

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMA THESIS no. 153

DIPLOMSKI RAD br. 153

**CONTEXT – AWARE FILTERING AND
DISSEMINATION OF RICH PRESENCE
INFORMATION**

**KONTEKSTNO – SVJESNO FILTRIRANJE I
RAZAŠILJANJE INFORMACIJA BOGATE
PRISUTNOSTI**

Aleksandar Antonić

Zagreb, June 2011.

Zagreb, lipanj 2011.

Acknowledgements

I would like to thank my advisors Prof. Ivana Podnar Žarko and Prof. Manfred Hauswirth for their guidance and provided assistance in making this thesis.

Zagreb, 18. veljače 2011.

DIPLOMSKI ZADATAK br. 153

Pristupnik: **Aleksandar AntoniĆ**
Studij: Informacijska i komunikacijska tehnologija
Profil: Telekomunikacije i informatika

Zadatak: **Context-aware filtering and dissemination of rich presence information**

Opis zadatka:

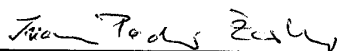
Presence information, expressing user willingness and ability to communicate, represents an essential prerequisite for real-time communications. Presence service enables users (watchers) to subscribe to presence information generated by their contacts (presentities), and to receive their presence updates in real-time. However, existing solutions typically ship all generated presence updates, without taking into account watcher context and actual interest. The rich presence layer and the underlying publish/subscribe system that have been designed and developed at the Department of telecommunications, FER, enable filtering and dissemination of rich presence information in accordance with user-defined subscriptions and policies. Your task is to integrate the rich presence layer with a prototype system where.deri.ie for tracking physical presence from sensor data, which has been developed at DERI, National University of Ireland, Galway. You should design and implement a matching algorithm for the publish/subscribe system which is optimized for the particular type of presence data coming from where.deri.ie. In addition, you should provide a complete presence solution optimized for efficient filtering and dissemination of presence updates, and perform extensive experiments to investigate system performance.

The work will be performed at the Department of telecommunications, FER, and DERI, NUI, Galway during a student internship from April 1st until June 30th, 2011. The work performed in DERI will be conducted under the supervision on Prof. Manfred Hauswirth, and the thesis will be written in English.

Zadatak uruĉen pristupniku: 25. veljaĉe 2011.

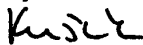
Rok za predaju rada: 10. lipnja 2011.

Mentor:



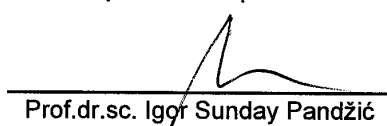
Doc.dr.sc. Ivana Podnar Žarko

Djelovođa:



Doc.dr.sc. Mario Kušek

Predsjednik odbora za
diplomski rad profila:



Prof.dr.sc. Igor Sunday Pandžić

The work presented in this diploma thesis has been performed at the Department of Telecommunications of the Faculty of Electrical Engineering and Computing, University of Zagreb, and Digital Enterprise Research Institute (DERI), National University of Ireland, Galway during an internship from April 1st until June 30th, 2011. The work performed in DERI has been conducted under the supervision of Prof. Manfred Hauswirth

Contents

Introduction	1
1. A Model for Presence	2
1.1. Communication between the entities.....	3
1.2. Message structure	5
1.2.1. Publication/Notification	5
1.2.2. Subscription.....	6
2. SIP Presence	7
2.1. Architecture	7
2.2. Communication and protocol	9
2.3. Presence Information Data Format – PIDF.....	12
3. Extensible Messaging and Presence Protocol	15
3.1. Architecture	15
3.2. Communication and protocol	16
3.2.1. XML stanzas.....	17
3.2.2. Subscription process	17
3.2.3. Publishing process	19
3.2.4. Notification process.....	19
4. Rich presence service for physical presence	21
4.1. Architecture	21
4.2. where.der.i.e	24
4.3. Implementation of the RPS and integration with where.der.i.e	26
5. Supported Presence Subscriptions.....	32
5.1. Simple subscription	32
5.1.1. Simple presence service.....	32
5.1.2. Geo–fence service.....	34

5.1.3.	Virtual secretary	34
5.2.	Parallel subscription	36
5.2.1.	Meeting scheduler.....	36
5.2.2.	Sequence subscription	38
	Conclusion.....	40
	References	41
	Summary.....	43
	Sažetak.....	44

Introduction

Presence as a service is widely spread in today's world of communications (telephony, instant messaging, e-mail, etc), and is often referred to as the dial tone of the 21st century. Currently there are two competing protocol suites for presence: SIP presence and eXtensible Messaging and Presence Protocol (XMPP). Although both protocols are widely deployed, existing solutions still face a number of challenges: existing presence solutions typically ship all generated presence updates from presentities to watchers, without taking into account watcher context, while it is difficult to control the disclosure of sensitive personal presence information. This generates a large number of messages that can use up client battery power rather quickly, and introduces serious scalability problems within the core network. Moreover, existing solutions do not include various aspects of presence information from physical, online, and virtual presence sources that would contribute to provide a usable flow of information within different communities (business partners, social groups, family, etc) while retaining the simplicity of use for end users.

This thesis describes a rich presence service as an integrating solution for presence which is compatible with existing protocols (SIP Presence and XMPP). It enables 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. The rich presence service can be used to extend existing services by integrating independent data sources and filtering their content in accordance with current user interests and needs.

The thesis is structured in the following way. Chapter 1 provides an overview of the basic model for presence services. Chapter 2 and Chapter 3 describe currently the two most popular solutions for presence: SIP Presence and XMPP. Chapter 4 describes the rich presence service, followed by a description of supported subscriptions and services in Chapter 5. Chapter 6 concludes the thesis.

1. A model for Presence

Presence information (abbreviated as presence) is defined as user willingness and ability to communicate with other users across a set of devices and tools. Presence service is a service which receives, stores, and disseminates presence information to all interested parties. Presence service has emerged in conjunction with instant messaging systems, but today is regarded as an independent service.

A model for presence, shown in Figure 1.1, defines all entities involved in the exchange of presence information, and describes the protocols and messages used in a system implementing a presence service [3]. Presence service allows users to subscribe to presence updates generated by their contacts, and to be notified of changes regarding their presence information (e.g. available, busy, do not disturb).

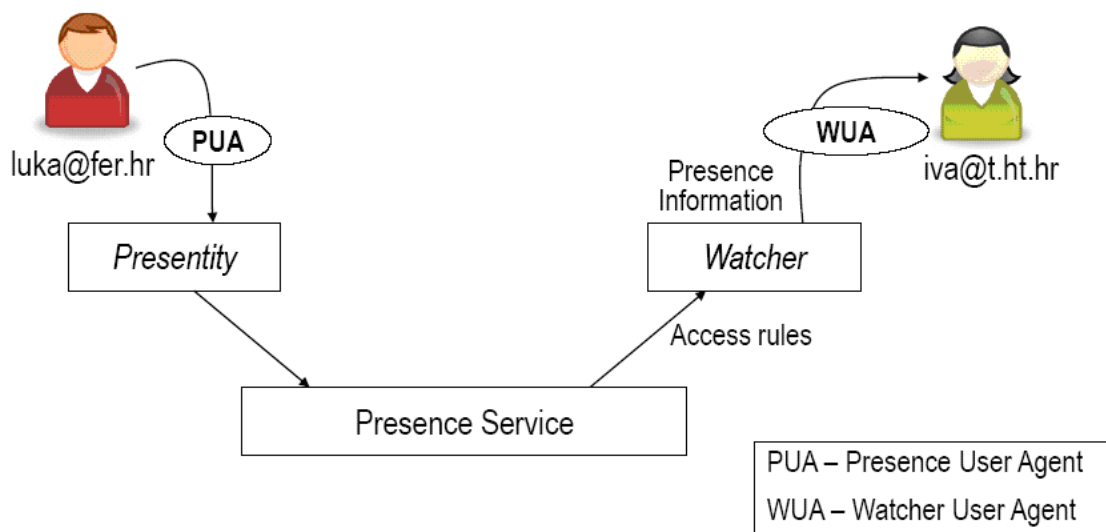


Figure 1.1 Presence Service model

A presence service entity accepts all presence updates, stores it and distributes to subscribed users. It also keeps records about watchers and their activities (subscriptions to presence information). Prior to disseminating any presence information, the service checks the watcher's access rights to verify whether the watcher is allowed to receive presence updates from a particular presentity.

A presentity provides presence information about users (humans) or resources (laptop, projector) to the Presence Service (PS). It is supplied with data (presence information) from Presence User Agents.

A Presence User Agent (PUA) is an entity that generates presence-related information such as status, location, contact means, etc. Usually it is a client application on a user's computer or mobile phone, but it can also be an object or resource that represents a presence information producer, e.g. a sensor generating presence-related contextual information.

A watcher is an entity that requests presence information about presentities from a presence service. Requests are defined as one-time queries or continuous user subscriptions. Accordingly, there are two types of watchers: a fetcher explicitly asks for the current presence status of one or more presentities using one time queries, and a subscriber specifies subscriptions or continuous queries that ask the PS to notify it immediately about any changes of presence information generated by one or more presentities. This request is defined as an active subscription stored by the PS. Watcher serves as an intermediary between a PS and Watcher User Agent.

A Watcher User Agent (WUA) is an entity that enables a watcher to specify subscriptions to presence updates and displays presence notifications to a user. Usually it is a client application on a user's computer or mobile phone, or part of another application (e.g. a meeting scheduler, calendar, etc.).

1.1. Communication between the entities

The communication between a PUA and presentity, or between a WUA and watcher uses some of the well known communications protocols (e.g. SIP, XMPP or HTTP). The PS communicates with presentities and watchers using the presence protocol (as shown in Figure 1.2).

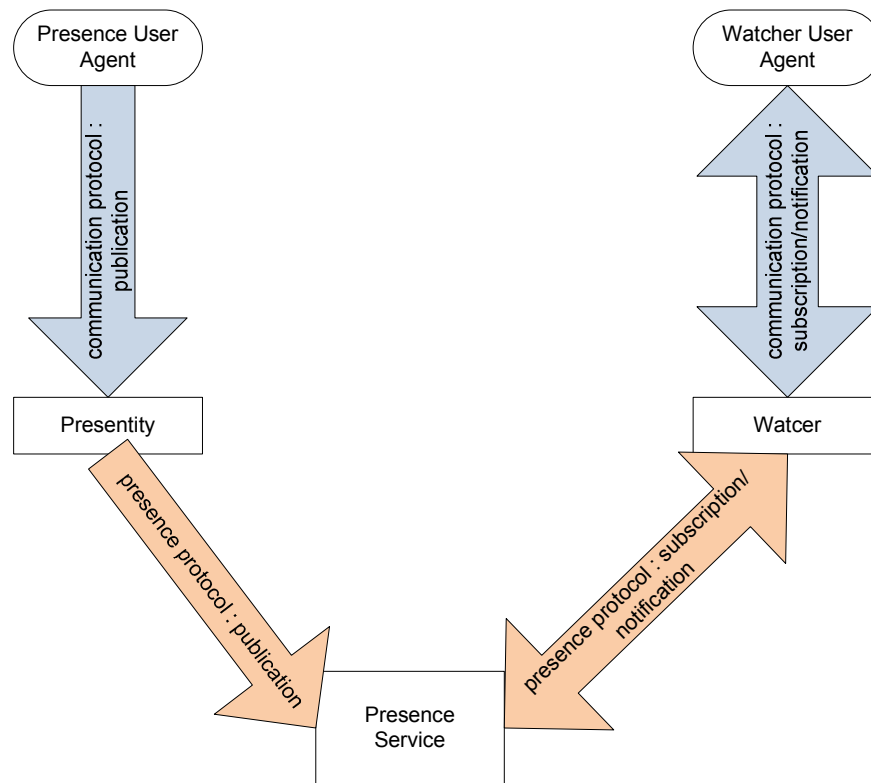


Figure 1.2 Presence Service communication

The above mentioned protocols, the well known communications protocols and presence protocol, carry three types of messages: subscription, publication and notification. A subscription is the message which expresses continuous request for presence information of some presentity on behalf of a user. A publication is a message that carries new, fresh presence information about a user or resource. A notification is a message that delivers presence information generated by a presentity of interest to the watcher user agent.

Since presence uses the publish/subscribe style of communication, it does not conform to a strict flow of messages between the entities which is a characteristic of the request–response communication mechanism. On one hand, subscriptions and publications are generated in an ad-hoc fashion independently of each other. On the other hand, notifications are produced by the PS only in cases of a matching event. The matching event is an event when a subscription previously defined by a watcher, which is defined by a presence URI (pres URI) identifying a presentity of interest, overlaps with the presence information generated by the presentity. Additionally, the watcher needs to be granted with a permission to receive presence information from the presentity of interest. Currently there are two widespread presence protocol implementations, SIP Presence and XMPP, described in detail in Chapters 2 and 3.

1.2. Message structure

1.2.1. Publication/Notification

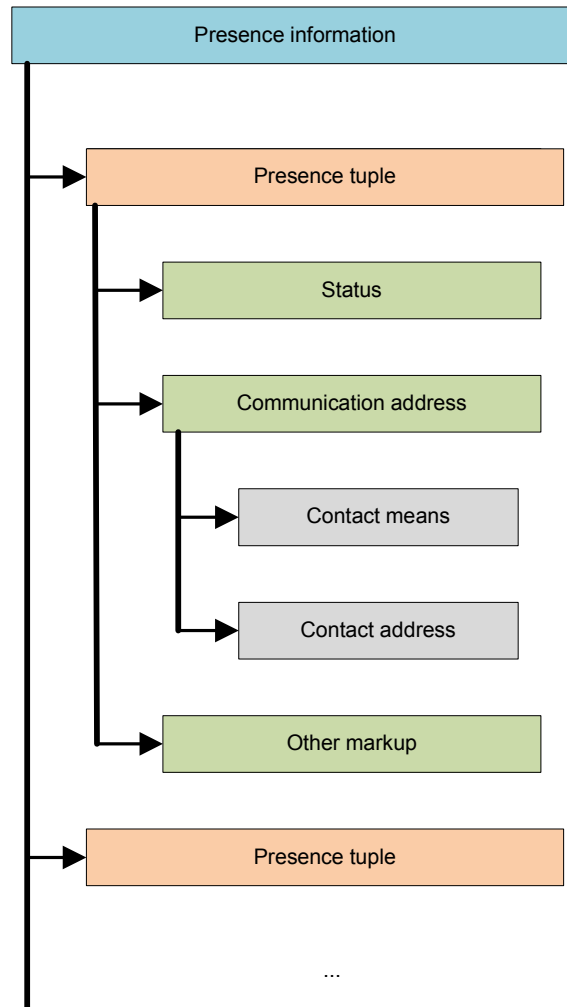


Figure 1.3 The structure of presence information

Both publication and notification messages have the same structure because they both carry presence updates about a presentity. Figure 1.3 shows the structure of presence information (defined by RFC 2778 [3]) which consists of one or more presence tuples. Each presence tuple describes a single communication point associated with a presentity, and consists of a status and optional communication address and presence markup. A status carries the basic information about a presentity, and usually describes the user's willingness for communication. It primary defines a user status as being either open or closed, and expresses the specific communication status related to a presentities communication point, e.g. online, offline, busy, not available. A communication address includes the following fields: communication means and a contact address.

Communication means indicates a method whereby communication can take place (e.g. instant messaging, e-mail, phone), and contact address is an identifier through which a user or resource can be reached. Presence markup includes any additional information included in the presence information of the presentity (e.g. location, mood, time).

1.2.2. Subscription

The basic presence model and RFC 3859 [4] define a subscription message as a subscription to a pres URI. Each subscription message must contain the following attributes: watcher, target, duration, subscription ID and TransID. A watcher attribute identifies a subscriber (via a user's pres URI), while a target attribute identifies a presentity (also via a pres URI). A duration specifies the maximum number of seconds that the subscription is active. A subscription ID is a unique identifier used for unsubscribing, and TransID is a message identifier that is used in a response message generated as a request to a subscription message.

RFC 4661 [5] describes an extension for a subscribe message. The subscribe message is an XML document containing the <filter-set> element as root element. The <filter-set> element may contain one <ns-bindings> element and one or more <filter> elements. The <ns-bindings> element is used to bind namespaces to local prefixes used in expressions that select elements or attributes in the <filter> element. The <filter> element is used to specify the content of an individual subscription. Each <filter> element has the uri attribute containing value of a pres URI of the user of interest. The <what> element is used to specify the content to be delivered to the user, and the <trigger> element identifies changes that a resource has to encounter before the content is delivered to the subscriber.

2. SIP Presence

SIP Presence is also known as Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (SIMPLE) developed by the SIMPLE Working Group. It is an instant messaging and a presence protocol suite based on the Session Initiation Protocol (SIP). SIP architecture and protocols are reused for presence because SIP location services already maintain certain user-related presence information in the form of user registrations. SIP networks are capable of routing requests from any network to the server that holds the registration state for another user.

2.1. Architecture

The architecture [6] (as shown in Figure 2.1) is similar to the generic model described in Chapter 1. The central presence component is the PS, which includes three entities: Register, Edge Presence Server, and Presence Agent. In SIP Presence, a watcher entity is embedded in a WUA, and presentitiy is embedded in a PUA.

A presentity may use multiple PUAs, one for every device that can produce new presence information (e.g. mobile phone and laptop). Each PUA independently generates a part (i.e. tuple) of the overall presence information, and pushes it into the presence system.

WUA manipulates user's subscriptions and displays to the user all incoming notifications. A user may apply multiple WUAs. The user can define subscriptions on any WUA that sends updates into the presence system, but notifications will be displayed to the user only on the a single (probably the last active) WUA.

WUA and PUA are implemented in client applications on user's communication resource (e.g. mobile phone, computer). Client application usually include implementations of both entities.

The core of the SIP Presence system is implemented by a Presence Agent. It is a SIP server which is capable of receiving and processing subscribe messages and generating notifications of changes in presence state. Presence Agent receives and processes publication messages and therefore maintains the knowledge about the presence statuses for all registered presentities.

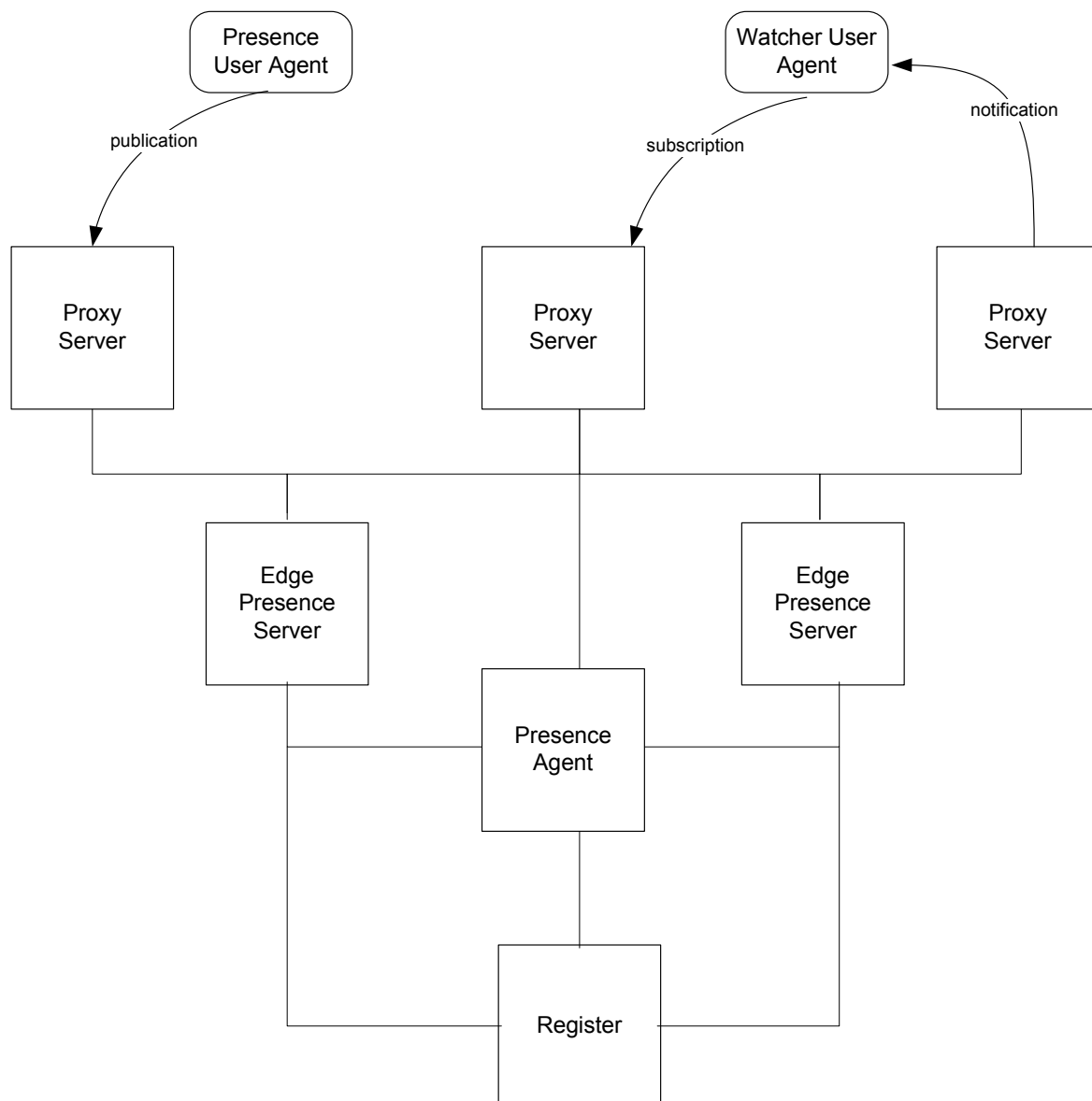


Figure 2.1 SIP Presence architecture

An Edge Presence Server is a Presence Agent that is co-located with a Presence User Agent. It is aware of the presentity's presence information and is therefore capable to process (part) of subscriptions in the system and thus reduce load on the Presence Agent. A Presence Agent and Edge Presence Server constitute a Presence Server.

A Register is the server that receives and processes registration messages and records every log in to the system from a presentity or watcher. It also keeps the data about the currently active WUA (this information is required for the delivery of notifications). The register in some cases also stores user's (partial) presence information.

Proxy Server routes messages to the appropriate server, Presence Agent or corresponding Edge Presence Server.

2.2. Communication and protocol

When a WUA (on behalf of a user) wants to receive presence information from some other user (a presentity), it creates a subscribe message where the presentity is identified in the Request-URI, using a SIP URI or pres URI. A proxy carries this message to the Presence Server (Presence Agent or Edge Presence Server with the most recent update of presence information). The Presence Server first authenticates and then authorizes the subscription. If the subscription is authorized, a 200 OK response is returned. A 403, 603 (rejected) or 202 (pending) response is returned for non authorized subscriptions. In both cases the Presence Server also sends a notification message containing requested user's presence information. In case of 202, a response notification message is also sent which always indicates that a user of interest is offline (not able to communicate).

The subscription persists for a duration stated in an Expires header field of the subscription message. It is necessary to refresh the subscription before its expiration, if the user is still interested in the presence information.

The unsubscription process is similar to the subscription process. WUA sends a subscribe message which is similar to the one previously sent when defining the original subscription: the only difference is that the Expires header field has a value set to zero. The Presence Server also replies with a status message (200 OK or 202) and after that sends a notification message with the current presence information for the indicated user. This feature also supports one-time queries for presence information. In both cases, either when used for unsubscriptions or one-time querying, the Presence Server authenticates and authorizes requests prior to processing them.

A Presence Server may send a notification message to a WUA at any time, usually when the presence information of a presentity changes (i.e. after the Presence Server receives a new publication message).

A notification message contains the presence information in its body in the Presence Information Data Format (PIDF). It is common to generate notification messages with complete presence information. Extensions enable a watcher to request notifications containing only changes in presence information, rather than complete presence information. If the resource is not in a meaningful state, the Presence Server can send notification message without a body (i.e. presence information).

For reasons of privacy, it is necessary to encrypt the contents of the notifications. This can be accomplished using the S/MIME format¹.

After the change of a presence status, a new publication message is created by a Presence User Agent (on behalf of a user) and sent to the Presence Server. The Presence Server sets a *soft state*² for a designated presentity and sends a 200 OK response as a confirmation. Presence information is contained in the body of a publication message, written in PIDF.

Before the expiry time specified in the Expires header field is reached, a Presence User Agent must refresh the presence information if it is still valid. The process of presence information refreshing requires that a PUA sends a new publication message. Such a message has an empty body (because there is no change in presence information) but has a value in SIP-If-Match field to indicate the refresh operation and new value in the Expires header field. After a successful refresh process, the PS responds with 200 OK.

For immediate removal of presence information, a PUA creates a publish message with an Expires header field set as zero. This message also has no body, and has a value in SIP-If-Match field to indicate which presence information to be removed from the Presence Server.

SIP Presence allows partial modification of presence information. For this operation, PUA creates and sends a publish message with SIP-If-Matched field and body that contains only changed presence information. Presence Server responds with a 200 OK message for successful partial update of presence information.

¹ Secure/Multipurpose Internet Mail Extensions

² *soft state* is a state of the presentity when there is active presence information with defined lifetime; persistent presence information sets the *hard state* of the presentity

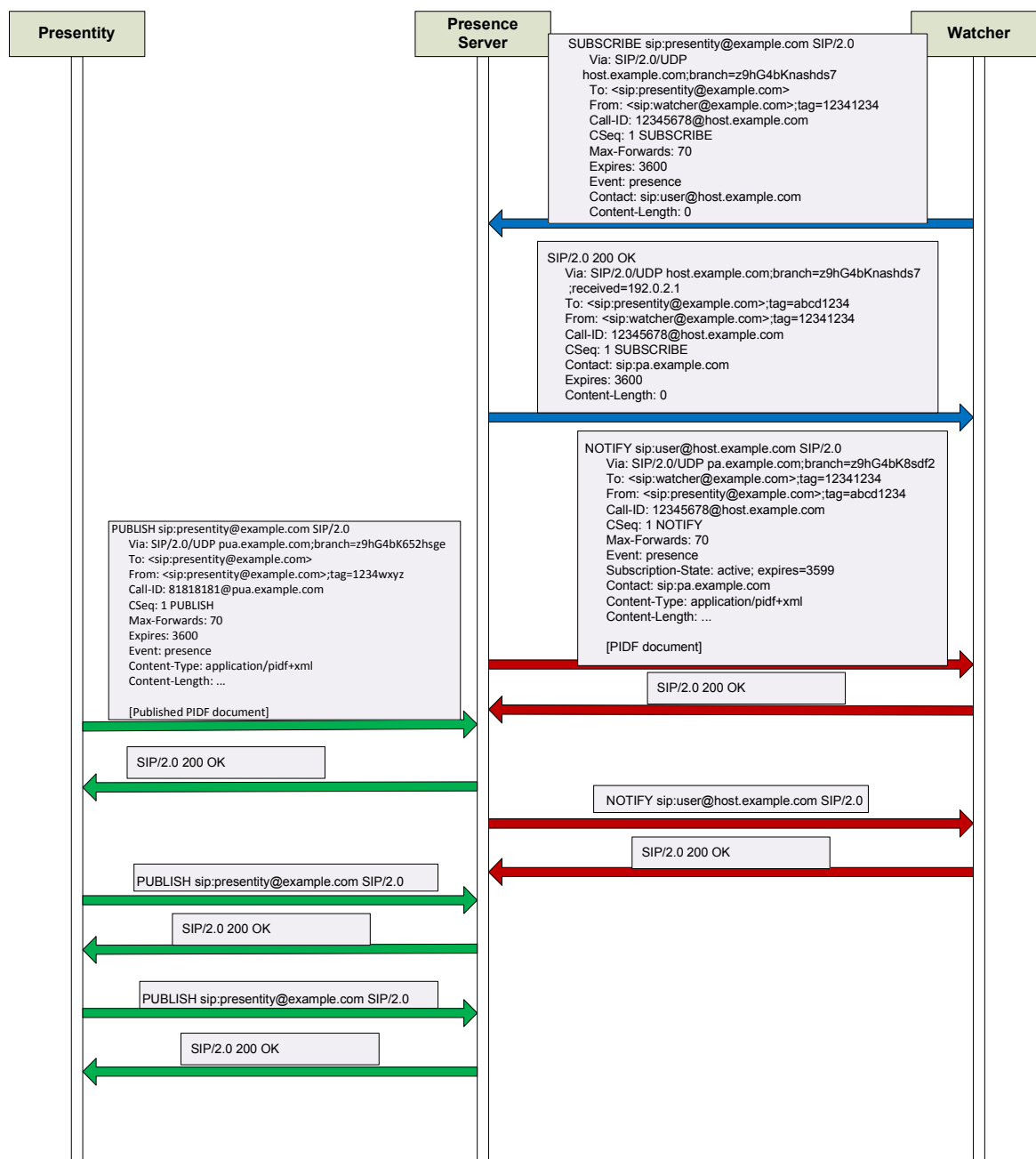


Figure 2.2 Message flow

Figure 2.2 shows a sample message flow in SIP Presence. A watcher subscribes to presence information of a presentity. After a successful subscription process, a Presence Server responds with a 200 OK message and includes the current presence status of the addressed presentity. Subsequently, when the presentity publishes a new presence message, the Presence Server processes it and also responds with a 200 OK message. The Presence Server also checks whether there is a subscription that matches the newly published information. This results with a notification which is sent to the watcher with a

matching presence subscription. The final message is used to refresh the current presence information of the presentity and it does not trigger a notification process.

The SIP protocol is used for communication between all presence entities and therefore all exchanged messages include a SIP header. The first line of a message defines the required operation. SUBSCRIBE identifies a subscribe message which is followed by a presentity URI and protocol designation. The following field To contains the presentity URI (not a Presence Server URI), and the From field contains the watcher URI that defines a new subscription. The Expires field sets the duration of the subscription (time in seconds from receiving subscription message).

A notification message is identified by a NOTIFY field followed by a URI of the intended watcher (subscriber) and protocol designation [7]. The To field also contains a watcher's URI while the From field contains the presentity URI (not a Presence Server URI). The body of the message contains the whole presence information in PIDF.

For publication message is used PUBLISH method [8], followed by presentity URI and protocol designation. Fields To and From fields have presentity URI. Expires field sets the duration of publication (time in seconds). Presence information is in the body of message, written in PIDF.

Responses to messages SUBSCRIBE and NOTIFY start with the protocol designation followed by status message. All header fields are the same as in the request message.

2.3. Presence Information Data Format – PIDF

SIP Presence requests that presence information is written in the Presence Information Data Format³ (PIDF) [9]. PIDF was developed from recommendations for the basic model of presence services⁴. PIDF supports all options mentioned in section 1.2.1 of this thesis, but also expands the presence information with priorities of contact addresses (to give preference to some communication means) and adds a timestamp to a presence update. PIDF encodes presence information in eXtensible Markup Language (XML). The

³ Also is used PIDF extension RPID (Rich Presence Information Data Format) which allows a more detailed description of presence.

⁴ RFC 2778

designation for the PIDF presence information carried in the SIP message body is application/pidf+xml.

All PIDF presence information must start with an XML declaration. The root element is the <presence> element. Attribute fields contain namespace declarations used in PIDF elements and a mandatory 'entity' attribute (URI of the presentity). The <presence> element contains any number of <tuple> elements, which includes any number of <node> elements with any number of optional extension elements from other namespaces. A <tuple> element consists of a mandatory <status> element, followed by any number of optional extension elements (including from other namespaces), followed by an optional <contact> element any number of <note> elements, and an optional <timestamp> element. Tuples are segments of presence information. The attribute Id is mandatory, and is used to distinguish a tuple from other tuples in the same PIDF document. The <status> element contains a single <basic> element, followed by any number of optional extension elements. The <basic> element can declare one of the following states: "open", which indicates a user willingness to communicate, or "closed" otherwise. The <contact> element contains a URL of the contact address. Each tuple has a priority attribute, whose value means a relative priority of this contact address over the others. A priority value is a decimal number between 0 and 1. The <note> element contains a string value, which is usually used as a human readable comment. The <timestamp> element contains a string indicating the date and time of status change of this tuple.

Figure 2.3 shows an example presence information in PIDF stating that the user can be reached via an instant messaging address, or e-mail. Status busy and lower priority on instant messenger contact mean indicates that user prefers contact via e-mail.

```

<?xml version="1.0" encoding="UTF-8"?>
  <presence xmlns="urn:ietf:params:xml:ns:pidf"
    xmlns:im="urn:ietf:params:xml:ns:pidf:im"
    xmlns:myex="http://id.example.com/presence/"
    entity="pres:someone@example.com">
    <tuple id="bs35r9">
      <status>
        <basic>open</basic>
        <im:im>busy</im:im>
        <myex:location>home</myex:location>
      </status>
      <contact priority="0.8">im:someone@mobilecarrier.net</contact>
      <note xml:lang="en">Don't Disturb Please!</note>
      <note xml:lang="fr">Ne derangez pas, s'il vous plait</note>
      <timestamp>2001-10-27T16:49:29Z</timestamp>
    </tuple>
    <tuple id="eg92n8">
      <status>
        <basic>open</basic>
      </status>
      <contact priority="1.0">mailto:someone@example.com</contact>
      <note>I'll be in Tokyo next week</note>
    </tuple>
  </presence>

```

Figure 2.3 Presence information in PIDF

3. Extensible Messaging and Presence Protocol

The Extensible Messaging and Presence Protocol (XMPP) is an application profile of the Extensible Markup Language (XML) that enables real-time exchange of data between two or more network entities. The protocol was originally named Jabber, developed by the Jabber open source community and subsequently supported by the XMPP Working Group. Many large companies (e.g., Google, Cisco, Facebook) support XMPP in their services which lead to a large user base.

3.1. Architecture

An XMPP Presence System is typically implemented using a distributed client – server architecture as depicted in Figure 3.1, wherein a client needs to connect to a server in order to gain access to the network. XMPP uses globally unique addresses (based on Domain Name System – DNS) in order to route and deliver messages over the network [10]. All XMPP entities are addressable on the network. The server's address is of the form <domainpart> (e.g. <pres.example.com>). Each user has a unique personal address formed as <localpart@domainpart> (e.g. <user@pres.example.com>). This address is also known as the “bare JID⁵”. When a user connects to the network using a client application on one of his/her resources, he uses the address formed as <localpart@domainpart/resourcepart> (e.g. <user@pres.example.com/mobile>).

A client is an entity that establishes an XML stream with a server and then completes resource binding in order to enable delivery of XML stanzas between the server and the client over the negotiated stream. XMPP allows simultaneous connection between multiple clients and server using the same registered account, where each client is differentiated by the resource part of an XMPP address.

A server is an entity that manages XML streams with connected clients and servers. The also server delivers XML stanzas to clients. Additional responsibilities can include the

⁵ JID = Jabber ID

storage of data that is used by clients and hosting for add-on services that also use XMPP as the basis for communication.

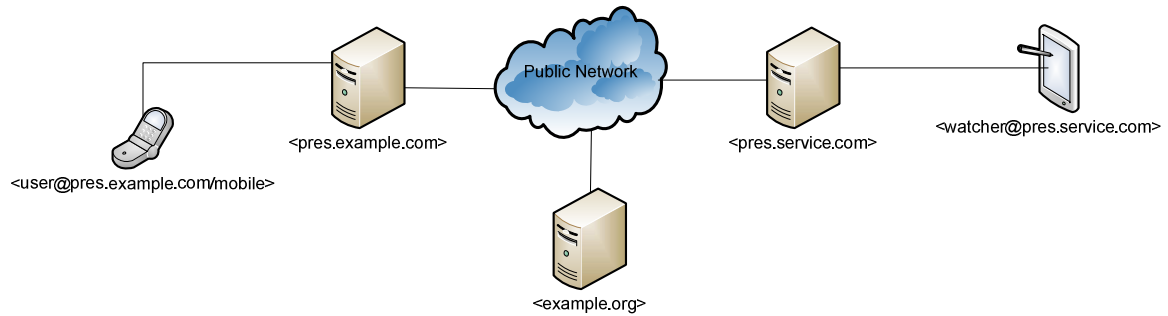


Figure 3.1 XMPP architecture

3.2. Communication and protocol

The communication between the entities in an XMPP network (source and destination) is carried out following the listed steps:

1. Determine the IP address and port number of the destination to which the source wants to connect to.
2. Open a Transmission Control Protocol (TCP) connection to the destination.
3. Open an XML stream over the TCP connection.
4. Negotiate Transport Layer Security (TLS) for encryption.
5. Authenticate entities in communication.
6. Bind a resource to the stream.
7. Exchange XML stanzas.
8. Close the XML stream.
9. Close the TCP connection.

The source entity first uses DNS to obtain the IP address and port of the destination entity. After receiving a response, the source opens a TCP connection and establishes an XML stream with the destination. It is recommended to encrypt the channel communication using TLS. The Simple Authentication and Security Layer (SASL) is used for entity authentication. This completes the initialization of the connection. If at least one entity is a client, a resource (i.e. communication mean, e.g. mobile phone, laptop) is bind to the

stream (in server to server communication this step is skipped). The exchange of XML stanzas fulfills the purpose of establishing the communication. An unlimited number of XML stanzas can be exchanged during one session. In the end of communication, the entities close the XML stream and the TCP connection.

Further on we focus on the exchange of XML stanzas which is used for the exchange of presence information.

3.2.1. XML stanzas

The basic message in the communication using the XMPP protocol is an XML stanza. There are three kinds of stanzas: <message/>, <presence/> and <iq/>. The <message/> stanza is used when one entity pushes information to another entity. The <presence/> stanza is used for disseminating and managing presence information following the publish/subscribe communication style implemented over XMPP. The <iq⁶> stanza serves as a control mechanism, using request – response style of communication. For additional functionality that extends the basic syntax of stanza, XMPP uses XML namespaces [13].

Attributes ‘to’, ‘from’, ‘id’, ‘type’ and ‘xml:lang’ are common to all stanzas. The ‘to’ attribute specifies the JID of the intended recipient for the stanza, and the ‘from’ attribute specifies the JID of the sender. The ‘iq’ attribute is used to track response or error stanza that it might receive from another entity. The ‘type’ attribute specifies the purpose or context of the stanza. The ‘xml:lang’ attribute specifies the default language of human readable XML data, is often omitted.

3.2.2. Subscription process

Presence information is disseminated only to other entities that a user has previously approved [11]. In XMPP, a subscription lasts until an entity unsubscribes or a user invokes the previously granted subscription approval. For managing subscriptions, entities are using presence XML stanzas with attributes: subscribe, unsubscribe, subscribed and unsubscribed.

Figure 3.2 shows an example subscription process using XMPP. A subscriber’s client generates a subscription request by sending a presence stanza of the type subscribe and a

⁶ iq = Info/Query

URL address of a user of interest in the attribute 'to'. This request represents a request for authorization from the user that he/she is interested in. The server expands the subscription request with the attribute 'from', and routes the XML stanza toward the user of interest. After the server forwards this request, it also sends a response in the form of an iq stanza to the subscriber. After the client application from the user of interest receives a subscription request, it presents the request to the user (or replies to it automatically following the explicit rules predefined by the user). A stanza of the type subscribed is used when an approval is granted, while a stanza of the type unsubscribed is used otherwise. Further on, the server expands the presence stanza with an attribute 'from', and routes the message to the subscriber finishing the process with a response to the user in the form of an iq stanza. Finally, the server needs to deliver the presence information of the successfully subscribed user to all subscriber devices.

XMPP mandates that each user has a subscription to his own presence information. The auto-subscribe process is managed by the user's server.

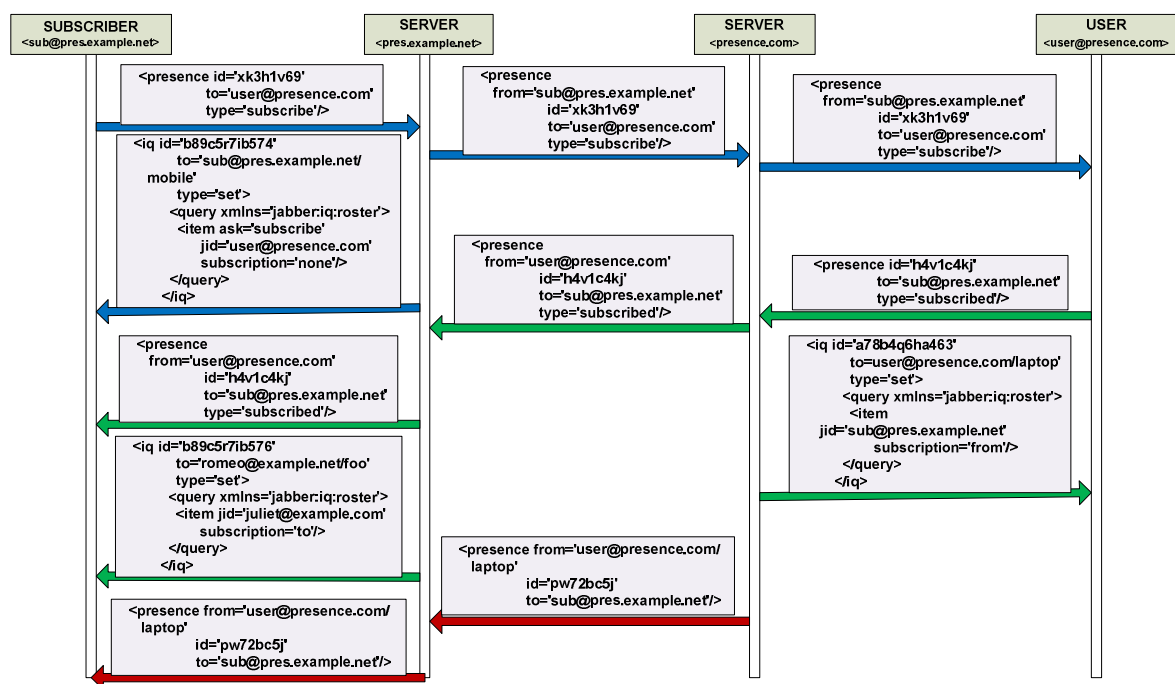


Figure 3.2 Subscription process using XMPP

When cancelling a subscription, a user can at any time send a presence stanza of the type unsubscribe and repeat the previously described process with a message of the type unsubscribe.

3.2.3. Publishing process

Publishing process using XMPP is shown in Figure 3.3. When publishing a new presence update, a user sends an empty presence stanza (also called the initial presence). The user's server sends the initial presence to all subscribers, extending the stanza with attributes 'from' and 'to'. It also sends this initial presence to the user that published it because of the auto-subscribe feature. After sending the initial presence stanza, the user can send the full presence information. This is done in the same fashion as with the initial presence. The user's server disseminates presence information to all subscribers, extending the presence stanza with attributes 'from' and 'to'. For ending the "presence session" (i.e. going offline), the user sends a presence stanza of the type unavailable. Status message may or may not be included.

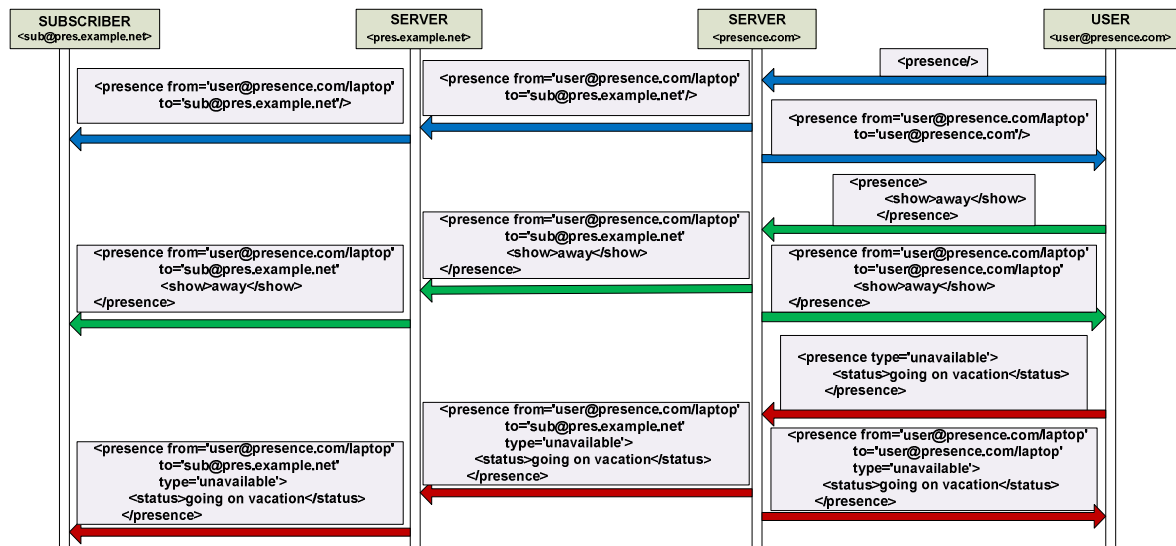


Figure 3.3 Publishing process using XMPP

If the user wants to send his/her presence information to another user that is not subscribed to his/her presence, he/she can do this by sending a presence stanza with a 'to' attribute that contains the JID of the intended recipient. The servers will route the stanza through the network to the intended user.

3.2.4. Notification process

Figure 3.4 shows notification process using XMPP. When a user logs onto the network, he/she wants to check the presence information of other users that he/she is subscribed to. This is the responsibility of the user's server, and thus it sends a presence probe stanza for each subscription. A server that has the presence information about the user of interest

responds to the presence probe stanza with a presence stanza that contains the full presence information.

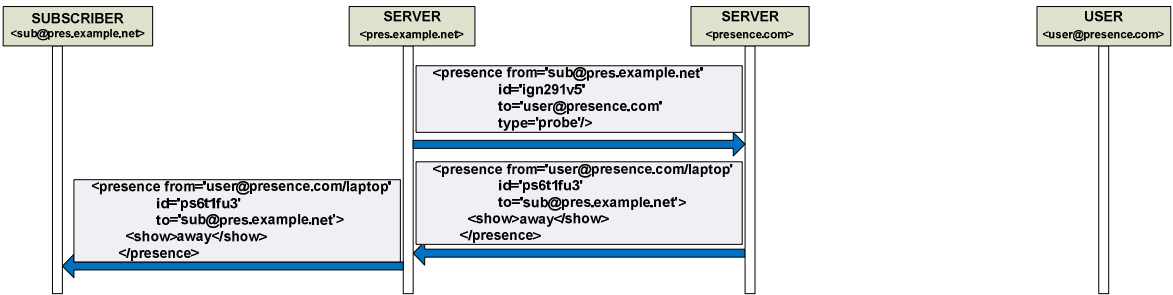


Figure 3.4 Notification process using XMPP

4. Rich presence service for physical presence

The rich presence service (RPS) is a service that manages physical and virtual presence. It may be used by many applications that need a unified presence solution. The RPS uses the RPID format [14] for storing and exchanging presence information. RPID includes contextual information, such as location, mood and activity (associated with physical presence) or status and contact means (associated with virtual presence).

State-of-the-art presence service implementations typically do not support physical presence, only virtual presence, offering simple subscriptions to all presence updates generated by a single presentity, information privacy is quite limited [2]. The RPS (developed at Department of Telecommunications of the Faculty of Electrical Engineering and Computing, University of Zagreb) [1] is a system that provides a presence service without mentioned deficiencies. Integration of the RPS with `where.deri.ie` enables additional services associated with the presence. Where.deri.ie updates user presence information, the RPS processes and disseminates it to interested users.

4.1. Architecture

The position of the RPS within a presence system and its components are shown in Figure 4.1. The RPS integrates SIP and XMPP presence solutions with additional presence related services (e.g. meeting scheduler, calendar).

The central components of the RPS are the following: rich presence layer, publish/subscribe layer, and policy server. The rich presence layer (RPL) offers a web service interface and can be accessed directly via SOAP. The RPL accepts presence updates in the RPID format and presence subscriptions defined using XPath. The RPL converts presence updates and subscriptions to a publish/subscribe message format.

The publish/subscribe layer (PSL) is implemented by the Mobile Publish/Subscribe (MoPS), a content-based publish/subscribe system for filtering and disseminating presence updates optimized for mobile environments. The PSL accepts presence data from various presence-related data sources (e.g. Facebook, Twitter, a sensor network, calendar) either directly when a data source is extended by a MoPS publisher that publishes data objects

according to the MoPS publish/subscribe protocol, or through the middleware layer for presence-data acquisition which fetches data and converts them to a publish/subscribe message format. The combination of different means for providing data to the system improves its performance, e.g. presence related data does not need to pass through the two layers (rich presence middleware and publish/subscribe) to be matched against subscriptions. Moreover, a distributed publish/subscribe implementations allows efficient filtering of data objects close to data sources even if they have high publication rates.

The rich presence service includes a Presence server which consists of an Openfire XMPP server and/or a SIP presence server. Both servers need to be expanded to communicate with the rich presence layer. Thus, the RPS serves may serve as a gateway between XMPP and SIP networks, thus offering a rich presence solution to the users of both networks (since servers without such extensions can process only simple presence messages). Such distributed architecture enables integration with various presence domains and presence service implementations.

A policy server handles watcher and presentity policies. A presentity can set different levels of presence visibility to different groups of watchers (e.g. his/her location is visible to family members all the time, but to colleagues location is visible only during office hours). A watcher defines policies for receiving presence updates (e.g. do not display presence updates during a meeting, a company can block presence updates from friends during office hours). Also policies can be used for defining preferable means (devices and applications) for notification delivery. In the current implementation, the policy server is used to define access rights to presence information between watchers and presentities.

Applications can use the RPS either directly, or through a presence server. When using clients that do not support the RPID format, contextual data from a RPID document is displayed as a textual message in addition to presentity state, while a watcher defines rich presence subscriptions directly using the web interface of the RPS.

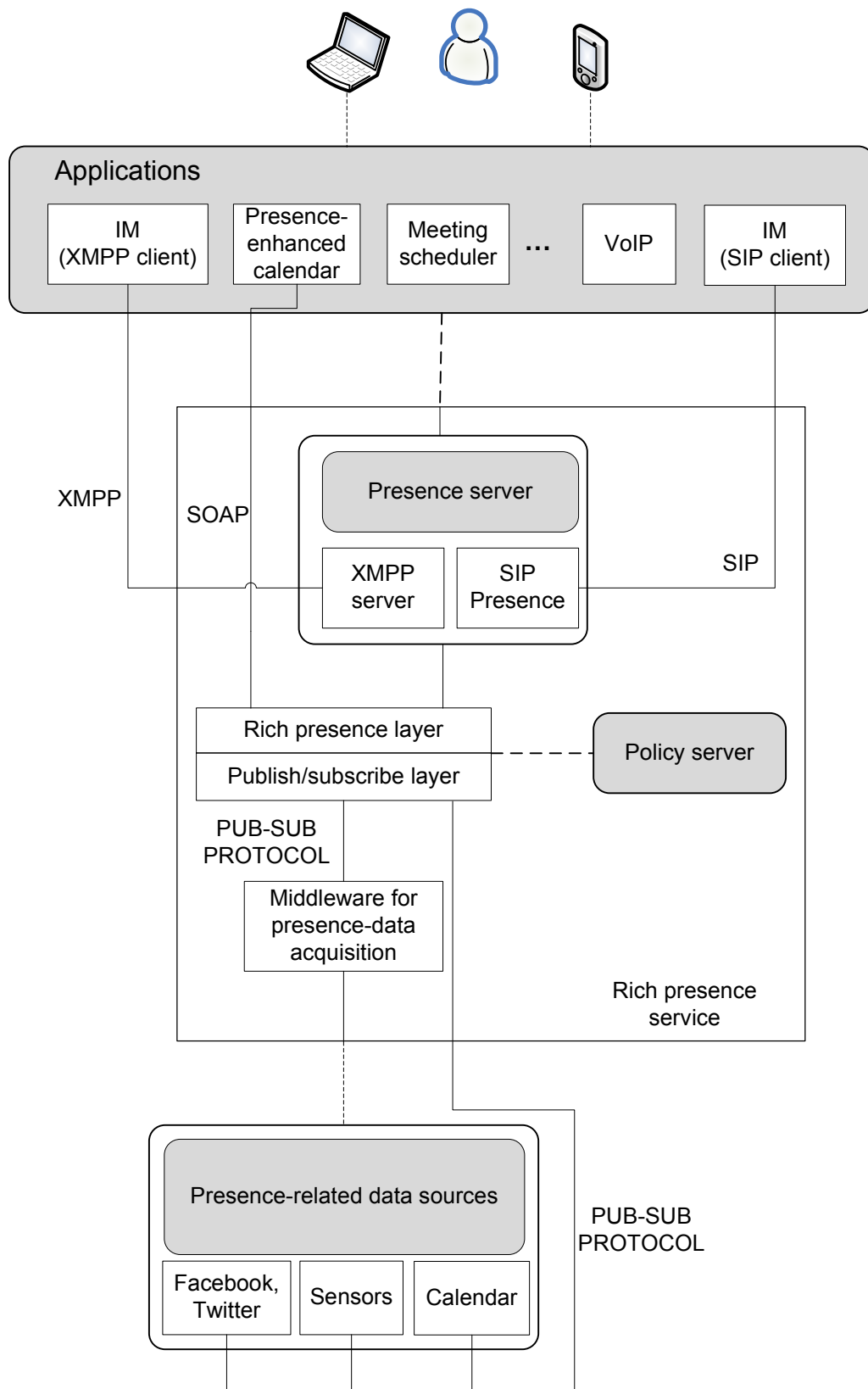


Figure 4.1 Rich presence architecture

4.2. where.deri.ie

Digital Enterprise Research Institute (DERI) has developed an application that manages presence information using the Linked Data and Social Semantic Web principles. The goal is to rely on the already existing models and data (e.g. the FOAF ontology), design a loosely-coupled architecture and make it accessible from a wide range of devices. This application is used as an independent presence-related data source that combines presence updates directly from users (using mobile phone applications) and from a sensor network.

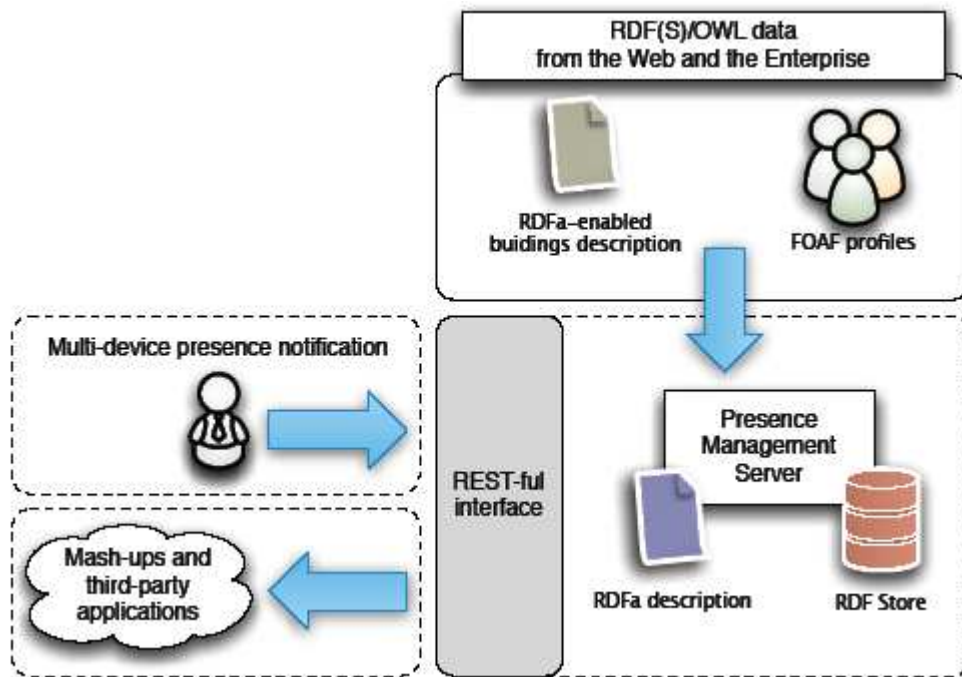


Figure 4.2 Architecture of the Presence Management System

The architecture (as shown in Figure 4.2) consists of independent data sources from the Web (e.g. user profiles in FOAF⁷), a Presence Management System (PMS) and third-party applications [15]. The PMS aggregates data and manages presence information. For publishing new presence information, third-party applications on various devices are used (e.g. a mobile application that checks whether a user is in the room or in the building). Independent data sources from the Web are used by PMS to retrieve context of presence information and details about users (e.g. picture, full name, affiliation).

⁷ FOAF (from „friend of a friend“) is an RDF based schema to describe persons and their social network in a semantic way

The service is visible on <http://where.deri.ie> (as shown in Figure 4.3), the URL points to the root of the PMS. For publishing new presence information, HTTP POST request is used containing a FOAF URI of the user, while the request is addressed to the URI of the room. For example, `POST /room/r309 HTTP/1.1` is used when a user logs into the room 309. The service is using a REST-ful design pattern, therefore a room URI is: <http://where.deri.ie/room/xxx>, where xxx is an identifier of the room. The Web service provides the information about a user, his/her location, and check in time. Also it indicates user availability, comment, and offers a link to the user's profile page.









- Aleksandar  in Room 309  on Tue, 10 May 2011 14:47:18 +0100 using Web interface
- Vinod Hegde  in Meeting Room A (213)  on Thu, 05 May 2011 14:56:19 +0100 using Android application (poi wg call)
- Alexandre Passant  in Conference Room (212)  on Fri, 15 Apr 2011 15:15:07 +0100 using Android application (Comments)
- Alexandre Passant  in Room 117  on Fri, 15 Apr 2011 14:02:47 +0100 using Android application (Comments)

Figure 4.3 Presence information published on the website

The publication of the new presence information is possible through the Web interface, using an application for Android phones or with an active RFID⁸ tag. RFID base stations are deployed in the DERI building, which, with active RFID tags form the global sensor network (GSN). Each RFID tag is associated to a FOAF profile of a user wearing it, thus enabling tracking the location of a user. A sensor network can provide location information about a user, but status information is provided using a simple mobile application that indicates status, available or unavailable. When the phone is turned over (display facing the floor) the user status is set to unavailable, while when the phone is turned back, the user status is set to available.

Another Android application allows users to manage their presence information by scanning a QR – code⁹ assigned to a location. This application requires deployed QR – codes throughout the environment.

An alternate way to update presence information is by using a web interface where a user fills out the form on the webpage of the appropriate room.

⁸ RFID = Radio Frequency Identification

⁹ QR = quick response; QR – code is a matrix barcode readable by dedicated QR barcode readers or camera phones.

4.3. Implementation of the RPS and integration with where.der.i.e

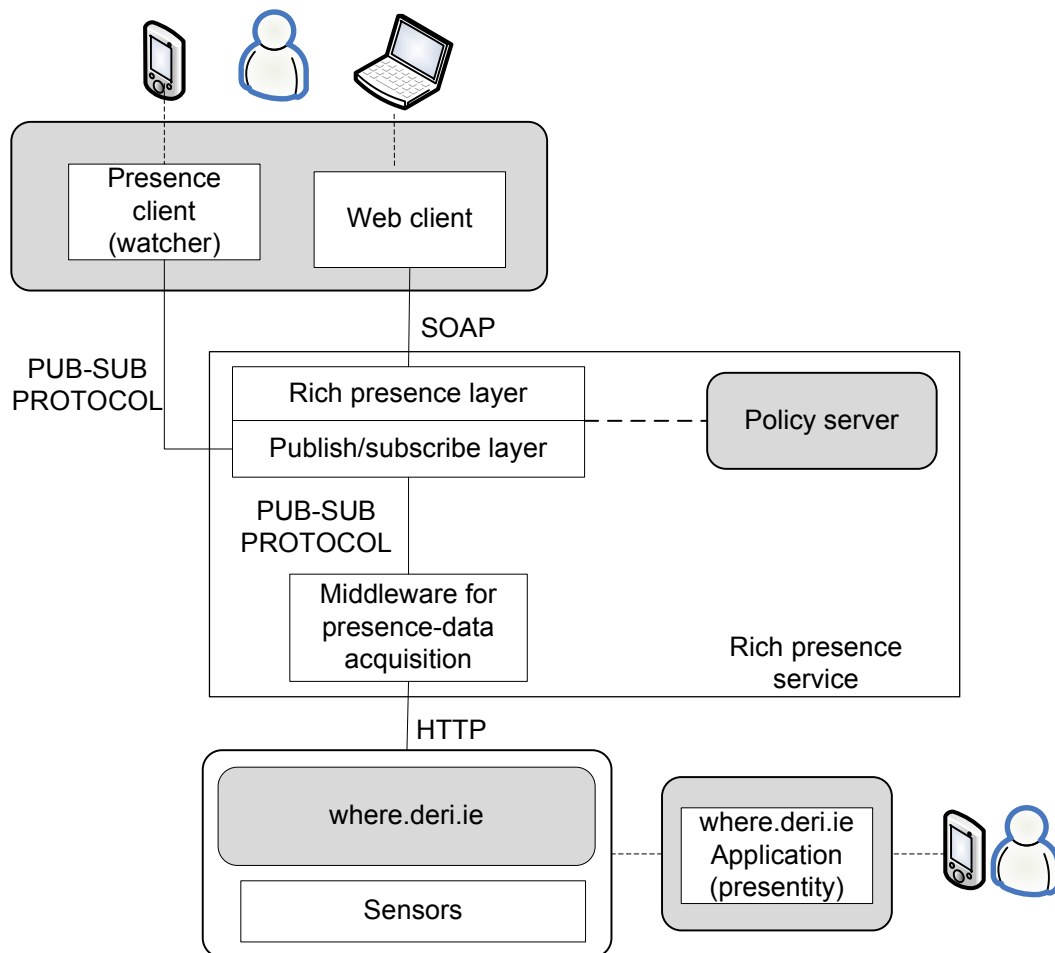


Figure 4.4 Implementation the RPS at DERI

The rich presence service implementation at DERI is shown in Figure 4.4 and it is done in accordance with the architecture defined in chapter 4.1.

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. It accepts rich presence subscriptions in the form of an XPath query or conjunction of XPath queries, and presence status updates in RPID format both defined by end users or generated by presence-enabled applications.


```
/presence/[@entity=http://www.deri.ie/about/team/member/aleksandar_antonic#me]  
/presence/tuple/[@location-info= http://lab.linkeddata.deri.ie/2010/deri-rooms#r309]
```

Figure 4.5 An example Xpath subscription

Figure 4.5 shows an example rich presence subscription defined using XPath. The subscription is related to Aleksandar AntoniĆ, identified by his FOAF profile URL which states that a presence update should be sent when his location equals room 309 in the DERI building (visible from the FOAF profile URL of room). In cases when the RPS includes an XMPP and/or SIP presence service that use pres URI to identify an entity (as defined in RFC 3859 [4]), it is necessary to convert a FOAF profile into a press URI. In addition, the RPL needs to identify a watcher that has defined a subscription in order to be able to forward presence updates to the adequate destination. Therefore, the RPL maintains a list of watchers that are currently registered and connected to it (each watcher is identified by a FOAF profile URI). An XPath subscription is converted by the RPL to publish/subscribe subscription format (as shown in Figure 4.6) before being forwarded to the PSL.

```
<?xml version="1.0" encoding='UTF-8'?>  
<subscription type="booleanHashtable">  
  <validity>-1</validity>  
  <attribute name="location-info" operator="=" type="string">  
    http://lab.linkeddata.deri.ie/2010/deri-rooms#r309</attribute>  
  <attribute name="entity" operator="=" type="string">  
    http://www.deri.ie/about/team/member/aleksandar_antonic#me</attribute>  
</subscription>
```

Figure 4.6 An example publish/subscribe subscription

Rich presence status updates are encoded using the RPID format. An example RPID publication is given in Figure 4.7 which defines a presence status for Aleksandar AntoniĆ as open, and his location is room 309. Figure 4.8 shows the same publication after conversion by the RPL into the publish/subscribe format. The same format of documents are used in the notification process, only this time the RPL converts messages from the publish/subscribe format to RPID format.

```

<?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:rpidd="urn:ietf:params:xml:ns:pidf:rpidd"
  entity=" http://www.derii.ie/about/team/member/aleksandar_antonic#me">
  <tuple id="bs35r9">
    <status>
      <basic>open</basic>
    </status>
    <priority>3</priority>
    <location-info> http://lab.linkeddata.derii.ie/2010/deri-rooms#r309</location-info>
    <timestamp>2011-05-27T16:49:29Z</timestamp>
  </tuple>
</presence>

```

Figure 4.7 An example RPID publication/notification

```

<?xml version="1.0" encoding="UTF-8"?>
<publication type="hashtable">
  <validity>-1</validity>
  <attribute name="presence/entity" operator="==" type="string">
    http://www.derii.ie/about/team/member/aleksandar_antonic#me </attribute>
  <attribute name="tuple/id" operator="=="
    type="string">bs35r9</attribute>
  <attribute name="status/basic" operator="=="
    type="string">open</attribute>
  <attribute name="location-info" operator="==" type="string">
    http://lab.linkeddata.derii.ie/2010/deri-rooms#r309</attribute>
  <attribute name="priority" operator="==" type="integer">3</attribute>
  <attribute name="timestamp" operator="==" type="string">2011-05-
    27T16:49:29Z</attribute>
</publication>

```

Figure 4.8 An example publish/subscribe publication/notification

The rich presence layer also integrates an XML database (eXist database) to store all subscription and publications. The database is used as persistent storage.

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 (based on TCP) 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 (as shown in Figure 4.6). A publication is a simple hashtable with (attribute, value) pairs (as shown in Figure 4.8). Such definition of publications and subscriptions on the PSL enables the implementation of efficient algorithms for matching incoming publications to numerous subscriptions organized in trees or lists (depends on the type of subscription). Since the publish/subscribe layer offers a remote service interface, presence sources can publish and receive presence updates directly through this interface, and thus avoid the conversion from the RPID to publish/subscribe format.

The policy server stores policies defined by presentities granting access rights to watchers and groups of watchers. After receiving a subscription from a watcher, the PSL retrieves, from the policy server, the watcher access right which is then integrated in the subscription. This avoids unnecessary generation of notifications inside the system (when condition from a subscription is satisfied but the user does not have sufficient access rights to see a notification). The policy server stores two types of policies in the XML database. The first one contains all subscribers and their membership in user groups together with their position within the group (e.g. Ph.D./M.Sc. student, unit leader), and the second one associates user groups and positions to access rights.

Methods implemented by the RPS (as shown in Figure 4.9) 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;
- `publish` – publishes a presence update;
- `notify` – notifies a watcher about presentity's presence updates.

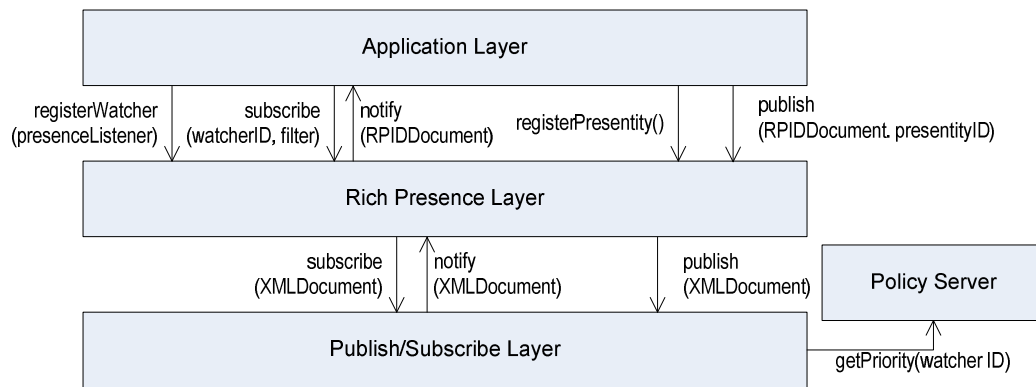


Figure 4.9 Rich presence service methods

Figure 4.10 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 RPL. The RPL responds with a unique watcher identifier and creates a new subscriber object associated with this watcher. When a watcher wishes to subscribe, it sends a subscription request containing an XPath subscription and its identifier. The RPL stores the subscription in the form of an XML document in its XML database, converts the XPath subscription into an XML document recognized by the PSL, and uses the watcher subscriber object to submit the XML subscription to the PSL. The PSL gets access level of watcher from the policy and integrates it into the subscription. Later on, a presentity updates its presence information using `where.derive`. The middleware layer receives a forwarded presence update, converts it into an XML document recognized by the PSL, and publishes the document to the PSL. Since the XML document matches a previously defined subscription, the PSL notifies the watcher subscriber object at the RPL with the matching document. Subsequently, the RPL converts the received document into RPID format, and forwards it to the watcher listener.

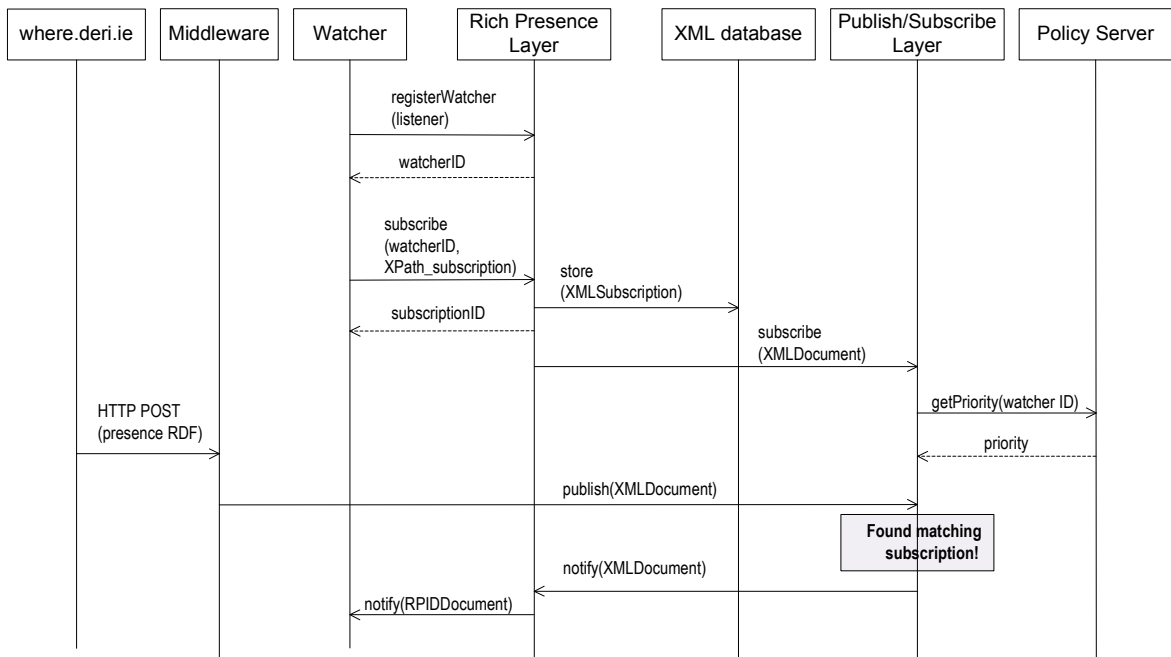


Figure 4.10 Sequence diagram showing an interaction within the RPS

5. Supported Presence Subscriptions

The rich presence service offers to a user three types of subscriptions: simple, parallel and sequence subscriptions. A simple subscription is used when a user subscribes to presence information of a single presentity. Two other subscription types are composite subscriptions, and enable subscriptions to more presentities at the same time.

5.1. Simple subscription

A simple subscription consists of a conjunction of predicates which describe the user request for presence information of a presentity. Each predicate is composed of an attribute (an attribute is an element in RPID format, i.e. entity, location-info, status/basic, note, etc.), operator (equal or not equal) and value. A user also defines the duration of a subscription. As long as the subscription is active, the user receives notification messages (presence updates about a presentity). Notification message carry complete presence information (complete RPID document as it is published by the presentity).

5.1.1. Simple presence service

Simple subscriptions need to contain a presentity URI (as shown in Figure 5.1). Additional constraints can also be specified, such as location and status. Figure 5.2 shows a received matching notification when user of interest updates his presence information..

Presence client

Notifications Subscriptions

Add new subscription Simple subscription

Entity (name) aleksandar_antonic ☐ NOT Validity (minutes)

Location (room, e.g. r309) ☐ NOT

Status (open or closed)

Subscribe

Load subscription

Figure 5.1 Simple subscription

Presence client

Notifications Subscriptions

```
<?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:rpidd="urn:ietf:params:xml:ns:pidf:rpidd"
entity="http://www.deri.ie/about/team/member/aleksandar_antonic#me">
  <tuple id="sd35r9">
    <timestamp>2011-06-12T16:18:28Z</timestamp>
    <priority>1</priority>
    <location-info>http://lab.linkeddata.deri.ie/2010/deri-rooms#r309</location-info>
    <status><basic>http://online-presence.net/opo/ns#Available</basic></status>
  </tuple>
</presence>
```

Load notification Delete notification

Figure 5.2 Presence service notification

5.1.2. Geo-fence service

Simple subscriptions can also be used for the implementation of geo-fence services. A geo-fence is a virtual perimeter for a real-world geographic area. A geo-fence service is a service which sends a warning to a user when a person or a resource crosses the fence (i.e. leaves a designated area). The rich presence system in conjunction with a sensor network can be used as a geo-fence system. Figure 5.3 shows a subscription used as the geo-fencing service. The subscription expresses a request for a notification whenever the video projector 1 is not in room 212 and the subscription validity time is infinite. Independent presence-related data source provides location information about a resource to the RPS (RFID base stations monitor location of a RFID tag associated with a resource). When the RPS detects that the resource is not in the appropriate room, it sends a notification to the administration office.

The screenshot shows a window titled "Presence client" with two tabs: "Notifications" and "Subscriptions". The "Subscriptions" tab is active. Under the heading "Add new subscription", there is a dropdown menu currently showing "Simple subscription". Below this, there are three input fields: "Entity (name)" with the text "video_projector1", "Location (room, e.g. r309)" with the text "r212", and "Status (open or closed)" which is empty. To the right of the "Entity (name)" field is a checkbox labeled "NOT" which is unchecked. To the right of the "Location (room, e.g. r309)" field is a checkbox labeled "NOT" which is checked. To the right of the "Validity (minutes)" field, which contains the text "-1", is another checkbox labeled "NOT" which is unchecked. At the bottom right of the form area is a "Subscribe" button. At the bottom center is a "Load subscription" button.

Figure 5.3 Geo-fencing service subscription

5.1.3. Virtual secretary

Another possible usage of a simple subscription is the implementation of a virtual secretary. Virtual secretary is a service that provides information about people who wanted to contact a user while he/she was unavailable (e.g. in a meeting, busy). A user subscribes to presence information related to his/her office (as shown in Figure 5.4). When someone comes to the user's office and realizes that the user is not available at the moment, the RPS

records the visitor's presence in the user's office. When the user becomes available (e.g. returns to the office, finishes a meeting) the RPS will send notifications for each person that visited the office during the period of unavailability. If a visitor wants to leave a message, he/she can include it in a presence update (as shown in Figure 5.5).

Figure 5.4 Virtual secretary subscription

```
<?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="http://www.deri.ie/about/team/member/aleksandar_antonic#me">
  <tuple id="sd2184">
    <timestamp>2011-06-12T14:15:48Z</timestamp>
    <priority>3</priority>
    <location-info>http://lab.linkeddata.deri.ie/2010/deri-rooms#r200</location-info>
    <note>Call me on my cell when you are back. I'm leaving now.</note>
    <status><basic>http://online-presence.net/opo/ns#Available</basic></status>
  </tuple>
</presence>
```

Figure 5.5 Virtual secretary notification

5.2. Parallel subscription

A parallel subscription consists of a conjunction of simple subscriptions thus allowing a user to receive presence notifications depending on the presence status of multiple users at the same time. Notification is generated when presence information of each present entity overlaps another matching subscription. A threshold for generating a notification message is set by a user via the variable coverage. The variable coverage represents the minimum percentage of satisfied simple subscriptions such that a matching event is detected. Figure 5.6 shows a parallel subscription to presence updates of three different users expressed as simple subscriptions. If the coverage has a value of 1 (100%) a notification is generated only at a point in time t_3 ; if coverage is set to 0.66 (66%) or less, a notification is generated at time points t_2 and t_3 . The coverage with value of 0.33 (33%) or less generates a notification at all three time points.

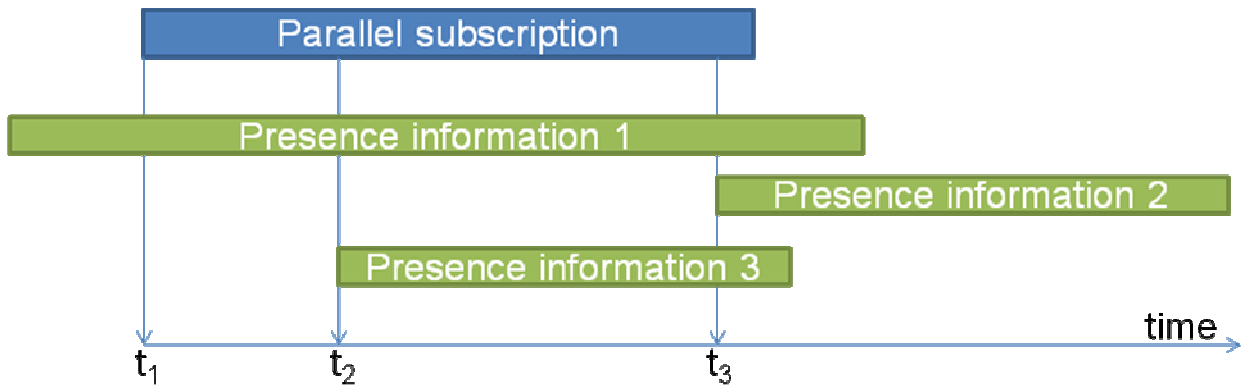


Figure 5.6 Parallel subscription coverage

5.2.1. Meeting scheduler

A meeting scheduler is an application for scheduling meeting times and organizing personal itineraries. A meeting scheduler can retrieve presence statuses (virtual and physical) of potential participants before the start of the meeting (e.g. 5 minutes before the start) and then proceed in accordance with the received data (cancel a meeting, reschedule or send a confirmation). Figure 5.7 shows a parallel subscription to three (potential) participants who have open (i.e. available) status and coverage is set to 66% (a meeting can be held without one person). If a meeting scheduler receives a notification from the RPS before the beginning of the meeting, it can send confirmation that the meeting will take place at the agreed time. If notification is not received before the beginning of the meeting

(it means that two or more users are not available to attend the meeting), a meeting scheduler should try to reschedule or cancel the meeting. A meeting scheduler waits for a notification as shown in Figure 5.8.

Presence client

Notifications Subscriptions

Add new subscription Parallel subscription ▼

Entity (name) ling_chen; anh_le_tuan; danh_le_phuoc; Validity (minutes)

Location (room, e.g. r309)

Status (open or closed) open

Coverage (0.0 - 1.0) 0.66

Subscribe

Load subscription

Figure 5.7 Meeting scheduler subscription

Presence client

Notifications Subscriptions

```
<?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="Rich presence system">
  <timestamp>2011/06/20 00:00:54</timestamp>
  <status><basic_3>http://online-presence.net/opo/ns#Available</basic_3></status>
  <presence><entity_3>http://where.deri.ie/person/danh_le_phuoc#me</entity_3></presence>
  <status><basic_2>http://online-presence.net/opo/ns#Available</basic_2></status>
  <presence><entity_2>http://where.deri.ie/person/anh_le_tuan#me</entity_2></presence>
  <status><basic_1>http://online-presence.net/opo/ns#Available</basic_1></status>
  <presence><entity_1>http://where.deri.ie/person/ling_chen#me</entity_1></presence>
  <type>parallel</type>
</presence>
```

Load notification Delete notification

Figure 5.8 Meeting scheduler notification

5.2.2. Sequence subscription

A sequence subscription consists of a conjunction of simple subscriptions, but a notification is generated after a sequence of events specified in a sequence subscription, regardless of the time when a certain event occurs. Figure 5.9 shows a sequence subscription to presence updates of three different users. The system generates a notification only at t_3 because then a sequence of events stated in the subscription is fully satisfied. The fact that presence information of the user 1 and user 2 does not satisfy simple subscriptions at the time t_3 , has no effect on a notification process.

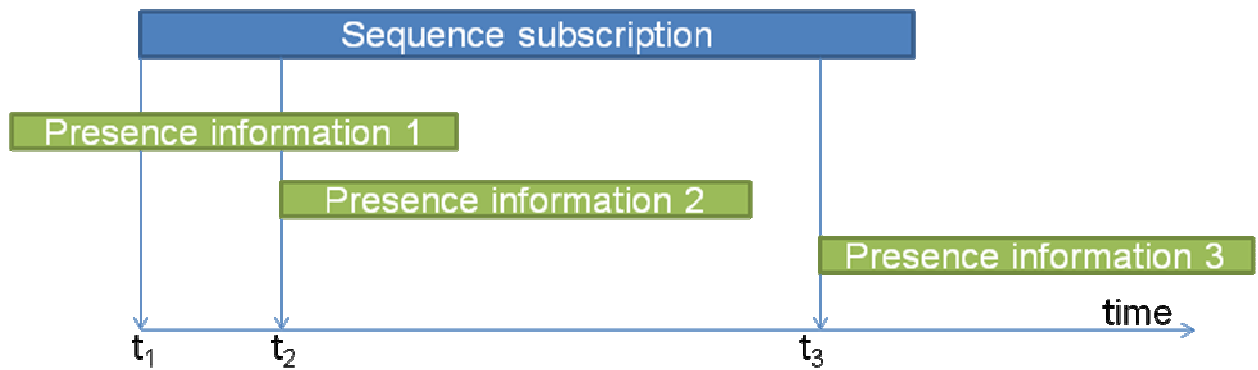


Figure 5.9 Sequence subscription coverage

A sequence subscriptions can be used for studying people behavior and habits, for example to monitor the physical movement within the building, number of changes in the availability of a user during a period of time. Figure 5.10 shows an example sequence subscription. The subscription is related to Ling Chen and states that a presence update should be sent when his availability changes from open to closed. Figure 5.11 shows a notification generated for a sequence subscription request.

Presence client

Notifications Subscriptions

Add new subscription Sequence Subscription ▼

Entity (name) ling_chen; ling_chen; Validity (minutes)

Location (room, e.g. r309)

Status (open or closed) open; closed;

Subscribe

Load subscription

Figure 5.10 Sequence subscription example

Presence client

Notifications Subscriptions

```
<?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:rpidd="urn:ietf:params:xml:ns:pidf:rpidd"
entity="Rich presence system">
  <timestamp>2011/06/20 00:05:50</timestamp>
  <presence><entity_2>http://where.deri.ie/person/ling_chen#me</entity_2></presence>
  <type>sequence</type>
  <presence><entity_1>http://where.deri.ie/person/ling_chen#me</entity_1></presence>
  <status><basic_2>http://online-presence.net/opo/ns#DoNotDisturb</basic_2></status>
  <status><basic_1>http://online-presence.net/opo/ns#Available</basic_1></status>
</presence>
```

Load notification Delete notification

Figure 5.11 Notification on sequence subscription request

Conclusion

Existing solutions implementing presence are very popular despite the fact that they have limited support for context-awareness. The main drawbacks are limited support for presence information filtering and insufficient privacy control. This thesis presents a solution which uses the rich presence specification in conjunction with filtering of presence updates. The rich presence service (RPS) can handle complex subscriptions, provides better information access control and is independent of a particular presence protocol. It integrates a content-based publish/subscribe implementation for efficient matching of presence-related subscriptions to presence updates, and uses where.deri.ie as an independent presence-related data source of linked data managing presence information related to people in the DERI building.

The integration of the RPS with where.deri.ie has created a platform for a number of new services for end users: a user-related presence service with additional presence status filtering, geo-fence service and virtual secretary. Besides the listed services, the RPS can interact with many other applications such as a calendar application, meeting scheduler, or various policy-enabled applications which change the presence status of a user depending on the current time of day, location and activity [1]. The interaction between the RPS and applications is achieved through the web service interface offered by the rich presence layer or directly through the publish/subscribe layer. The RPS is a step closer to a complete solution to the problem of consolidated presence as described in Hauswirth et al. [2].

Future work includes expansion of the policy server to include context-dependant policy rules. Another interesting direction is the replacement of the RPID format with an appropriate RDF Schema as a uniform platform for processing presence information. Future efforts will also be directed to improving system performance and to the development of new applications which rely on the RPS.

References

- [1] PODNAR ŽARKO, I., KUŠEK, M., PRIPUŽIĆ, K., ANTONIĆ, A. Presence@FER: An Ecosystem for Rich Presence, in *Proceedings of the 11th International Conference on Telecommunications*, Graz, Austria, June 2011, pp. 133-140.
- [2] HAUSWIRTH, M., EUZENAT, J., FRIEL, O., GRIFFIN, K., HESSION, P., JENNINGS, B., GROZA, T., HANDSCHUH, S., PODNAR ŽARKO, I., POLLERES, A., ZIMMERMANN, A. Towards consolidated presence, in *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2010*, Chicago, Illinois, USA, October 2010, invited paper.
- [3] DAY, M., ROSENBERG, J., SUGANO, H. A Model for Presence and Instant Messaging (RFC 2778), IETF, February 2000., <http://www.ietf.org/rfc/rfc2778.txt>
- [4] PETERSON, J. Common Profile for Presence (CPP) (RFC 3859), IETF, August 2004., <http://www.ietf.org/rfc/rfc3859.txt>
- [5] KHARTABIL, H., LEPPANEN, E., LONNFORS, M., COSTA-REQUENA, J. An Extensible Markup Language (XML)-Based Format for Event Notification Filtering, IETF, September 2006., <http://www.ietf.org/rfc/rfc4661.txt>
- [6] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., SCHOOLER, E. SIP: Session Initiation Protocol (RFC 3261), IETF, June 2002., <http://www.ietf.org/rfc/rfc3261.txt>
- [7] ROSENBERG, J. A Presence Event Package for the Session Initiation Protocol (SIP) (RFC 3856), IETF, August 2004., <http://www.ietf.org/rfc/rfc3856.txt>
- [8] NIEMI, A. ED. Session Initiation Protocol (SIP) Extension for Event State Publication (RFC 3903), IETF, October 2004., <http://www.ietf.org/rfc/rfc3903.txt>
- [9] SUGANO H., FUJIMOTO S., KLYNE G., BATEMAN A., CARR W., PETERSON, J. Presence Information Data Format (PIDF) (RFC 3863), IETF, August 2004., <http://www.ietf.org/rfc/rfc3863.txt>
- [10] Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Core (RFC 6120), IETF, March 2011., <http://www.ietf.org/rfc/rfc6120.txt>
- [11] Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence (RFC 6121), IETF, March 2011., <http://www.ietf.org/rfc/rfc6121.txt>
- [12] MILLARD, P., SAINT-ANDRE, P., MEIJER, R. XEP-0060: Publish-Subscribe (v1.13), XMPP Standards Foundation (XSF), July 2010., <http://xmpp.org/extensions/xep-0060.html>
- [13] SAINT-ANDRE, P., SMITH, K. XEP-0163: Personal Eventing Protocol (v1.2), XMPP Standards Foundation (XSF), July 2010., <http://xmpp.org/extensions/xep-0163.html>
- [14] SCHULZRINNE, H., GURBANI, V., KYZIVAT, P., ROSENBERG, J. RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF) (RFC 4480), IETF, July 2006., <http://www.ietf.org/rfc/rfc4480.txt>

- [15] PASSANT, A., HEGDE, V., REYNOLDS, V., CYGANIAK, R., HAUSWIRTH, M. Linked Data, Social Semantic Web and QR codes for Presence Management in the Enterprise
- [16] Content Networking lectures, Faculty of Electrical Engineering and Computing, University of Zagreb, 2011.

Summary

Title: Context – aware filtering and dissemination of rich presence information

Summary: Presence is defined as user willingness and ability to communicate with other users across a set of devices and tools. Presence service is a service which receives, stores, and disseminates presence information to all interested parties. This thesis presents the rich presence service solution developed at the Department of Telecommunications of the Faculty of Electrical Engineering and Computing, University of Zagreb which is integrated with where.der.i.e developed by Digital Enterprise Research Institute, National University of Ireland, Galway. The first part of the thesis gives an overview of the basic model for presence service and describes the two most widely used protocols for presence, SIP and XMPP, while the second part describes the rich presence service solution and supported services.

Keywords: presence information, physical presence, virtual presence, rich presence, PIDF, RPID, publish–subscribe system, XMPP, SIP Presence, watcher, presentity, policy control

Sažetak

Naslov: Kontekstno – svjesno filtriranje i razašiljanje informacija bogate prisutnosti

Sažetak: Prisutnost se definira kao spremnost i mogućnost korisnika na komunikaciju s drugim korisnicima koristeći različite uređaje i alate. Usluga prisutnosti je usluga koja prima, pohranjuje i razašilje informacije o prisutnosti svim zainteresiranim strankama. Ovaj rad prezentira sustav koji pruža uslugu bogate prisutnosti razvijen na Zavodu za telekomunikacije, Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu, integriran sa sustavom where.der.i.e razvijenim u Digital Enterprise Research Institute, National University of Ireland, Galway. Prvi dio rada daje pregled osnovnog modela za uslugu prisutnosti i opis dva najraširenija protokola, SIP i XMPP, a drugi dio opisuje sustav za uslugu bogate prisutnosti i usluge koje pruža.

Ključne riječi: informacija o prisutnosti, fizička prisutnost, virtualna prisutnost, bogata prisutnost, PIDF, RPID, objavi–pretplati sustav, XMPP, SIP Presence, pretplatnik, objavljiivač, kontrola pristupa