Composition for Enhanced SIP Presence

Ron Shacham
Department of Computer Science
Columbia University
New York, NY 10027
shacham@cs.columbia.edu

Henning Schulzrinne
Department of Computer Science
Columbia University
New York, NY 10027
hgs@cs.columbia.edu

Wolfgang Kellerer
DoCoMo Communications Labs Europe
Munich 80687
Germany
kellerer@docomolab-euro.com

Srisakul Thakolsri
DoCoMo Communications Labs Europe
Munich 80687
Germany
thakolsri@docomolab-euro.com

Abstract

Presence has taken shape as a way to present a comprehensive view of the communications capabilities of a user or resource. Standardized protocols allow for receiving presence data from a variety of sources, such as a user's communication devices, cellular provider, online calendar and sensors in his environment. This data can describe many different aspects of his communication capability, such as his device characteristics, activities, and even physical locations. The large number of presence sources may lead to irrelevant or inconsistent data. Also, the data may be incomplete and not as usable as it could be to the presence watcher. This paper proposes the use of presence composition to remove unusable data and create new more usable data. We discuss the details of this composition and present a format with which a user can specify a policy for composition.

1 Introduction

In its simplest form, presence is a binary indication of a user's online status for a specific application, such as instant messaging. As IP multimedia communication becomes more widespread, presence is taking on a more comprehensive and ambitious role. One standards document [2] defines it as follows: "Presence conveys the ability and willingness of a user to communicate across a set of devices." In fact, presence describes not only human users, but anyone or anything, broadly grouped under the term "presentity", found in [14]. Receivers of this presence information, called watchers, can use it not only to know whether the user is online or not, but also to make better decisions about communicating with him.

This comprehensive view of a user's communication is different in the type of information that it conveys and in the sources of information. Far richer than a simple on/off status, it can potentially provide information about the types of media and devices to which the user currently has access, what he is currently doing, and even his physical location. Furthermore, instead of being based only on the input of a single type of application, presence could be based on input from PCs, cellular devices, calendar applications, and sensors in the user's environment.

The large amount of information could prove difficult for the watcher to use. Some available information may be incorrect because it is stale, contradictory or redundant. Humans may be able to resolve these issues by looking at the data supplied by all of the sources. However, that may sometimes by difficult to do, besides requiring more effort than many people are willing to exert. Moreover, watchers could also be applications that provide user-customized behavior based on presence. As an example, [3] describes the use of presence in personalized telephony services, such as call routing, using extensions to the Call Processing Language [4]. Such an application requires a single, accurate indication of the user's presence. The problem caused by having multiple sources can be remedied through the process of composition. Presence composition, is defined in [5] as the aggregation of the multiple sources of presence into a single record. This does not mandate any manner of merging the sources. In this paper, we more fully define composition, showing how it can present a more usable view of the user's presence. This is done by removing infor-

mation that falls into one of the categories mentioned above. Additionally, much useful information about the user is not provided by any source, but could be easily extrapolated based on existing information. For this reason, composition also derives new presence information.
We begin by introducing presence as defined in the Session Initiation Protocol (SIP) in Section 2. We then reference related work in Section 3. Following that, we discuss different types of presence sources in Section 4. In Section 5 we discuss the four steps of composition. Section 6 presents a format for specifying composition policies as well as an example of such a policy. We conclude in Section 7.

2 Introduction to SIP Presence

The Session Initiation Protocol (SIP) [1] is used for the establishment of multimedia sessions between multiple parties. This is accomplished using a specific method, or function, called INVITE. SIP has subsequently standardized event notification through the SUBSCRIBE and NOTIFY methods, for creating a subscription and sending a notification, respectively. The event-specific information may be updated in any manner, and SIP defines its own PUBLISH method for this purpose. The event mechanism has been used most notably for presence notification, defined in [15].

The general format of a user's presence is described in [2]. This is an XML document, divided up into multiple tuples, each describing information about the person himself, a communication service he uses or a device that he uses. These are called person tuples, service tuples, and device tuples, respectively, and they are represented by distrinct <person>, <tuple> and <device> elements. Each of these elements may have any number of sub-elements called attributes (not to be confused with XML attributes which are part of the element itself. We will refer to those in this paper as "XML attributes"). For example, a <person> element may have an <activities> sub-element listing his current activities or a <user-input> element to indicate whether the user is active or idle. The sub-elements contain further sub-elements representing their value. For example, a <person> element may contain an <activities> element which contains the elements <on-the-phone> and <appointment>. These elements are defined in [12] and other standards documents.
Services are defined by a "service URI", a unique identifier used to reach the user, such as a SIP URI. Each service must contain at least one sub-element, <status>, which includes a <basic> element that may be set to "open" or "closed". A service may be available on multiple devices. For example, a tuple that describes the user's ability to accept communications at his "sip" address may list multiple devices that may be used as endpoints for the service. Since specific device information is described in separate tuples, a "device-id" attribute is defined. This "device-id" may be used in a single device tuple, and in any number of service tuples to indicate that the service is available on that device.
Presence information is based on input received from possibly many sources. The compositor, defined in [5] is a module that is charged with joining these together into a single view of the user's presence. This paper discusses an expanded role for the compositor to remove ostensibly incorrect or irrelevant presence information and extrapolate new information, based on user specifications.

3 Related Work

The problem of inconsistent information associated with multiple contributing presence sources is presented in [7]. That work describes an implemented solution based on scripts that are triggered by changes in the presence data. Our paper goes into greater detail about the conceptual process of composition, such as the causes of inconsistent data and the need for deriving new data. Furthermore, their solution, though powerful, makes use of extensions to the presence format itself. Our solution works with the standard presence data that is normally aggregated by the presence server from various sources. Furthermore, the full-fledged programming solution described there, while providing more fine-grained capabilities to the user, is more complex than the policy rule format that we present here. We are more concerned with determining a set of basic composition functions and providing the user access to these through a simple XML format.

4 Types of Information Sources

A presence source is a provider of any presence information that is received by the compositor and integrated into the view of the user's presence. This information could be provided either through publication or subscription. The difference between these modes, besides the protocol-specific ones, is whether the compositor must actively seek the presence information. In publication mode, the source is configured to provide it to the compositor, whereas in subscription mode, the source waits for requests. However, how the information is injected into the presence system is less important to us, and we assume, in this paper that the compositor has a set of presence information from different sources without getting into how it was received.
We can classify all presence data as falling into one of the following categories:

- Reported current: Reported current information has been provided by the presentity within processing time

delays of the current time. We assume that this information is correct when entered, but the trustworthiness of the information is likely to decay as time goes on, given that most human users will find it difficult to continuously keep presence information up-to-date.

- Reported scheduled: For reported scheduled information, a presentity indicates its plans for the future rather than the present, in some type of calendar. The reliability of this information depends largely on the diligence of the user in updating calendars and similar sources.

- Measured device usage information: Measured device information uses observed user behavior on communication devices, such as the act of placing or receiving calls or typing, or device status such as signal strength. This data may come from the device itself or a network element. The main source of error is that it may not be known whether the presentity itself is using the device or some other person.

- Measured by sensors: Presence information measured by sensors reflects the status of the presentity, such as his location, place type, activity or other environmental factors. Examples of sensors include Global Positioning System (GPS) information for location or a BlueTooth beacon that announces the type of location, such as "theater", in which a person finds himself. Sensors have the advantage that they do not rely on humans to keep the information up-to-date, but are naturally subject to measurement errors, partly because they must make assumptions about identity. For example, a passive infrared sensor (PIR) can detect that somebody is in the office of the presentity, but cannot detect whether this is the presentity himself or the cleaning staff. A GPS sensor cannot detect whether the cell phone is being used by the presentity or has been borrowed by the presentity's spouse.

- Derived: Presence information might be derived indirectly from other sources of data. We discuss derivation in Section 5.2.

5 Composition Steps

The composition process takes an input set of data from presence sources and produces another presence document. The following steps are followed in order:

1. Discarding tuples
2. Deriving presence information
3. Resolving conflicts
4. Merging tuples

Discarding tuples should be done first. Since the eliminated tuples are not useful, regardless of any possible conflict, as described in Section 5.1, they have no reason to be included in the other steps. Deriving presence information is useful for identifying conflicts later, and should therefore be done next. Resolving conflicts between tuples must be done before merging, while the tuples are still distinct. We describe these steps in detail in the next four sub-sections.

5.1 Discarding

Whole tuples may be discarded, even if they do not conflict with others, if they fall into one of the following categories.

- Closed contacts: All service tuples with a basic status of 'closed'. The presentity is currently not reachable using such services.

- Old tuples: All person, service, or device tuples whose age is older than a given threshold. Any presence information has an expiration time, and should be removed at that point, so discarding applies only to tuples before their expiration.

- Unreferenced tuples: device tuples that are not referenced by any 'open' service tuple. This means that it does not represent current communication capabilities of the presentity, and is therefore of limited use to the watcher.

5.2 Deriving Presence Information

Certain presence sources may not be capable of publishing all relevant information, and users are unlikely to always update all information that requires their input. Such information may be derived in order to include it in the user's presence.
Derivation of new information makes it easier to identify a conflict with another presence source. For example, knowing the locations of two presence sources allows the compositor to determine that the user is only colocated with one of them, and the information from the other one is inaccurate. Therefore, deriving location where it would not otherwise be included could make a conflict clear to the compositor that it would not have detected.
It can also provide information to the watcher indicating communication capability that may not otherwise be known based on automatic presence sources. Based on known presence input, and possibly outside information, such as time of day, new and more useful data could be added to the user's presence. For example, a user's mobile device may easily be able to identify and publish that it is in a car based on available sensors. However, more relevant information for the watcher is that the user is driving, which

may be derived if this is usually true when the user is in his car (possibly during certain times, such as mornings and evenings). The user may also wish people to know that when he is "on-the-phone" (which may be discovered automatically by subscribing to the 'dialog' event, which summarizes a user's current communication sessions), this means that he is "busy" and shouldn't be called unless it is urgent. The user may know that a specific place does not allow for private communications, and he may automatically supplement his location information with privacy information. When 'user-input' appears as 'idle' between certain hours of the night, the user's activity should be set to 'sleeping'.
Such derivations each have two parts: a predicate and additional content. The predicate is one or more elements that must all be present in a tuple, or outside information that must be true, in order for the specified content to be added.
A special case of this is the supplementing of static information that doesn't depend on dynamically changing data. For example, a device may not support publishing certain presence extensions, but they may be added to its tuple. Such a derivation could be defined using only the device's device-id as the predicate.
There is another way that this static information can be supplemented. The Extensible Markup Language (XML) Configuration Access Protocol (XCAP) [9] is used to update XML-based application configuration data on a server. The document, and all of its attributes, are addressable using HTTP URIs, and manipulated through HTTP. This protocol has been applied to directly manipulate a user's presence, as described in [11]. XCAP does not manipulate the user's final presence that is shown to watcher, but, rather, a single document which is one of the inputs to the compositor, as described earlier. This document may contain device tuples containing static information about the device. In the merging stage of composition, described in Section 5.4, multiple tuples associated with a single device (containing identical device-IDs) are merged together. If no identical device tuple has been received from any other source, the static tuple will appear in the resulting raw presence document. If there is another identical tuple, the static and dynamic elements will be merged into a single tuple, adding the new information.

5.3 Resolving Conflicts

5.3.1 Sources of Information Conflict

Information conflict occurs when multiple sources give different views of the presentity, some of which may be outdated or incorrect. Information can be incorrect for any number of reasons, but some examples include:

- Location divergence: The publisher collecting the information may not be colocated with the presentity at this particular time. For example, Alice's home PC may report that the user is idle (not typing), but Alice is using the office PC. Presence information may be available based on the state of a device, but it may be borrowed temporarily by someone else (if the owner were to "log out" and unregister this device as a contact, this problem would not occur)

- Update diligence: Some sources, particularly those updated manually, are prone to only approximate reality. For example, few users record all appointments or meetings in their calendar or, conversely, remove all canceled meetings. This is particularly true for regularly scheduled activities such as meals or commute times.

- Sensor failure: Sources that report their information differentially are subject to silence ambiguity. If such a source does not report new data, the receiver cannot tell whether the sensor is malfunctioning or whether the information last received is still current. This can be partially mitigated by requiring sources to report when they are no longer confident of the data. However, this does not deal with sudden source failures. Thus, some form of keep-alive mechanism may well be needed that overrides differential notification mechanisms. Even with keep-alive, there is likely to be a substantial period of time between source failure and failure detection, causing stale information.

5.3.2 Detecting information conflicts

Detecting information conflicts is the first step in removing the inaccurate information. There are many elements in person tuples that could end up having conflicting values from different sources. However, this step is less relevant for service tuples. The elements found there are not likely to conflict, even if multiple tuples report information about the same service. For example, the basic status in a service tuple cannot be said to conflict with the status sent for the same service by another device. <deviceID>, <privacy>, and <user-input> describe a specific instance of the service and can all be true. Our discussion of conflict resolution is focused on person and device information.
Information conflicts can be classified as to how easy it is to detect them. We distinguish three types of information conflict: obvious, probable and undetectable, described in turn below. While it is possible that conflicts could exist across elements, where the value of one element conflicts with another, for simplicity, we look only at conflicts in the same element.
For some pieces of presence information, information conflicts are obvious and readily detectable. For example, a

single presentity can only be in one place at a time. Thus, if two sources report location information that differs by more than the margin of error, one must be wrong. Some elements such as <place-is> and <user-input> elements have exlusive values.
For other types of information, an automaton can guess with some probability that two sources of information contradict each other, but this may well depend on the values themselves. For example, the <activities> combination of:
<away>, <appointment>, <in-transit>, <meeting>, <on-the-phone>, <steering>
incrementally reported by different sources may well reflect the activity of the typical Wall Street commuter in the Lincoln Tunnel, speaking on his cell phone. The <place-type> element is another one that may take different values that are sometimes, but not always, contradictory. For example, the values "outdoors" and "stadium" differ only in their specificity. In lieu of a better approach, different values may be treated as complementary. However, a more refined approach for determining conflict between elements may be possible. For example, the approach used in [16] to describe situational terminology, such as "in a meeting" using ontologies may allow a logical determination to be made as to whether two terms, such as types of place or activities are contradictory.
Undetectable information conflicts are those where a machine lacking human intelligence cannot reliable detect that the two pieces of information cannot both be true. For example, an automaton is unlikely to be able to decide which of several notes or free-text fields is valid, without basing this on other information in the tuple, person or device element.

5.3.3 Handling Information Conflicts

Once an information conflict is detected, a choice must be made about how to handle it. In some cases, no action should be taken. For an element such as <activities> or <mood>, for which different reported values make sense and it is hard to distinguish which values really conflict, as mentioned above, the different values can be treated as non-conflicting. This means that both tuples are retained, and handling is deferred to the merging step, during which the multiple values will be unioned within a single tuple.
For other elements, however, conflict is more easily detectable and multiple values are not sensical. A conservative approach to handling such a conflict would be to simply list all values. This is different from the approach mentioned earlier, because the tuples are kept distinct and not merged in the next step. Multiple versions are presented which are admittedly conflicting, and the watcher may make a judgment about which is more correct. To limit the amount of information that the watcher must digest, it may be more useful to choose one value over the other. For this decision, one of the following heuristics may be used:

- Choose recent tuple: Choose the value from the tuple that was more recently published for the first time. Simply choosing the most recently updated value is likely to cause flip-flopping between dueling publishers.

- Choose trustworthy tuple: Choose the element from the more trustworthy tuple. Trustworthiness may be based on the source identity, such as a user's cell phone. Alternatively, it is based on the types of reporting listed in Section 2. For example, they may be ranked in the order "reported current", "measured device information", "measured by sensors", "reported scheduled". Derived information may be considered as reliable as the original information.

- Value of another element: Other elements may indicate that one version of the information should be trusted. For example, <user-input> may indicate that one device that provides presence is actually being used, and another is not.

When one value is chosen over another, the resulting presence document may be affected on the tuple level or on the element level. On the tuple level, the more trusted tuple is chosen and the other is discarded. On the element level, both tuples are maintained, but only the more trusted element is kept, while the other is discarded. Either of these approaches may have advantages in certain situations, but using only tuple-level conflict resolution is simpler and avoids inconsistencies in the final document.

5.4 Tuple Merging

Merging combines several tuples that logically represent the same information. For example, a presence document should only contain one report of <person> information, so the multiple reports from different sources should be merged. Merging of device tuples may be useful for deriving presence information as described above in Section 5.2. Merging of service tuples is less useful.
In any of the above cases, the elements in the resulting tuple must be based on the original tuples. Although the original values should not conflict, following the previous step, some elements will have multiple non-conflicting values. For example, person tuples will be merged which contain elements which are treated as non-conflicting, such as <activities> and <mood>, as described above. These values should be unioned.

6 Composition Policy Format

We define here an XML format for specifying a policy for composition. The aim is to define the most common manipulations of presence data so that a user's policy may be defined fairly easily. Full programming solutions may give more exact results, and could complement this approach. Standard composition documents would likely be created by network administrators. More advanced users could define policies by themselves, as they are expected to do with authorization rules in SIP [8] or with mail filtering scripts using Sieve [17]. Graphical user interfaces could also help users in a way similar to their usage in defining customized services in multimedia communications [18].

The document is a sequence of composition steps, each with its own options for customization. The steps are "discard", "derive", and "resolve-conflicts", each represented by an XML element, with sub-elements to define the processing done in that step. As we mention below, the merging step does not require user customization, and it is therefore absent from this format.

6.1 Discard Step

This step allows for discarding of tuples. Three types of discarding may be specified: discard all service tuples with closed contacts, all tuples whose timestamps are older than a certain amount of time, and all device tuples not associated with a service.

6.2 Derive Step

This step contains rules for deriving new information based on existing information. The XML Patch format [6] is generally used to express changes to XML content, such as adding or removing elements. The <add> element contains a "sel" XML element contains an XPath [13] expression which identifies the location where the content is to be added. The XML content under the <add> element is the content to be added. We use this format to express the derivation of new content. For example, the following Patch operation:

```
<add sel=
   '//person[place-type/car]'>
   <activities>
     <driving />
   </activities>
</add>
```

adds the 'driving' activity directly under a <person> element of a tuple that contains a <place-type> element that contains <car>.

In order to make derivation dependent on the time of day, the selecting Xpath expression may refer to the tuple's timestamp in the predicate. Functions built into Xpath 2.0 may be used to retrieve the desired part of the date/time expression. For example, if someone sleeps between the hours of midnight and 7 am unless he is working on a deadline, a derivation of his sleep based on his user- input may be expressed as follows:

```
<add sel= \
  '//person[user-input="idle"]\
  [fn:hours-from-dateTime(timestamp)
      > 0 \
      and \
  fn:hours-from-dateTime(timestamp)
      < 7]'>
   <activities>
     <sleeping>
   </activities>
</add>
```

This states that if the user-input is 'idle' during normal sleeping hours, the user is sleeping. If the value is not 'idle' during those hours, he is likely working on a deadline.

6.3 Resolve Conflicts Step

In this step, conflicts are identified and resolved using one of a number of policies. Identifying conflicts is a matter of local policy and we do not consider it to be something that users should specify.

The <resolve-conflicts> element contains possibly several <conflict> elements, each defining how conflict is to be resolved. An "element" XML attribute may be included so that the included policy applies only to that element. When this attribute is omitted, it applies to all elements.

Options for resolution are "merge", "union", "most-recently-published", "source-precedence", or "other-attribute". Several policies may be listed, and conflict resolution is attempted. <merge> is not a conflict resolution, per se, but, in effect, defines the given element as non-conflicting. Examples of elements appropriate for this are <activities> and <mood>. The use of <merge> for a given element precludes any other conflict resolution policy for that element.

Choosing "union" causes both conflicting tuples to be included, and precludes any other policy for conflict resolution for the specified elements. It also ensures that the two tuples will remain distinct, even after the merging step, so that multiple versions will be represented, and the human watcher will be able to decide which is more likely to be accurate. This is the default value for the resolution of a conflict for any given element when an alternative policy is not given.

The <most-recently-published> element directs the compositor to choose the tuple which was most recently published for the first time. This does not choose a tuple simply because it was refreshed more recently.
The <source-precedence> element lists a number of source types. This list may contain any of the following tokens at most once: "reported current", "reported scheduled", "measured device information", "measured by sensors". If each of the conflicting tuples is from one of the sources listed, the one with a higher value is chosen. If only one of the tuples is from a source with a listed value, that one is chosen. If neither of them are, the conflict is not resolved by this method.
The <other-attribute> element specifies that resolution be done based on another element besides the one in conflict. An XML attribute is included to specify the element. A list of elements gives the ordered preference of various values.

6.4 Merging Step

This final step merges multiple tuples to present a final view of the user's presence before continuing to later steps such as privacy filtering. We currently consider only merging of person tuples as this is the most likely to be useful.
When multiple tuples are merged, they may have different values for the same attribute. The conflict resolution step is used to declare for which elements, such as <activities> multiple values should be listed, rather than be treated as conflicting. Therefore, no real specification is required by the user in this step for person tuples. It is expected that for the merging of service tuples, input from the user will be desired regarding whether to merge them and, if so, how to handle multiple values of elements.

6.5 Example

We provide an example use case to help motivate the need for our composition steps. Bob is sitting in his office, waiting to attend a meeting in the conference room. He receives a call about a very important matter to which he must attend. He calls up the other meeting participants to tell them, then rushes out of the office, without logging off of his PC, proceeds to his car and drives away.

Bob now has three different accounts of his place-type that a watcher may see. His office PC indicates that he is in the office, his cellular phone indicates that he is in the car, and his schedule indicates that he is in the conference room. He is currently in the car, but there is no indication that he is driving. The following shows an abbreviated example of a composition format that Bob may have defined or that may have been provided by the network administrator.

```
<discard>
  <old-tuples \
       age="00:30:00.000" />
  <tuples-with-closed-contacts />
</discard>
<derive>
   <add sel= \
      '//person[place-type/car]'>
     <activities>
       <driving>
     </activities>
   </add>
</derive>
<resolve-conflicts>
   <conflict \
         element="activities">
     <merge />
   </conflict>
   <conflict element="place-type">
     <other-attribute \
        attribute=\
           'person/user-input'>
       <value>active</value>
       <value>idle</value>
     </other-attribute>
     <source-precedence>
       <source>
          reported current
       </source>
       <source>
          reported scheduled
       </source>
     </source-precedence>
   </conflict-element>
</resolve-conflicts>
```

The compositor, detecting that Bob is in the car, derives that his activity is "driving." Though he is separately reported as having activities "working," "in a meeting," and "driving," the compositor does not take any action to resolve this conflict, but leaves the activities to be merged at a later stage if no other conflict causes them to be removed. There is a conflict about the type of place where Bob currently is. The compositor first tries to resolve the conflict based on another attribute, user-input. Since the user's cellphone reports him as "active," the tuple published by that device is chosen over the other two. Now the conflicting activities will also not appear. A watcher of Bob's presence would now only see that he is in the car and driving.

7 Conclusion

We have discussed the need for composition in enhanced presence. This includes the need to decide between multiple, possibly conflicting presence reports. Also, deriving

new presence information is a way to offer watchers with rich, useful information without requiring the user or devices to report everything. We have provided solutions, both conceptually and in the definition of an XML format for users to define their own policies.

References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M, Schooler, E., *SIP: Session Initiation Protocol*, IETF RFC 3261, June 2002.

[2] Rosenberg, J., *A Presence Data Model*, IETF RFC 4479, July 2006.

[3] Jiang, D., Liscano, R., Logrippo, L., *Personalization of Internet Telephony Services for Presence with SIP and Extended CPL*, Computer Communications, Vol.29, No.18 (November 28, 2006), pp. 3766-3779.

[4] Lennox, J., Wu, X., Schulzrinne, H., *Call Processing Language (CPL): A Language for User Control of Internet Telephony Services*, IETF RFC 3880, October, 2004.

[5] Rosenberg, J., *A Processing Model for Presence*, IETF Internet Draft, June 2006 (Work in Progress).

[6] Urpalainen, J., *An Extensible Markup Language (XML) Patch Operations Framework Utilitizing XML Path Language (XPath) Selectors*, IETF Internet Draft, March 2006 (Work in Progress).

[7] Bergmann, O., Ott, J., Kutscher, D., *A Script-based Approach to Distributed Presence Aggregation*, In Proceedings of International Conference on Wireless Networks, Communications and Mobile Computing 2005, Maui, HI, USA, June 2005.

[8] Rosenberg, J., *Presence Authorization Rules*, IETF Internet Draft, October 2006 (Work in Progress).

[9] Rosenberg, J., *The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)*, IETF Internet Draft, October 2006 (Work in progress).

[10] Khartabil, H., Leppanen, E., Lonnfors, M., Costa-Requena, J., *An Extensible Markup Language (XML)-Based Format for Event Notification Filtering*, IETF Internet Draft, September 2006 (Work in Progress).

[11] Isomaki, M., Leppanen, E., *An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Manipulating Presence Document Contents*, IETF Internet Draft, October 2004 (Work in Progress).

[12] Schulzrinne, H., Gurbani, V., Kyzivat, P., Rosenberg, J., *RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)*, IETF RFC 4480, July 2006.

[13] *XML Path Language (Xpath) 2.0*, W3C Candidate REcommendation 8 20060608, June 2006.

[14] Day, M., Rosenberg, J., Sugano, H., *A Model for Presence and Instant Messaging*, IETF RFC 2778, February 2000.

[15] Rosenberg, J., *A Presence Event Package for the Session Initiation Protocol (SIP)*, IETF RFC 3856, August 2004.

[16] Luther, M., Mrohs, B., Wagner, M., Steglich, S., Kellerer, W., *Situation Reasoning-A Practical OWL Use Case*, In Proceedings of 7th International Symposium on Autonomous Decentralized Systems (ISADS 2005), Chengdu, China, April 4-8, 2005, pp. 461-468.

[17] Showalter, T., *Sieve: A Mail Filtering Language*, IETF RFC 3028, January 2001.

[18] Wu, X., Schulzrinne, H., *Location-based Services in Internet Telephony*, In Proceedings of IEEE Consumer Communications and Networking Conference (CCNC'05), Las Vegas, NV, USA, Jan. 2005