacy settings.

When it comes to privacy-preserving Presence Management Systems (PMSs), the context in which users and resources may or may not be seen as available plays an important role as well, both in terms of physical context (e.g., location, ongoing meetings, booked resources), and virtual presence in online communication channels (e.g., IP telephony, video conferences or Instant Messaging (IM)). The contextual accessibility of presence information optimises communication time and hence time to resolution, in turn driving productivity increase, customer satisfaction and business revenues.

Despite the importance of delivering presence-based services to applications in both corporate and home environments, current PMSs still fail in providing a context- and privacy- aware presence management for users, which is in turn open and flexible enough to support easy integration of arbitrary sources of presence information through the use of open standards. In terms of interoperability, available standards for presence and presence-related information systems define an abstract model of presence, a data model, several data formats, and protocols that have been recently extended to provide additional flexibility, as detailed in Sect. 2.2. Unfortunately, the current standards and their extensions are still unsupported by the majority of IM tools, and they are not flexible enough to enhance interoperability, context-awareness and personalised privacy.

In order to push the existing PMSs towards a more flexible, open and context-aware concept of presence (later defined in this paper as *semantic presence*), we consider two existing extensions of the standard Extensible Messaging and Presence Protocol (XMPP) as detailed in Sect.2: the first extension enables to structure contacts of an IM client into (possibly nested) sub-groups (XEP-0083), while the second allows to define access to presence information via declarative rules, grouped into *privacy lists* (XEP-0016), for users to personalise the way they enable or disable communication with other entities or clients.

The main outcome of this work is the elaboration of these two extensions through the definition and implementation of i) mechanisms to deal with more complex users' taxonomies (including multiple inheritence for user groups) and ii) context-dependent privacy rules for accessing and sharing presence information. As part of our contribution, we address interoperability issues, by providing a mapping of both nested groups and privacy lists into semantic data structures based on RDF. This paves the way for the integration of PMS privacy settings with dynamic information that can be extracted from web sources (e.g. Facebook, Linkedin, etc.), email filters, google calendars and alike, as well as sensor information.

In order to make the reader familiar with the state-of-the-art in PMS standards and terminology, a basic overview is provided in Sect.2. Sect.3 provides a characterisation of the semantic presence framework, and Sect.4 shows how we modified the XMPP extensions for nested groups and privacy lists towards a context- and privacy- aware access to presence information that takes interoperability issues into account. A preliminary evaluation is presented in Sect.5 based on a prototype of the extended presence framework run on an XMPP presence server, and several open issues and next steps are summarised in Sect.6.

2 Overview and Concepts

2.1 Privacy and Access Control Policies

Policies are becoming the emerging paradigm for configuring complex systems and controlling the interaction between distributed entities, in that they represent an abstract way to define the changing behaviour of a system without changing the implementation. Privacy and Access control policies are the best known and well explored example of the use of policies in complex systems. They are concerned with the (declarative) specification of *who* is allowed to access *which* information according to some *conditions*.

Traditional role-based policies [14] cannot be applied when the role of the various actors is not known in advance (e.g. in open environments) or when it cannot be accessed by some applications. In this setting, attribute-based access control models turn out to help [5]. Context-centric approaches can provide a valid support to dynamic attribute-based access control, but given the heterogeneous nature of contextual information, some abstraction is needed to enhance interoperability.

Several solutions have been proposed, to semantically describe the conditions (or the *context*) in which policies are to be applied to enhance appropriate security [8, 12]. Some of these solutions support mainly context-awareness [1] and semantic interoperability for policy integration and reuse [16, 7, 17, 18]. In the *Proteus* policy model [15], a set of semantically annotated attributes trigger the appropriate context-related policy which returns permitted/forbidden actions for that specific context [16]. Conflict resolution between possible incompatible contexts are defined at design-time by specific constraints aimed at selecting which context should be active in case of conflict: this makes the approach not flexible enough to be adapted to different domains. Security issues have also been considered in *XACML* [11] though little support is provided for interoperability and flexibility to integrate new domain-related knowledge. The semantic policy language *REI* [7] provides a strong support to interoperability and flexibility in both integration and definition of policy for distributed security control, in particular in the definition and evaluation of actions, based on deontic logic, and in the conflict resolution mechanism, based on priorities and precedences.

The top-down approach that is generally adopted by the policy frameworks mentioned earlier provides a solid basis for research in this area, but it is sometimes too general to capture specific issues that are present in real environments, and target them appropriately. In this paper we adopt a bottom-up approach to allow for personalised privacy settings, rather than dealing with general security issues. We consider a context as a special view on presence information, which goes beyond the possibility of allowing or forbidding access to it: users can contextually adapt and transform the presence information by making a given rule list active, while semantic mappings enhance integration and interoperability.

Ideally, the general top-down and the specific bottom-up approaches will converge to a point where policy frameworks can benefit from generality principles but without losing the specificity needed to solve a given problem in a more effective and efficient way.

2.2 Presence Management System: Models and Protocols

A presence management system is described by connections between objects that expose their presence state (called *presentities*) and objects expressing standing interest in presence information related to a set of presentities (called *watchers*).

The IETF working group SIMPLE[5] published a set of standards for presence information systems. These are identified as RFCs followed by an identification number, and they define an abstract model of presence, a data model, several

[5] SIP, http://www.ietf.org/dyn/wg/charter/simple-charter.html

data formats and protocols for message exchange. In *RFC 3856*, **presence** is described as as *the ability, willingness, or desire to communicate across a set of devices*. In the abstract presence model introduced in *RFC 2778* [2], both presentities and watchers interact with the presence system via User Agents (UA), which manipulates presence information for them.

The flow of information from a presentity towards a watcher is ensured by a presence service as illustrated in Fig.1, where the *Principals* are the people, groups, and/or software outside the presence system, which use the system as a mean of coordination and communication.

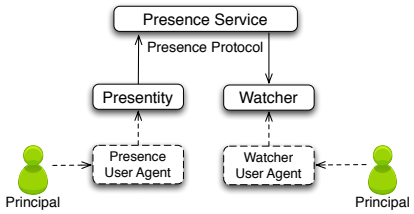


Fig. 1: Simple model of a presence service

In order to enable the near-real-time exchange of structured yet extensible data in form of messages between watchers and presentities of a PMS, the IEFT also defined the Extensible Messaging and Presence Protocol (XMPP)[6], a set of open technologies for instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalised routing of XML data. All messages exchanged in a PMS under XMPP are referred to as *stanzas*, while a user's *roster* consists of groups of contacts where each group is identified by a label.

The XMPP Standards Foundation (XSF) develops extensions to XMPP through a standard process centred around XMPP Extension Protocols (XEPs). Many extensions to XMPP have been proposed to enable representation and sharing of various context elements, e.g., current location and user activities. Among these extensions, we focus on those that mainly affect contextual and privacy-preserving presence management:

XEP-0083 on *Nested Roster Groups*, which allows the client's roster to store nested subgroups without breaking existing clients;

XEP-0016 on *Privacy Lists*, which defines a flexible method for communication blocking.

In the specific scenario of online PMS based on XMPP, dynamic and modular context-awareness for privacy setting is not properly addressed. In fact:

– it's not possible to define presence at different levels in accordance with some watcher's properties, nor to dynamically change presence information according to the context (activity, position, current agenda);

[6] http://xmpp.org

- there is a lack of support for appropriate policy mechanisms and a uniform semantic model of context for both the presentities and the watchers;
- policies are all-or-nothing: one can decide who to show presence to but it's not possible to share different presence statuses to different watchers;
- properties of watcher are strictly hierarchical: groups can be defined and nested, but they cannot incorporate multiple subgroups or be defined implicitly in terms of static properties;
- overlapping groups and distributed policies can create conflicts in presence results, and conflict resolution mechanisms should take contextual aspects into account in order to dynamically adapt to presentities' and watchers' contexts.

In this paper, we address these limitations by extending XEP-0083 and XEP-0016.

2.3 RDF, SPARQL and OWL

The *Resource Description Framework* (*RDF*) [10] is a data modelling language to describe resources on the Web identified by a URI. Resources are typically Web sites or services, but can also refer to physical or immaterial entities which are not directly obtainable through the Web. A RDF document is essentially a set of triples, whose components, in ascending order, are distinguished as *subject*, *property* and *object*. Each RDF triple (s, p, o), intuitively, specifies an instance of a distinct property p which puts a subject s and an object o into relation. Moreover, a RDF document can be interpreted as a graph containing nodes and arcs to represent resources (or literal values) and their relationships.

Similar to schemata for relational databases, the semantics of RDF data can be further enriched by ontological background information. Ontological languages can be encoded in RDF and support the definition of classes (of resources and literals) and properties, as well as the definition of inclusions of classes and properties. The W3C recommends the *Web Ontology Language* (*OWL*) [9], which is strongly influenced by Description Logics, a carefully analysed set of formal logics for concept descriptions with extensively studied fragments that are tractable or decidable with highly optimised algorithms and efficient implementations

Additionally, W3C recommends *SPARQL* [13], the standard language for querying RDF. SPARQL is well supported and widely deployed with a respectable number of query engines implemented and integrated in various RDF stores (Jena ARQ, Sesame 2, OpenLink Virtuoso, AllegroGraph, 4store, etc)[7].

3 The Semantic Presence Framework

Semantic Presence. Disclosed information in corporate environments should be controlled by personal policies as well as corporate level policies, such that only those watchers who conform to a certain profile can access specific presence information of a presentity, at a given time, in a certain (dynamic) context.

[7] A comprehensive list of SPARQL implementations is available at http://www.w3.org/wiki/SparqlImplementations.

Therefore, beyond the dynamic nature of heterogeneous presence information, users' privacy profiles should also play a role in the presence model. We therefore generalise the definition of presence to that of *semantic presence* as follows:

Definition 1. ***Semantic presence*** *[6] is defined as the* contextualised *availability of a person or a resource, where both the presentity's and the watcher's contexts are taken into account. The context-related aspects that contribute to the identification of semantic presence can be determined and influenced by:*

- *Presentities'/Watchers' physical/virtual presence;*
- *Presentities'/Watchers' profiles, including group membership and preferences;*
- *Presentities' privacy policies;*
- *Governing (Corporate) policies.*

In this work we specifically focus on profiles and privacy policies, and illustrate the interoperability of our approach via RDF mapping.

Scenario. We consider a user, Buster, and his communication with other user accounts connected via IM clients, in a real-work situation.

The group taxonomy for Buster is represented in Fig.2, and it constitutes an important source of contextual information for the semantic presence framework.

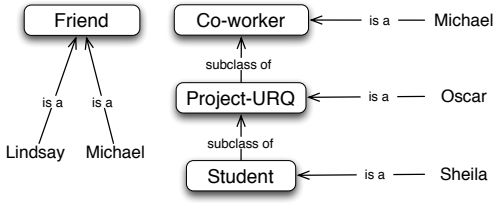


Fig. 2: Buster's taxonomy of groups

The active privacy settings for Buster are represented by a list of privacy rules, which are applied whenever Buster changes his status to "in a meeting":

1. show status as "available" to all co-workers;
2. show status as "unavailable" to students;
3. block all messages from contacts which are not part of the *co-workers* group.

When Buster joins a meeting with his co-workers on Project-URQ, he selects the status "in a meeting" (or it is selected automatically according to his calendar). As soon as the status changes, Buster receives a notification of a conflict, since two inconsistent rules with the same priority are applicable for Sheila: she is a student and so she should see Buster as "unavailable", but she is also a co-worker, thus she is allowed to still see Buster as "available". Buster is asked to solve this conflict by assigning priorities to the conflicting privacy rules, and these priorities will be applied by the semantic presence system in any future conflict related to that specific privacy list.

Requirements. A similar messaging system needs to deal with a more flexible and modular knowledge management mechanism for groups, contacts and their relations and properties. Desirable features of the systems are also its capacity of inferring implicit knowledge from explicit representations and dealing with a more fine-grained policy control. As a result, the user's status and the message content can be adapted to different classes of users via specific policies that are concerned with privacy issues and contextual settings. Presence information as well as the classification of users into groups should be dynamically updated, based on properties of the recipients and of the message. In this respect, semantic technologies can be useful to gather this dynamic knowledge and adapt policy evaluation accordingly. The general idea presented in this paper is that of enabling the XMPP-based Instant Messaging framework to:

- intercept and rewrite messages (or XMPP *stanzas*) based on high-level policies specified in a rule-like fashion and dynamically evaluated according to context- and user- related settings.
- map XMPP stanzas, groups and privacy rules into RDF (Resource Description Framework), a standard model for data interchange on the Web which facilitates data merging even if the underlying schemas differ, and specifically supports the evolution of data schemas over time without requiring all the data consumers to be changed.

All information passed through XMPP stanzas is stored on the server side, but since access to this information is restricted to a single authentified user, this sharing of information between client and server does not constitute a serious threat on privacy.

4 Implementation

The framework described in Sect.3 can be obtained by introducing two main extensions to XMPP: the first is related to the organisation of contacts of the IM client into a hierarchical taxonomy, and the second is related to the introduction of policy rules that determine how messages need to be managed according to how the sender and/or the receiver are related in this taxonomy. In the next subsections we detail how these two aspects have been implemented as extensions to XMPP.

4.1 Nested Groups

According to XMPP [3], a user's roster consists of a list of contacts, where contacts may have group labels attached to them, but there is no notion of "group" as a separate entity, a limitation in term of group management that our extension addresses. To improve this, we introduce the notion of *nested groups*, such that groups in rosters can be handled separately and related by a binary relation *subGroup*, where A subGroup B means that group A contains a subset of the users from group B.

Users are connected to an XMPP server via a Jabber client. Each user on this server is logged in by a personal user account and communication between server

and client takes place by an exchange of XMPP stanzas. In order to implement the concept of Semantic Presence, the Jabber software client has to support the regular roster management extended by methods for nested groups. Moreover, for each user account, all roster information including group taxonomy, is persistently stored at the XMPP server, in RDF.

The following operations need to be supported:

- adding subgroup relations, including multiple supergroups: in this case, the server should enforce that group members are also members of all the supergroups of the group they belong to;
- removing a subgroup relation: in this case, all the group membership that were the consequence of the nesting should not exist anymore.
- visualisation: a user is able to get a convenient hierarchical visualisation of the group nesting in his/her roster.

A user should be able to manipulate a contact or an entire group in his roster by adding, renaming or deleting groups as well as by defining *virtual groups* on the basis of (set) intersection or union of existing groups. Based on this rich classification of contacts into groups, privacy rules can be applied over intersection, union of groups, and any combination of these operations.

All information about a user's roster and groups are stored on the server side but since access to this information is restricted to a single authentified user, this sharing of information between client and server does not constitute a serious threat on privacy.

Group Management In order to allow for more complex group taxonomies to be created, we define new stanzas, such that it is possible to create/update/delete and relate groups independently of the roster items.

Example 1. Group creation:

```
<iq from='juliet@example.com/balcony' type='set' id='group_1'>
  <query xmlns='jabber:iq:group'>
    <group name='Servants'/>
    <group name='Friends'/>
  </query></iq>
```

The stanza in Ex.1 is creating two groups, 'Servants' and 'Friends'. The server must update the group information in persistent storage, and also push the change out to all of the user's available resources that have requested the roster. Stanza types are *presence*, *message*, *iq*. The roster push consists of an *iq* stanza of type *set* from the server to the client and enables all available resources to remain in sync with the server-based roster information. The server pushes the updated roster information to all available resources that have requested the roster and then replies with an *iq* result to the sending resource. As required by the semantics of the *iq* stanza kind, each resource that received the roster push must reply with an *iq* stanza of type *result* (see Ex.3) or *error*.

Example 2. Add subgroup relations:

```
<iq from='juliet@example.com/balcony' type='set' id='group_2'>
  <query xmlns='jabber:iq:group'>
```

```
    <group name='LocalFriends'>
      <subgroupof>Friends</subgroupof>
      <subgroupof>Nearby</subgroupof>
    </group></query></iq>
```

The stanza in Ex.2 defines 'LocalFriends' as a subgroup of two groups, 'Friends' and 'Nearby'.

Example 3. Client requests/receives current roster from server:

```
<iq from='juliet@example.com/balcony' type='get' id='group_1'>
  <query xmlns='jabber:iq:group'/>
</iq>

<iq to='juliet@example.com/balcony' type='result' id='group_1'>
  <query xmlns='jabber:iq:group'>
    <group name='Friends'/>
    <group name='Servants'/>
    <group name='Nearby'/>
    <group name='LocalFriends'>
      <subgroupof>Friends</subgroupof>
      <subgroupof>Nearby</subgroupof>
    </group></query></iq>
```

The stanza in Ex.3 shows both the request and the answer from the server which contains the list of all groups and their subgroup relations.

Example 4. Delete a group:

```
<iq from='juliet@example.com/balcony' type='set' id='group_4'>
  <query xmlns='jabber:iq:group'>
    <group name='Nearby' remove='true' />
  </query></iq>
```

As with adding a group, when deleting a group the server updates the group information in persistent storage, initiate a group push to all of the user's available resources that have requested the group hierarchy (with the 'remove' attribute set to a value of *true*), and send an *iq* result to the initiating resource. The server should return an error when trying to remove a non empty group, or a group having subgroups.

We now illustrates how privacy rules can be used to enable Semantic Presence.

4.2 Privacy Lists

The same mechanism used for group management, is also applied for creating privacy lists composed by privacy rules: the roster pushes an *iq stanza* for the creation of both the privacy list and the policy rule. Directions are either *inbound* or *outbound*, meaning that the rule apply to incoming and outgoing stanzas. Subscription types restrict the application of the rule depending on mutual subscription (or lack thereof) of the sender and receiver. The values can be *to*, *from*, *both* or *none*. When all criteria for a rule are match, the *action* is performed, as explain below. Ex.5 illustrates a privacy list stanza with one rule.

Example 5. Create a privacy list and add rules to the list:

```
<iq from='juliet@example.com/balcony' type='set'
  <query xmlns='jabber:iq:privacy'>
    <list name='MyList' />
  </query></iq>
```

```
<iq from='juliet@example.com/balcony' type='set'
  <query xmlns='jabber:iq:privacy'>
    <list name='MyList' />
      <item type='jid'
           value='Juliet@example.com/balcony'
           action='allow'
           order='4'>
        <presence-in/>
      </item></list></query></iq>
```

Analogous to roster and group information, a users's privacy list is stored on the server side and can only be accessed via authentication.

Syntax of Policy Rules Formally, we define a (policy) rule as an expression

$$Action \leftarrow Condition$$

where Action stands for either *allow*, *block* or *transform(e)* with a POSIX extended regular expression e; Condition is an expression defined over a set of fixed predicate symbols *Preds* and a set of constant symbols *Cons*. Specifically, a Condition can be an expression of the form

1. $p(c)$ where $p \in Preds$ and $c \in Cons$;
2. $expr_1 \wedge expr_2$ where $expr_1, expr_2 \in Cons$;
3. $expr_1 \vee expr_2$ where $expr_1, expr_2 \in Cons$;
4. $\neg exp$ where $exp \in Cons$.

Furthermore, *Cons* exclusively comprises the constants *payload*, *presence*, *message*, *iq*, *to*, *from*, *both*, *none*, *inbound*, *outbound*, all possible group names and all POSIX regular expressions, while *Preds* is a fixed set which exclusively contains the symbols *stanzaType*, *direction*, *subscriptionType*, *affects*, *owner* and *regexMatch*.[8]

Semantics of Policy Rules The evaluation of a policy rule p is always tied to an XMPP session of a Jabber user and a current send or receive action of a stanza by the user. We formally integrate these diverse information as a contextual session $cs = (cs_{jid}, cs_{groups}, cs_{presence}, cs_{stanza})$ which holds the user's Jabber ID (cs_{jid}), the roster groups (cs_{groups}), given as a set of JIDs, the user's presence status ($cs_{presence}$) and the stanza which is currently processed (cs_{stanza})., i.e. sent or received by the user.

The evaluation of the rule itself is conducted in two major steps. First the truth value of the condition has to be determined. Second, if the Condition is true the Action has to be applied. A function *atomicTv* maps each condition $c \in Cons$ to true or false as follows:

[8] Note that future extensions may generalise rules to include other n-ary predicates.

$atomicTv(stanzaType(c))$	is true iff c is the stanza type of cs_{stanza}
$atomicTv(direction(c))$	is true iff c is the direction of cs_{stanza}
$atomicTv(subscripitonType(c))$	is true iff c is the subscription type of cs_{stanza}
$atomicTv(affect(c))$	is true iff c is the 'from' or is in a group that contain the 'from' (for an inbound stanza) or c is the 'to' or is a group that contains the 'to' (for an outbound stanza) of cs_{stanza}
$atomicTv(owner(c))$	is true iff c is cs_{jid}
$atomicTv(regexMatch(c))$	is true iff the payload of cs_{stanza} matches the regular expression c

If the condition of the policy rule is true, the corresponding action is applied to *cs_stanza*, and the message can be i) blocked, ii) allowed or iii) transformed by a sed (Unix stream editor) rewriting command e and afterwards forwarded to the receiver.

An example of rule could be

$$\begin{aligned} block \leftarrow\ & stanzaType(message) \\ & \wedge regexMatch(/.*\texttt{Cisco}.*/) \\ & \wedge \neg affects(Cisco{-}Staff) \\ & \wedge subscriptionType(to) \end{aligned}$$

which says that each message containing the word 'Cisco' is blocked when sent to a user who is not in group 'Cisco-Staff'.

Processing Privacy Rules The processing of the privacy rules is made in two steps. The first step consists in retrieving the potentially applicable privacy rules from the RDF store with a SPARQL query, based on the type of the stanza, the sender, the recipient and the subscription type assigned to the contact affected by the rule. This may involve some reasoning over the RDF data, especially to accurately determine implicit group membership from the group nesting. An optimised implementation would store user specific information (roster and privacy list) in memory so that this step can be performed very fast.

Rules that are retrieved this way may still not apply to the stanza in question, because there can be additional conditions that cannot be directly represented in RDF, such as a time frame, a location an expression of the possible payloads affected by the rules (cf. *regexMatch* in our simple policy rule syntax). To decide this, a dedicated policy engine interprets the condition (which must be in a language understandable by such an engine) and evaluates it as explained above. Rules are processed in order of priority and whenever a rule is applicable, the engine stops processing the rest of the rules fir that privacy list. Software clients should inform users when two rules with equal priority apply to a given stanza.

Enabling Interoperability In order to make our proposal easily extensible, we provide a mapping of XMPP stanzas into RDF, for both nested groups and privacy rules. This also allows roster groups to be interconnected with existing enterprise level staff member management. For instance, an RDF version of Active Directory

```
<iq type='set'>
  <query xmlns='jabber:iq:groups'>
    <group name='URQ'>
      <subgroupof>
        Coworkers
      </subgroupof>
      <subgroupof>
        Semantic Search Stream
      </subgroupof>
    </group>
  </query>
</iq>
```

(a)

```
group:urq
  rdf:type sioc:UserGroup ,
           owl:Class ;
  rdfs:label "URQ" ;
  rdfs:subClassOf
           group:coworkers ,
           group:semSearchStream .
```

(b)

Fig. 3: Translation of roster group 'URQ' given by stanza (a) into an OWL class (b)

could be integrated in the roster management system to facilitate categorisation of contacts. The mapping is illustrated in Fig.3b and Fig.4b and it paves the way for integrating heterogeneous sources of contextual knowledge (provided that the same semantic mapping is given for this knowledge), such as sensors and corporate policies.

```
<iq type='set'
    from='michael@corp.com'>
  <query xmlns='jabber:iq:privacy'>
    <list name='MyList'>
      <item type='group'
            value='Coworkers'
            action='transform'
            ed='available'
            order='3'>
        <presence-out/>
      </item>
      <item action='allow'
            order='68'/>
    </list>
  </query>
</iq>
```

(a)

```
:Michael
  rdf:type privacy:UserAccount ;
  privacy:has_privacy_list
           list:myList .
list:myList
  rdfs:label "MyList" ;
  privacy:has_privacy_rule
           rule:myRule .
rule:myRule
  privacy:affects group:Coworkers ;
  privacy:applies_to
           privacy:presence ;
  privacy:has_action
           transform:available ;
  privacy:priority "3" ;
  privacy:direction
           privacy:outbound .
transform:available
  privacy:transformation
           "available" .
```

(b)

Fig. 4: Translation of a privacy list (a) into OWL (b)

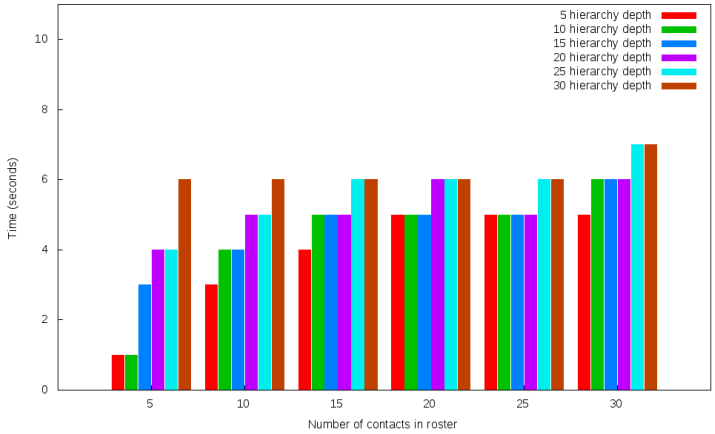
5 Preliminary Experiments

We conducted a set of preliminary experiments based on the runtime of the application in various extreme cases, for the purpose of assessing how our component will scale in a real-world situation (i.e. having a large group hierarchy of up to 30 nested groups, as defined in the Psi[9] XMPP IM client). More intensive testing will be done in future experiments, when the conflict resolution mechanism will be made more expressive and some of the issues mentioned in Sect.6 will be tackled.

As seen in Fig.5, the reaction time grows proportionately with the increase of the number of users and the hierarchy depth, the likely reason being that the

[9] http://psi-im.org/

Fig. 5: Approximate runtime (in seconds) when applying privacy XMPP stanza.



implementation is currently non-optimised w.r.t. the number of queries against the triple store. However, when discussing a real-world scenario, it should be taken into account that, even in a corporate environment, roster groups are usually declared and maintained by users, and therefore the complexity of their hierarchy is limited by the fact that groups need to stay human-readable. In this context, hierarchy depths of more than 10 become very unlikely. As demonstrated by our evaluation, in cases with small group depth, the runtime measurements showed a negligible delay. We conclude that, even if the runtime of the component increases with group depth, this would have little impact in a real-world scenario, and therefore our component can be deemed scalable.

6 Future Work and Conclusion

In this paper we present a bottom-up approach to specify and evaluate privacy settings in XMPP-based PMSs on the basis of contextutal group information. The proposed implementation takes advantage of semantic technologies for flexible and interoperable solutions to Semantic Presence and it paves the way for numerous challenging issues we already started to tackle, including the following:

Temporal-/location- based validity of rules refers to advanced privacy rules to change the granularity of the location attached to presence information: for instance, colleagues may know the exact room in which the person is, while external collaborators are only notified with the city in which the user is.

Federated policies consider that presence should reconcile general company policies with users' privacy settings. Interruptions are counterproductive in the workplace and adequate handling of (corporate) policies to avoid these could be of potentially high impact. Some companies that ban the use of social networks for those reasons, so active support by presence systems might be desirable.

Physical presence is also a key aspect in context-aware pervasive environments, thus the extention of XMPP to take into account sensor data, aggregated at

the right level of granularity is of interest, where the management and processing of a large amount of dynamic sensor data can be a challenge.

Conflict analysis requires to deal with efficient conflict resolution strategies, taking presence information, groups structure and roles into account.

These aspects are all relevant for advanced semantic presence management and we believe semantic technologies and abstract policy specification and enforcement introduced in this paper can help addressing these challenges.

References

1. Acharya, A., Banerjee, N., Chakraborty, D., Dasgupta, K., Misra, A., Sharma, S., Wang, X., Wright, C.: Programmable presence virtualization for next-generation context-based applications. In: PerCom. pp. 1–10 (2009)
2. Jabber Software Foundation: A model for presence and instant messaging (2000)
3. Jabber Software Foundation: Extensible messaging and presence protocol (xmpp): Instant messaging and presence (2004)
4. Jabber Software Foundation: A presence event package for the session initiation protocol (sip) (2004)
5. Frikken, K., Atallah, M., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Transactions on Computers 55, 1259–1270 (2006)
6. Hauswirth, M., Euzenat, J., Friel, O., Griffin, K., Hession, P., Jennings, B., Groza, T., Handschuh, S., Zarko, I.P., Polleres, A., Zimmermann, A.: Towards consolidated presence. In: CollaborateCom 2010.
7. Kagal, L., Finin, T., Joshi, A.: A policy based approach to security for the semantic web. In: ISWC2003
8. Ko, H., Won, D., Shin, D., Choo, H., Kim, U.: A semantic context-aware access control in pervasive environments. In Proc. of: ICCSA 2006. LNCS vol. 3981, pp. 165–174 (2006)
9. McGuinness, D.L., van Harmelen, F.: OWL web ontology language overview. W3C recommendation, W3C (Feb 2004)
10. Miller, E., Manola, F.: RDF primer. W3C recommendation, W3C (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
11. Moses, T.: eXtensible Access Control Markup Language (XACML) Version 1.0. (February 2003)
12. Priebe, T., Dobmeier, W., Kamprath, N.: Supporting attribute-based access control with ontologies. Int'l conf on Availability, Reliability and Security, 465–472 (2006)
13. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C recommendation, W3C (Jan 2008), http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
14. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. Computer 29(2), 38 –47 (1996)
15. Toninelli, A., Montanari, R., Kagal, L., Lassila, O.: A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In Proc. of: ISWC 2006. LNCS, vol. 4273, pp. 473–486.
16. Toninelli, A., Montanari, R., Kagal, L., Lassila, O.: Proteus: A semantic context-aware adaptive policy model. In: POLICY. pp. 129–140 (2007)
17. Uszok, A., Bradshaw, J.M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., Aitken, S.: Kaos policy management for semantic web services. IEEE Intelligent Systems 19, 32–41 (July 2004)
18. Uszok, A., Bradshaw, J.M., Lott, J., Breedy, M.R., Bunch, L., Feltovich, P.J., Johnson, M., Jung, H.: New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of kaos. In: POLICY. pp. 145–152 (2008)