Presence@FER: An Ecosystem for Rich Presence*

Ivana Podnar Žarko, Mario Kušek, Krešimir Pripužić, Aleksandar Antonić

Faculty of Electrical Engineering and Computing (FER), University of Zagreb, Croatia
{ivana.podnar_zarko, mario.kusek, kresimir.pripuzic, aleksandar.antonic}@fer.hr

Abstract— **Presence information—expressing user willingness and ability to communicate with other users across a set of devices and tools—represents an essential prerequisite for real-time communications. Although presence-related protocols have primarily emerged in conjunction with instant messaging systems, presence is today viewed as a primer service offered by contemporary unified communication platforms, and is often referred to as the dial tone of the 21st century.**

The paper presents Presence@FER, an ecosystem for presence based on the Rich Presence Information Data format (RPID) which comprises both XMPP and SIP-based solutions for presence. Presence@FER supports context-aware collection and exposure of rich presence information, and offers fine-grained filtering of presence information in accordance with user context and predefined policies. It consolidates virtual, physical, and online presence into a single rich-presence platform which relies on content-based publish/subscribe service for efficient filtering and dissemination of presence information. We show a number of applications that can be built of top of such rich-presence platform, and provide details of our prototype implementation.

I. Introduction

We are witnessing a big step forward in the area of communications system software with the rise of novel platforms and tools for unified communications which typically comprise voice/multimedia telephony, instant messaging (IM), and e-mail. Such systems necessarily integrate a presence service for managing and disseminating presence information generated by user contacts, such that a user is aware of the current presence status (e.g. available, busy, unavailable) prior to placing a call or sending an instant message. Moreover, we are faced with an abundance of novel context-aware and location-based services such as Google Latitude, Facebook places, or Foursquare, that also need to maintain and distribute current presence and location information of their users while taking into account user privacy settings.

In the context of presence services, presence is defined as the willingness and ability of a user to communicate across a set of devices with other users. Presence service enables users (*watchers*) to subscribe to presence information generated by their contacts (*presentities*), and to receive their presence updates in real-time. However, existing presence solutions typically ship all generated presence updates from presentities to watchers, without taking into account watcher context. Indeed, watchers are not really interested to closely and continuously follow their contacts, but would rather prefer to receive filtered presence notifications at their own convenience. In addition, presentities would also like to control the disclosure of their presence information to watchers, especially when it comes to personal context-related presence information. For example, Alice would like to receive a presence notification when her contact Bob is nearby (e.g. in the office next door, or on the same floor), in a good mood, while his presence status is set to available. On the other hand, Bob specifies that his current location during office hours may only be shown to his boss, wife, and a few colleagues. As Alice is on the list of Bob's colleagues, his presence status is sent to Alice including the current location when conditions of her presence-related subscription are met.

Presence information is event-based and generated in an ad-hoc fashion. It is disseminated in real-time from one source to many destinations following the publish/subscribe communication model [1]. However, current presence service implementations typically support rather simple presence subscriptions capturing all presence updates generated by a single presentity, and integrate only topic-based publish/subscribe solutions [2]. This generates a large number of messages that need to be transmitted both from and to a watcher terminal while using up its battery power rather quickly, and introduces serious scalability problems within the core network due to presence-related signalling [3]. Content-based publish/subscribe systems offer a solution to the aforementioned problem as they implement fine-grained filtering and efficient matching algorithms that can greatly reduce the number of exchanged presence updates, and enable the definition of personalized subscriptions, i.e., presence-related filters, that can filter out presence messages for which there is no current interest.

The need for novel context-aware presence solutions has been identified in [4]. The authors propose a *consolidated presence* service for corporate environments, and investigate the requirements and technologies needed for implementation of such a service. In addition to deployed industry standards and products, a number of specific presence-related solutions has lately been published in the literature [5], [6], [7], [8], which shows that the management of sensitive personal presence information

*Acknowledgment. This work was carried out within the research project "Content Delivery and Mobility of Users and Services in New Generation Networks", supported by the Ministry of Science, Education and Sports of the Republic of Croatia.

is still challenging.

We present a presence solution which supports the Rich Presence Information Data (RPID) format [9] including context information, such as location, mood, and activity, and extends the basic presence format, the Presence Information Data Format (PIDF) [10]. Our presence solution enables context-aware collection and exposure of presence information because it enables a watcher to specify special conditions that need to be satisfied such that he/she receives a presence update, while a presentity defines the policies for the collection and dissemination of presence updates. It is in line with RFC 4661 [11] that specifies content-based filtering rules associated with a presence subscription.

We design a special rich presence service (RPS) for handling RPID status messages and defining context-aware presence filters. The RPS uses a content-based publish/subscribe system for efficient matching of RPID status messages to presence filters, and dissemination of matching status messages to interested watchers. The RPS is designed as an independent component that can be integrated with the two standardized presence solutions: Session Initiation Protocol (SIP) Presence and Extensible Messaging and Presence Protocol (XMPP). It can therefore be used as a gateway between the two deployments. Please note that our solution may integrate various presence sources, such as sensors and calendars as sources of *physical presence*, *virtual presence* from various communication channels, e.g., IP telephony/multimedia, IM, and *online presence* from, e.g., Facebook, LinkedIn, and Twitter. Furthermore, we present a number of context-aware presence applications built on top of the rich presence layer as a showcase of our presence platform named Presence@FER. Therefore, we add context-awareness and flexibility into the presence domain which allows the development of a new class of applications based on rich presence. This can potentially also solve the scalability issues and overload with presence updates as experienced by the end users.

The main contributions can be summarized as follows:

- We present a novel rich presence architecture for context-aware dissemination and filtering of rich presence information;
- We integrate our rich presence solutions with a number of physical, virtual, and presence sources and design a number of presence-related applications;
- We provide a complete rich-presence solution as a testbed for future rich presence applications.

The paper is structured as follows. Section II provides an overview of standardisation efforts in the area of presence management because a plethora of presence-related standards and protocols exists today. Section III gives an overview of the `Presence@FER` ecosystem, describes our rich presence architecture, and briefly describes implemented example applications. We present the details of our rich presence prototype in Section IV. Related work in covered in Section V, and we conclude the paper in Section VI.

II. Presence-Related Standards and Protocols

During the last decade, significant efforts have been put into the process of presence service standardization by developing a suite of services collectively known as *Instant Messaging and Presence*. The Internet Engineering Task Force (IETF), being the main standardization body in this domain, has adopted two competing protocol suites in parallel, namely, *SIP presence* and *Extensible Messaging and Presence Protocol* (XMPP).

Both SIP presence and XMPP are based on the abstract model defined in RFC 2778 [12]. The presence service model defines two types of clients: *presentities*, which expose their presence state, and *watchers*, which express interest in presence information related to a set of presentities. A watcher interest can be expressed either as a *single request* for the current presence status, or as a *continuous subscription* to presence updates. Presence information contains a set of *presence tuples*, where each tuple includes status information (e.g. online, offline, busy, away, do not disturb), and optionally a *communication address* and additional *presence markup information*. To enable the exchange of presence information between watchers and presentities, the presence service accepts and distributes presence information to interested parties using a *presence protocol* such as XMPP or SIP Presence. RFC 3859 [13] defines a Common Profile for Presence (CPP), i.e., semantics and data formats for presence that different presence services must support to enable basic interoperability and federation. It defines the "pres URI", a unique identifier for watchers and presentities, and the structure of three basic messages: *subscribe* (a message from watcher to presence service specifying watcher subscription to presence information generated by a presentity), *response* (a response message to subscribe) , and *notify* (a message which carries presentities presence information to a watcher).

SIP Presence is developed within the SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) Working Group[1]. It comprises a set of presence-related protocols, data models, and data formats that use SIP [14] as the underlying communication protocol. SIP architecture and associated protocols are reused for presence because SIP location services already maintain certain user-related presence information in the form of user registrations, and furthermore, SIP networks are capable of routing requests from any network to the server that holds the registration state for another user. Thus, SIP routing inherently supports routing of watcher interests in multi-domain networks.

The core SIP Presence protocol describing the usage of SIP for routing presence-related subscriptions and notifications of presence is defined in RFC 3856 [15]. RFC 3863 [10] defines the basic document format for representing presence information named Presence Information Data Format (PIDF) in accordance with the abstract

[1]SIMPLE Working Group: http://www.ietf.org/dyn/wg/charter/simple-charter.html

model defined in RFC 2778, while RFC 4480 [9] and RFC 4481 [16] extend PIDF with rich presence attributes and future/past presence information, respectively. The Rich Presence Information Data (RPID) format [9] is introduced to extend PIDF with context-based presence information. While keeping the backward-compatibility with PIDF, RPID defines additional presence attributes to associate person, service, and device data elements with rich presence information such as current activity (e.g. lunch, meeting, vacation), mood (e.g. happy, angry, impressed), location type (e.g. office, library, street), and location property (e.g. noisy, dark, uncomfortable). Our presence system uses RPID as the data format for presence status representation.

XMPP is developed originally by the Jabber open-source community and subsequently standardized by the XMPP Working Group[2]. The core version of this protocol is defined in RFC 3920 [17], while RFC 3921 [18] defines its extensions. In brief, RFC 3920 defines the XMPP network architecture which includes clients, servers, and gateways to foreign network, and uses XML streams for communication between network elements, while RFC 3921 defines a data format for presence stanzas, XML elements within an XML stream, which are used to convey presence-related information. A presence stanza includes the following information: message source and destination, presence state accompanied by an optional free text note, and priority level of the source.

When comparing XMPP to SIP Presence, XMPP is often promoted as being substantially simpler and with a large user base. Estimates are that tens of thousands of XMPP servers have been deployed around the globe with more than 50 million users [19]. Conversely, SIP Presence has been widely adopted in the telecom world with the development of IP multimedia subsystem (IMS) which uses SIP as the underlying signaling protocol. The Open Mobile Alliance (OMA) defines a presence enabler which manages and disseminates presence information over the IMS presence architecture using SIP Presence protocols [20].

Publish/Subscribe and Presence. Both SIP Presence and XMPP community have recognized the need for efficient filtering of presence updates, especially when presence clients are used on mobile devices. However, to our knowledge, there is limited or no support for context-aware and fine-grained filtering of presence updates in both presence worlds. RFC 4660 [21] defines the operations performed by a a subscriber when declaring filtering rules associated with a presence subscription, while RFC 4661 [11] specifies the filter format. The two RFCs assume that presence information is encoded as an XML document, and define a filter with two elements: The *what* element is an XPath expression and defines the content to be delivered to the user. The *trigger* element is used to identify the package-specific changes that a resource has to encounter before the content is delivered to the subscriber.

[2]XMPP Working Group: http://www.ietf.org/dyn/wg/charter/xmpp-charter.html

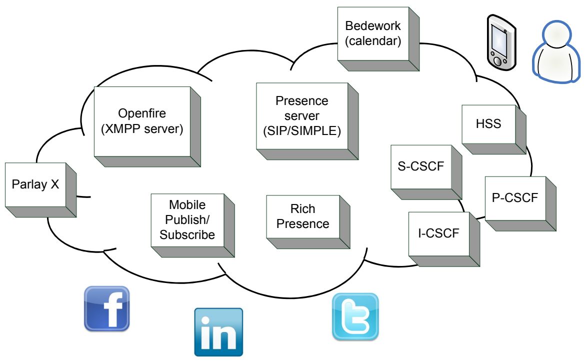

Figure 1. The Presence@FER ecosystem

XMPP extensions XEP-0060 [2] and XEP-0163 [22] define the publish/subscribe functionality for XMPP-based presence solutions. XEP-0060 specifies the full publish/subscribe protocol used to control which type of extended presence data is sent to subscribers specifically authorized to receive topic updates. In general, the relationship between publishers and subscribers is mediated by a topic service that receives publications, delivers data to subscribers, and enables management of entities authorized to publish or subscribe to the topic. Since XEP-006 is quite complex and has not been widely implemented, XEP-0163: Personal Eventing Protocol [22] is introduced. It includes a simplified subset of publish/subscribe functionalities to allow easier deployment of personal eventing services. Additionally, this protocol defines the semantics for broadcasting state change events associated with instant messaging and presence accounts.

III. Rich Presence Ecosystem

Presence@FER is an evolving system comprising both SIP and XMPP-based presence solutions extended to support rich presence. Fig. 1 depicts the components of this ecosystem which represents a test-bed for our rich presence applications. The ecosystem integrates an Openfire XMPP server and our own implementation of a SIP presence server (PS), as part of the IMS domain. We have developed a rich presence component that accepts presence updates in RPID format [9], and presence subscriptions defined using XPath, in accordance with RFC 4661 [11]. The rich presence component relies on Mobile Publish/Subscribe (MoPS), a content-based publish/subscribe system for filtering and disseminating presence updates optimized for mobile environments [23]. Presence@FER also includes applications that use and test our rich presence implementation, such as IM provided by the Openfire and SIP/SIMPLE presence servers, and Bedework calendar system. The Parlay X server is used as a gateway to a mobile phone network. Additionally, Presence@FER integrates virtual sources (calendar, XMPP and SIP-based IM clients), online sources (Facebook, LinkedIn, and Twitter), and physical sources (mobile phones with built-in sensors) of presence information.

In this section we describe the details of our rich presence architecture, and give an overview of applications built on top of this architecture.

A. Architecture

The components of our rich presence architecture are given in Fig. 2. The rich presence service (RPS) is implemented by two layers: 1) a rich presence layer that offers a web service interface and can be accessed directly via SOAP, and 2) publish/subscribe layer for dissemination and filtering of presence updates in accordance with rich presence subscriptions. It integrates a policy server for handling watcher and presentity policies, and accepts presence-related data from various sources. The RPS is used by existing presence servers since they deal with simple presence. The top layer of our architecture is composed of applications that use the RPS either directly, or through a presence server. Acquisition of presence-related data from virtual sources such as sensors, calendar, or Facebook, is integrated through the publish/subscribe layer.

The *publish/subscribe layer* accepts presence-related data from online, physical, and virtual sources directly, such that each data source is extended by a MoPS publisher that publishes data objects according to the MoPS publish/subscribe protocol, and subsequently filters and delivers presence updates according to existing watcher interests. Such design improves performance because presence related data does not need to pass through the two layers (rich presence and publish/subscribe layer) to be matched against subscriptions. We have decided to use two separate layers for the RPS implementation for two reasons: 1) existing content-based publish/subscribe implementations are optimized for efficient matching and dissemination of publications, and 2) as publish/subscribe implementations are distributed, a network of publish/subscribe brokers can cope with high publication rate of data sources and filter data objects close to data sources. We have decided to use an independent RPS rather than to integrate it directly into existing presence servers such that our solution is generic, and can be integrated with various presence domains and PS implementations. It can therefore also be seen as a gateway between presence domains using various protocols, while the RPID format represents the common language.

The *policy server* is needed to handle watcher and presentity policies. For example, a presentity can set different levels of presence visibility to different groups of watchers such that, e.g., his/her current location is visible only to family members, while current activity and location is visible to colleagues during office hours. Watcher, on the other side, can also define policies for receiving presence updates such that, e.g., presence notifications are blocked during an oral presentation, or presence updates from friends are blocked during office hours. As in current presence systems, policies can also be set in relation to particular watcher priority, and additionally, watchers may define preferable means (devices and applications) for notification delivery. Such policies need to be transformed to specific subscriptions/filters that are subsequently processed by the publish/subscribe layer.

For policy implementation, we are considering XACML (eXtensible Access Control Markup Language), a general purpose access control policy language ratified by the OASIS standards organization [3]. This declarative language is based on the abstract model for policy enforcement defined by IETF in RFC 2753 and RFC 3198. Besides policy syntax, XACML also defines the semantics for processing of such policies. Currently, we use policies to define access rights to presence information between watchers and presentities.

Applications can interact with the RPS either directly over SOAP, or through XMPP or SIP PSs. PSs need to be extended to interact with the RPS, e.g. the Openfire server needs a special plugin to communicate with the RPS. When using native clients that do not support the RPID format, contextual data from a RPID document can only be displayed as a textual message in addition to presentity state, while watchers can define rich presence subscriptions only directly through the RPS interface. We are also considering the implementation of rich presence clients that support the RPID format and enable watchers to define their presence subscriptions in a simple fashion.

An example interaction between a watcher, presentity, and our RPS for the SIP Presence deployment is depicted in Fig. 3. A watcher user agent (UA) sends a subscribe message to the PS. This message carries a filter in its body, and the PS forwards it to the RPS via SOAP. The RPS responds with a response message carrying subscription identifier which is further on used to identify messages for this particular subscription, and subsequently the PS sends a 200 OK message back to the watcher UA. When a presentity publishes its presence status in RPID over the RPS, if this message matches the watcher filter, a notify message carrying the RPID document is forwarded to the watcher UA. It responds with a 200 OK message. Please note that the described scenario and message format complies with the core SIP Presence protocol defined in RFC 3856 [15].

Further details regarding our RPS prototype implementation are given in Section IV.

B. Example Applications

As rich presence information is a source of context information, it can be used to implement various context-aware services. Please note that a presentity is not necessarily only human: any resource, e.g., a projector, laptop, inventory item, or pdf document, represents a presentity. We are considering the following sources of rich presence information: *user terminals and applications* that can, e.g., infer that a user is watching a movie on his/her smart phone, *sensors* located either in the environment or on user terminals that can provide user location, or *information systems*, e.g., showing that an item is on an inventory list. User calendars are also sources of presence information as they provide information regarding planned activities. However, please note that true physical presence may differ from the one which is planed, and therefore a user activity can be set to "in a meeting" when he/she arrives to the particular meeting location.

[3]`http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`

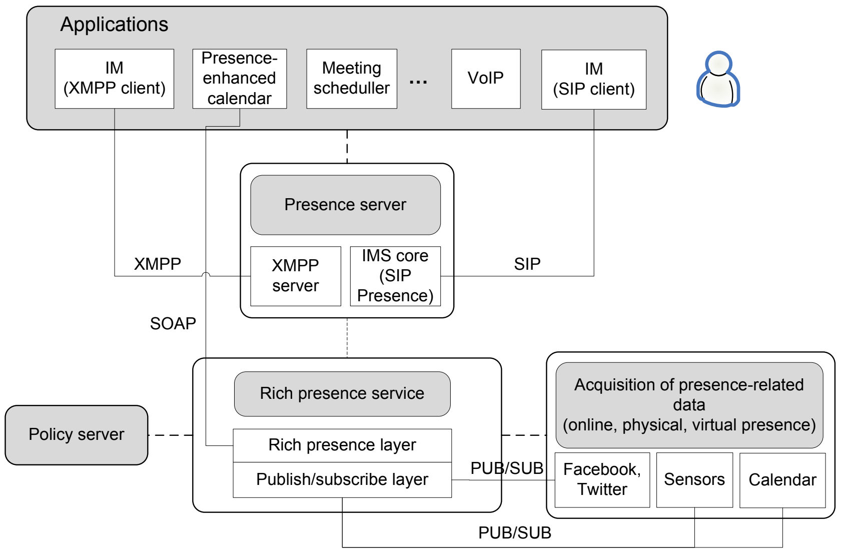

Figure 2. Rich presence architecture

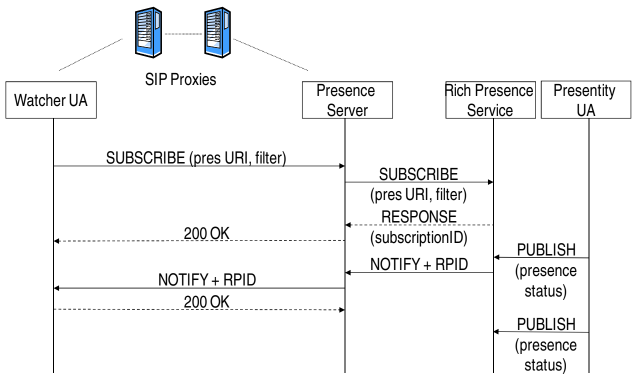
SIP Proxies
Watcher UA
Presence Server
Rich Presence Service
Presentity UA
SUBSCRIBE (pres URI, filter)
SUBSCRIBE (pres URI, filter)
RESPONSE (subscriptionID)
200 OK
PUBLISH (presence status)
NOTIFY + RPID
NOTIFY + RPID
200 OK
PUBLISH (presence status)

Figure 3. A watcher subscribes to the rich presence layer (compatible with SIP Presence)

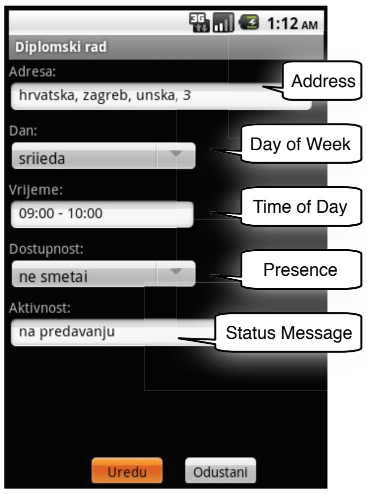

Figure 4. Policy-enabled rich presence application

Hereafter we present a few applications that have been build on top of our RPS.

The first application infers user context by using sensors built into a mobile phone with the Android OS. When the mobile phone is turned upside down such that its screen is facing the floor (recognized by a gyroscope or accelerometer identifying the swift movement), the user status is set to ”unavailable”. This application is integrated with a native XMPP client connected to an Openfire server.

The second application is the policy-enabled rich presence application developed for Android phones. A user can define policies which change its presence status such that they depend on the current time of day, location, and activity. For example, Fig. 4 shows a screen shot of a user terminal that is currently at Unska street in Zagreb, which is the address of the Faculty of EE and Computing (FER). The policy states that on Wednesdays between 9AM to 10AM if the user is located near FER, his/her status should be set to ”do not disturb” and activity to ”in lectures”.

The third application is the calendar-driven presence application that checks the calendar information of a user and changes his/her presence status in accordance with calendar events. Therefore, the status is changed implicitly. Fig. 5 shows an example when a user named ”test” changes his/her presence to ”Do Not Disturb - CALENDAR”. This implies that his/her presence will currently be triggered by calendar events. Therefore, the other user ”admin” sees his/her state as busy, because the current calendar event is entitled ”meeting”.

The last application is a message service based on online presence information. It enables a user to send a message to another user identified by a unique identifier, while the application is responsible for delivering the message using the most appropriate means. In particular, it checks the current user status at the following services: Openfire, Facebook, LinkedIn, and Twitter. If the user is online in any of the services, this service is chosen for message delivery. Since Twitter does not support presence, our application checks whether the user has twitted in the last few minutes. If the user is not available on any of the listed systems, the message is finally sent as

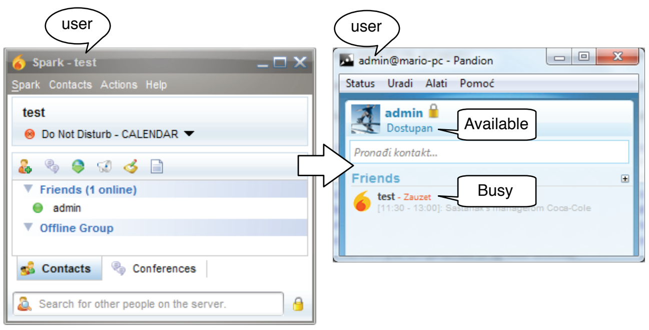

Figure 5. Calendar-driven presence

an SMS message and forwarded to the network provider through the Parlay X gateway.

IV. Rich Presence Service (RPS) Prototype Implementation

According to the architecture defined in Section III-A, the RPS prototype implementation consists of the following subsystems: a rich presence layer, publish/subscribe layer, and policy server. They are presented in Fig. 6 together with their interfaces showing the main methods offered by each subsystem.

The *rich presence layer* represents an entry point for distributing rich presence information in RPID format, and relies on the underlying publish/subscribe layer for efficient dissemination of rich presence updates. On one hand, it accepts rich presence subscriptions defined either directly by end users, or generated by presence-enabled applications that create new rich presence subscriptions on demand. On the other hand, the rich presence layer receives presence status updates in RPID format from various sources, such as IM clients extended with rich presence support, presence servers converting PIDF to RPID, or other rich presence sources.

Methods offered by the rich presence layer are the following:

- `registerWatcher`: registers a watcher with a listener object which can receive notify messages from the rich presence layer;
- `registerPresentity`: registers a presentity;
- `subscribe/unsubscribe`: creates/deletes a watcher subscription; and
- `publish`: publishes a presence update in RPID format.

Subscriptions are defined in the form of an XPath query or conjunction of XPath queries. Listing 1 shows an example rich presence subscription defined using XPath. The subscription is related to Alice identified by her pres URI `alice@tel.fer.hr` which states that a presence update should be sent when her location equals library.

Listing 1. An example XPath subscription
```
/presence[@entity=alice@tel.fer.hr]
/presence/tuple[@location-info=library]
```

Rich presence status updates are encoded using the RPID format. An example RPID publication is given in Listing 2 which defines a presence status for `alice@tel.fer.hr` as open, and her location is `library`. A RPID document is transferred to the rich presence layer using the method `publish`, and after a matching subscription is found, the same document is transported from the rich presence layer to the application layer and adequate listener using the method `notify`.

Listing 2. An example RPID publication (notification)
```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
 xmlns:dm="urn:ietf:params:xml:ns:pidf:data-model"
 xmlns:lt="urn:ietf:params:xml:ns:location-type"
 xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
 entity="alice@tel.fer.hr">
  <tuple id="bs35r9">
   <status>
    <basic>open</basic>
   </status>
   <priority>3</priority>
   <location-info>library</location-info>
   <timestamp>2011-01-27T16:49:29Z</timestamp>
  </tuple>
</presence>
```

Please note that the rich presence layer also integrates an XML database to store all subscription and publications. The database is used as a persistent storage and deals with one-time watcher queries.

The *publish/subscribe layer* is a content-based publish/-subscribe system optimized for mobile environments, and frequent disconnections of publishers and subscribers. It uses its own publish/subscribe protocol and syntax for publications and subscriptions. Subscriptions are defined as a conjunction of predicates, where each predicate is composed of an attribute, operator, and value. A publication is a simple hashtable with (attribute, value) pairs. Such definition of publications and subscriptions on the publish/subscribe layer enables the implementation of efficient algorithms for matching incoming publications to numerous subscriptions organized in trees based on the covering property between the subscriptions. Since the publish/subscribe layer offers a remote service interface, presence sources can publish presence updates directly through this interface, and thus avoid the conversion from the RPID to publish/subscribe format.

Our *policy server* is used to store policies defined by presentities granting access rights to watchers and groups of watchers. Prior to delivery of a matching RPID document to a watcher, the rich presence layer checks whether the priority associated with a watcher is sufficient (greater or equal to the priority defined in the message) to forward it further. The method `checkPolicy` is used for this purpose. The policy server stores two types of policies in the XML database. The first one contains all subscribers and their membership in user groups. The second one associates user groups to access rights.

Fig. 7 depicts a sequence diagram showing an interaction within the RPS subsystems. A watcher first registers with the rich presence layer, and provides its listener object that is used later on to receive matching presence notifications from the rich presence layer. The rich presence layer responds with a unique watcher identifier, and creates a new subscriber object associated with this watcher. When a watcher wishes to subscribe,

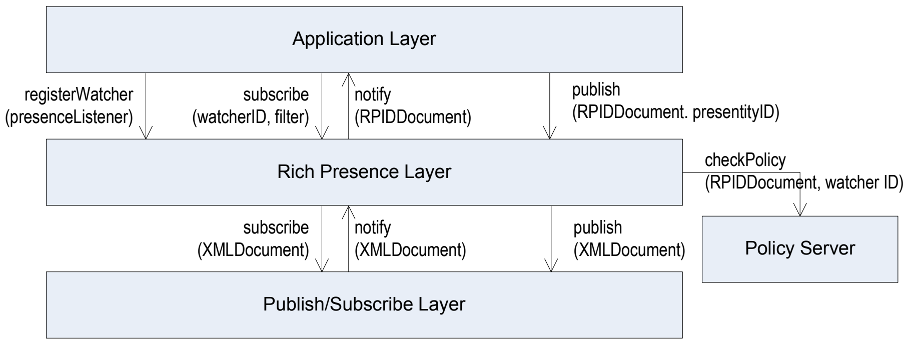

Figure 6. RPS design

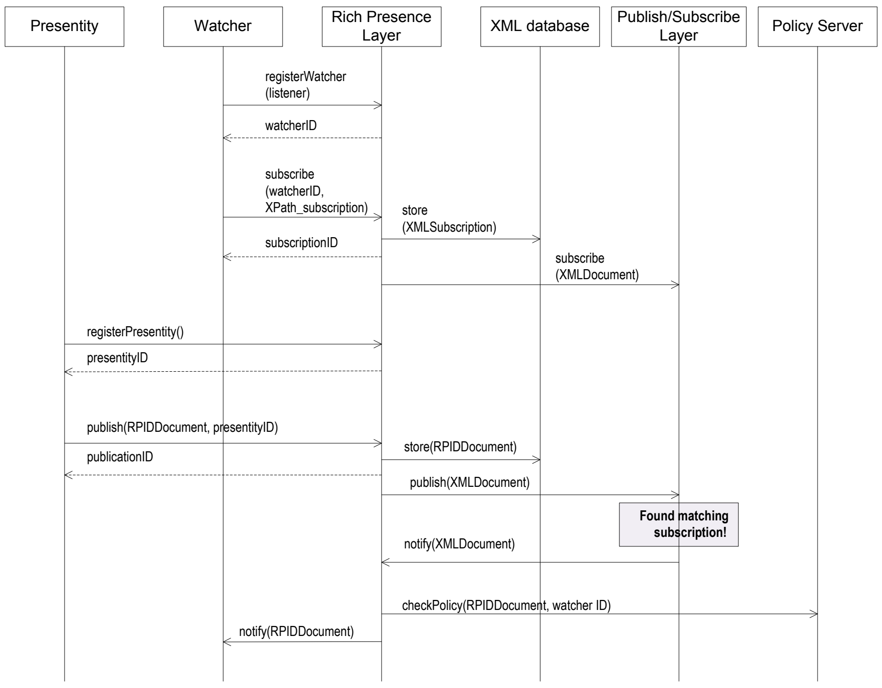

Figure 7. Sequence diagram showing an interaction within the RPS

it sends a subscription request containing an XPath subscription and its identifier. The rich presence layer stores the subscription in the form of an XML document in its XML database, converts the XPath subscription into and XML document recognized by the publish/subscribe layer, and uses the watcher subscriber object to submit the XML subscription to the publish/subscribe layer. Later on, a presentity registers with the rich presence layer and receives a unique presentity identifier. Next, it publishes its presence information in the form of an RPID document accompanied by its unique identifier. The rich presence layer stores the subscription in the form of an XML document in its XML database, converts the RPID document into and XML document modeling a publication as recognized by the publish/subscribe layer, and sends it to the publish/subscribe layer. Since the XML document matches a previously defined subscription, the publish/-subscribe layer notifies the corresponding subscriber at the rich presence layer with the matching document. After checking the policy associated with the subscriber, the rich presence layer converts the received document into the RPID format, and, finally, forwards it to the watcher listener.

V. Related Work

Presence management is currently a hot research topic, both in industry and academia. We have already surveyed a number of presence-related standards in Section II, and in this section we compare recent research results and prototype systems to Presence@FER.

A feasibility prototype of a presence service based on a RESTful web service architecture is presented in [5]. This service is used as a gateway and integrator for online sources of presence information (MSN, Yahoo, and Gmail), and is therefore comparable to our solution for integrating Facebook, LinkedIn, and Twitter into the Presence@FER. However, we focus on rich presence, while the authors in [5] use simpler PIDF format for presence implementation, and investigate whether RESTful web service architecture is adequate for the implementation of a ”universal presence service”.

Another paper by the same group of authors presents a lookup service to dynamically discover persons or physical objects within an IMS-based presence system [6]. While SIP presence and XMPP are designed such that a watcher necessarily needs to know the identifier of a presentity it wants to follow, the authors have extended the format of publish and subscribe messages such that they define abilities and affiliations which are used for finding the right presentity. However, they are not using the standardized rich presence format which also fits this purpose, while the implementation is described as query based, instead of the standard publish/subscribe interaction. Our architecture can also be used to implement such lookup service on top of RPS, while a watcher could specify continuous queries when looking for a person/object with particular characteristics and presence status.

A system which relies on a rich presence implementation is presented in [7]. It is a social networking site which enables users to collaborate and participate in common activities based on their interests and current context-aware presence status. We argue that our rich presence implementation can be used as the underlying infrastructure for developing such context-aware social network applications. Another related system is the Nomatic prototype [8] which is designed to automatically infer users' place, activity, and availability from sensors on their terminals. This system monitors user presence over time and learns their context (place, activity, mood) based on prior behavior. This solution is orthogonal to our system, and we would largely benefit if such machine learning algorithms would be integrated within Presence@FER.

VI. Conclusion

The paper presents Presence@FER, an ecosystem for management of rich presence information which includes context, e.g. user activity, location, mood, and interest. The system includes an original implementation of a rich presence service which supports the publish/subscribe interaction model, and enables efficient content-based filtering and dissemination of presence information such that watchers are enabled to define context-aware filters for presence updates, while presentities may control the disclosure of rich presence updates to watchers. We include a number of applications developed on top of the rich presence service to demonstrate its functionality. Presence@FER is an evolving system, and we plan to further extend the functionality and improve performance of the core system, while designing novel exciting rich presence applications.

References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, pp. 114–131, 2003.

[2] P. Millard, P. Saint-Andre, and R. Meijer, "XEP-0060: Publish-Subscribe (v1.13)," XMPP Standards Foundation (XSF), July 2010. [Online]. Available: http://xmpp.org/extensions/xep-0060.html

[3] P. Bellavista, A. Corradi, and L. Foschini, "IMS-based presence service with enhanced scalability and guaranteed QoS for interdomain enterprise mobility," *Wireless Commun.*, vol. 16, no. 3, June 2009.

[4] M. Hauswirth, J. Euzenat, O. Friel, K. Griffin, P. Hession, B. Jennings, T. Groza, S. Handschuh, I. P. Zarko, A. Polleres, and A. Zimmermann, "Towards consolidated presence," in *The 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2010*, Chicago, Illinois, USA, October 2010, invited paper.

[5] C. Fu, F. Belqasmi, and R. Glitho, "RESTful web services for bridging presence service across technologies and domains: an early feasibility prototype," *IEEE Commun. Mag.*, vol. 48, pp. 92–100, December 2010.

[6] Z. Zhu, F. Belqasmi, C. Fu, and R. Glitho, "A case study of a presence based end-user lookup service for the dynamic discovery of entities across technologies and domains," *IEEE Commun. Mag.*, vol. 48, pp. 82–89, November 2010.

[7] N. Banerjee, D. Chakraborty, K. Dasgupta, S. Mittal, S. Nagar, and Saguna, "R-u-in? - exploiting rich presence and converged communications for next-generation activity-oriented social networking," in *Proceedings of the 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, ser. MDM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 222–231.

[8] D. J. Patterson, X. Ding, S. J. Kaufman, K. Liu, and A. Zaldivar, "An ecosystem for learning and using sensor-driven IM status messages," *IEEE Pervasive Computing*, vol. 8, pp. 42–49, October 2009.

[9] H. Schulzrinne, V. Gurbani, P. Kyzivat, and J. Rosenberg, "RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF) (RFC 4480)," IETF, July 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4480.txt

[10] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson, "Presence Information Data Format (PIDF) (RFC 3863)," IETF, August 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3863.txt

[11] H. Khartabil, E. Leppanen, M. Lonnfors, and J. Costa-Requena, "An Extensible Markup Language (XML)-Based Format for Event Notification Filtering," IETF, September 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4661.txt

[12] M. Day, J. Rosenberg, and H. Sugano, "A Model for Presence and Instant Messaging (RFC 2778)," IETF, February 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2778.txt

[13] J. Peterson, "Common Profile for Presence (CPP) (RFC 3859)," IETF, August 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3859.txt

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol (RFC 3261)," IETF, June 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[15] J. Rosenberg, "A Presence Event Package for the Session Initiation Protocol (SIP) (RFC 3856)," IETF, August 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3856.txt

[16] H. Schulzrinne, "Timed Presence Extensions to the Presence Information Data Format (PIDF) to Indicate Status Information for Past and Future Time Intervals (RFC 4481)," IETF, July 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4481.txt

[17] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core (RFC 3920)," IETF, October 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3920.txt

[18] ——, "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence (RFC 3921)," IETF, October 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3921.txt

[19] ——, "XMPP: lessons learned from ten years of XML messaging," *Comm. Mag.*, vol. 47, no. 4, pp. 92–96, 2009.

[20] Open Mobile Alliance, "OMA Presence SIMPLE Specification, v.1.1: OMA specification of the presence enabler," June 2008. [Online]. Available: http://www.openmobilealliance.org/Technical/release_program/presence_simple_v1_1.aspx

[21] H. Khartabil, E. Leppanen, M. Lonnfors, and J. Costa-Requena, "Functional Description of Event Notification Filtering," IETF, September 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4660.txt

[22] P. Saint-Andre and K. Smith, "XEP-0163: Personal Eventing Protocol (v1.2)," XMPP Standards Foundation (XSF), July 2010. [Online]. Available: http://xmpp.org/extensions/xep-0163.html

[23] I. Podnar and I. Lovrek, "Supporting mobility with persistent notifications in publish/subscribe systems," in *In Proc. of the 3rd Int. Workshop on Distributed EventBased Systems*. ACM Press, 2004, pp. 80–85.