The SIMPLE Presence and Event Architecture

Henning Schulzrinne
Dept. of Computer Science
Columbia University
hgs@cs.columbia.edu

***Abstract*— Current presence systems offer only basic services that are mostly useful for dial-up environments, while event services have not seen widespread deployment. We describe the SIP-based event architecture that is being developed within the SIMPLE working group, that supports richer functionality, multiple event sources and better privacy control.**

I. Introduction

Presence applications, usually bundled with instant messaging (IM), have been around since the mid-1990s, popularized by companies such as AOL, Yahoo! and MSN, with little recent advances. Presence applications indicate the availability of human users for communications, following a publish-subscribe-notify paradigm. For these IM clients, presence was necessary as users were available only intermittently, constrained by dial-up connections to the Internet. With the advent of broadband and wireless always-on connectivity, the notion of "on-line" and "off-line" has largely lost its meaning as users will be technically on-line all the time. There is an opportunity to re-think presence as an instance of a more general event notification framework, adding crucial functionality to the current model. In particular, emerging presence applications distinguish themselves from their predecessors by a number of features:

- They use standardized message formats such as SIP/SIMPLE [1, 2] and XMPP/Jabber [3], facilitating inter-domain communication rather than being restricted to a single provider.
- They facilitate not just text messaging, but also replace the classical ring-and-hope cycle that has defined telephony applications for 125 years (and the ring-and-voice mail cycle of the past 20 years).
- Some offer a generic event notification mechanism, where human presence is just one of many event types. There has been a large volume of work on generic event notification systems, but none have seen widespread deployment outside closed environments, as they tended to be suited only for intra-enterprise applications.
- They provide "rich" presence, reflecting user activity by more than one value drawn from half a dozen choices and allowing to represent past and future user states, as described in Section II.
- They offer far more extensive privacy controls than simple white and blacklists, with a policy language described in Section IV.

We briefly describe some of the core aspects reflecting our contributions in the sections below, focusing on the SIP-derived extensions. SIMPLE and XMPP use the term "presentity" to refer to the person whose presence is being reported and "watcher" to the person or program who is observing a presentity. Typically, each presentity is being observed by many watchers, each of which may see different degrees of detail for that presentity. A presentity is a logical abstraction and is described by data provided by any number of devices and programs that publish this information to the presence agent managing the presentity. The core prototol messages are shown in Fig. 1, where publishers use the SIP PUBLISH method to upload information, while watchers subscribe using the SUBSCRIBE method and then receive information from the presence agent (PA) asynchronously using NOTIFY messages.

The basic presence model in the IETF evolved from an attempt to separate the presence architecture [4] from the protocols carrying subscriptions and notifications to the data formats for presence objects, the so-called Presence Information Data Format (PIDF) [5], an XML schema. Initially, four proposals were competing within the IETF to become the standard protocol mechanism for presence, but in the end, only two have been fully developed, namely SIMPLE and XMPP. The SIMPLE architecture is a special application of the generic SIP event notification model [1], while XMPP is the standardized version of the protocol known generally as Jabber.

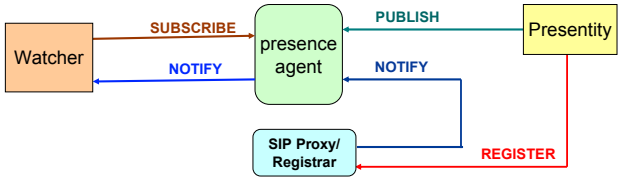

Fig. 1. Basic presence components

The presence data model [6] describes formally how presence information is published via the SIP PUBLISH request, composed from multiple sources, filtered to account for privacy considerations and the distributed to a variety of receivers, that may in turn restrict the rate or content of the notifications sent to watchers (subscribers). While the components have been mainly discussed for presence, it seems likely that most are also applicable for other types of events. Fig. 2 pictures the data flow as a presence document moves from published information provided

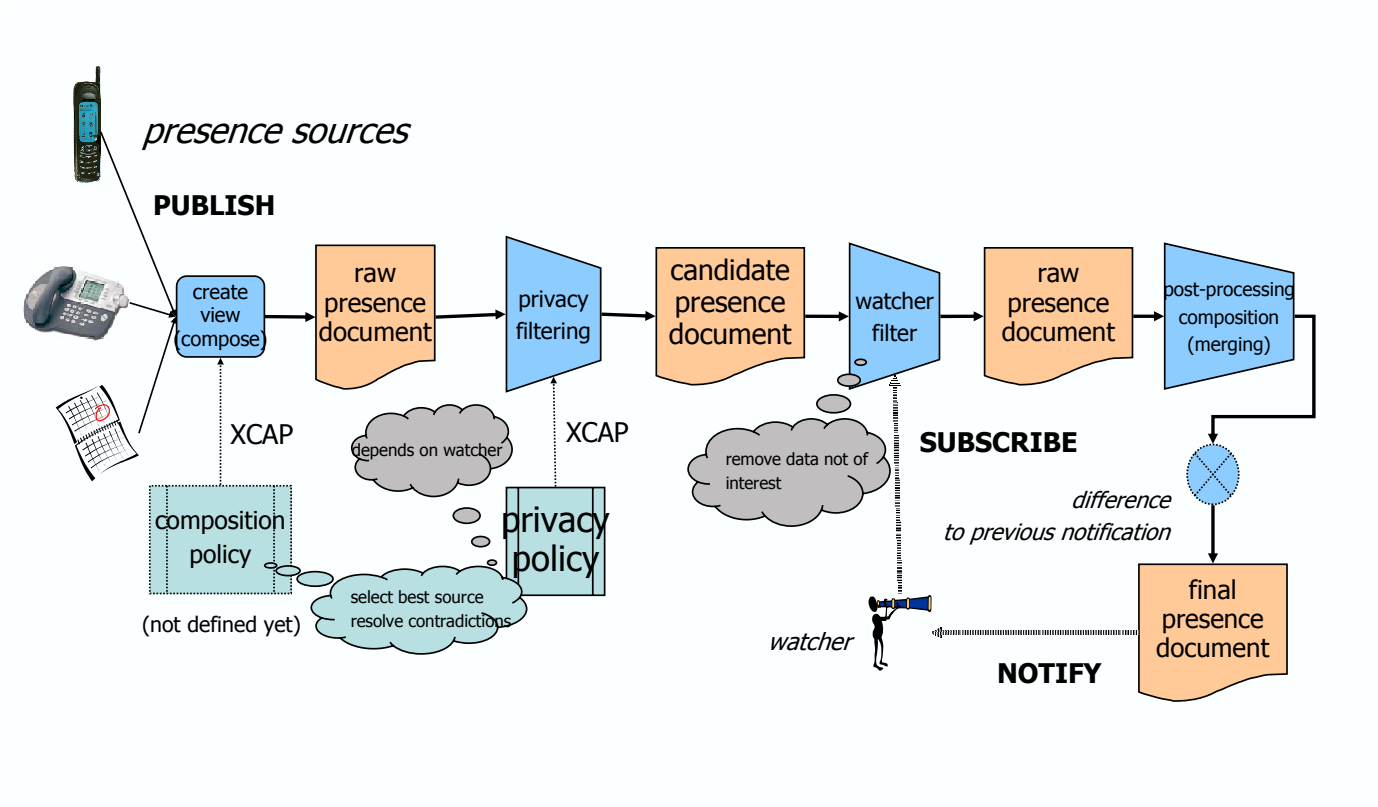

Fig. 2. Presence data model [6]

A working presence system requires more than just publish, subscribe and notify operations. Rather, users need to be able to configure the system and determine who is currently watching them, for example. Fig. 3 shows how other event packages and data formats are used to provide this information. Generally, such configuration files, such as those describing privacy policies (Section IV), are formatted as XML. XCAP, the XML configuration access protocol [19], can modify parts of XML files, so that users do not have to upload complete XML files each time they make a minor change.

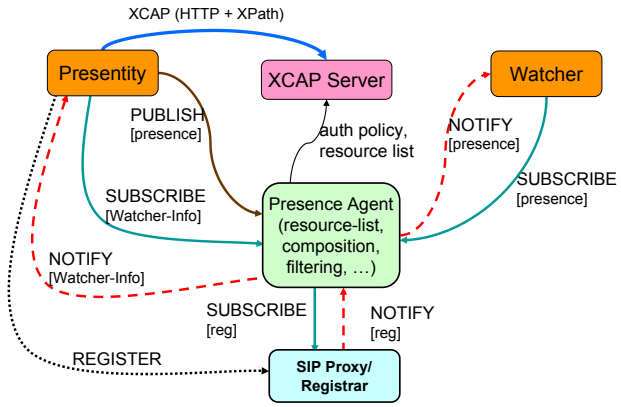

Fig. 3. Presence configuration

II. Rich Presence

Classical presence applications convey only two pieces of information: an indication of being "available" or a short list of indications of why the person is not reachable such as "busy" or on the phone, augmented by an "away" message. Users have to update the information manually, with the exception of an idle indicator based on keyboard input.

Relying on users to update the information means that most users are either permanently available, since they are too busy to fiddle with the presence agent settings, or permanently away, since they forgot to change back their status after an absence. We advocate using automated sensors and data inputs to set this information at a much finer level of granularity. Three such presence extensions have been proposed: The core rich presence XML format (RPID) [7] allows users to indicate their activities, moods, sphere of life, properties of the place they are currently located in (place type), media privacy, timezones, alternate contact points for friends and colleagues and user input. The sphere of life indicates the "hat" that somebody is wearing, e.g., "work" vs. "family" vs. "community service", as users are likely to be visible to different groups of people during each of those parts of their life. The sphere indication is primarily used to tailor the privacy permissions, but also can be useful to tailor one's conversation, e.g., to avoid private conversations if somebody is in work mode.

In some cases, it is as important to know about future

activities and status as about the most current status, motivating RPID to add support for time ranges for descriptive elements. More generally, timed presence [8] can label other status information, such as geo location [9], with a time range reflecting its predicted validity. This allows a user to make parts of their calendar visible to others, facilitating joint scheduling. Recently, a more direct way of exporting vCal information via presence has been proposed [10].

Instead of a single presence description, the SIMPLE developers noted that a single person may be using multiple communication services, some of which may have their own availability status, provided by a multitude of stationary and mobile devices [6]. For historical reasons and inherited from PIDF, services are represented by <tuple> elements. Each view of a person is encapsulated in a <person> XML element, while each device is reflected in a <device> element. Devices and services can also be annotated with capabilities [11].

Fig. 4 shows an example of a snippet from a rich presence document.

The PIDF-LO (location object) format [9, 12] can be used to encode geospatial or civic (street address [13]) information in presence objects.

III. Composition

A. Introduction

Unlike existing systems, SIMPLE allows a multitude of sources to contribute presence information for one presentity. Such sources might include data entered through an IM client, phone status, calendaring information and environmental sensors such as a BlueTooth beacon that announces to bystanders that they are in a movie theater.

However, this multitude of sources also creates the potential for conflicting information. Some conflicts cannot be resolved by rules and algorithms, so SIMPLE allows to deliver conflicting <person> elements to a watcher. In other cases, composition in the presence agent can create a likely state of the presentity and show it to watchers. Information about the source, timeliness and other rules can be used to give more credence to certain pieces of information.

Composition combines multiple presence or event sources into one view, which is then delivered, after various filtering operations, to watchers [6, 14]. Composition is required whenever there are several sources contributing information about a single presentity or event. [1]

We assume that the composition operation does not depend on the watcher identity, as there seems little functional gain by introducing per-watcher composing operations. The composed document contains the maximum set of information, i.e., no watcher can obtain more information than is contained in the composed raw presence document. (In some cases, a presentity wants to "polite block" a person by providing presence information that offers no information to the watcher, but avoids indicating that the watcher's subscription request has either not yet been processed or that it has been turned down. For those cases, a simple template that reflects a minimal PIDF document is sufficient, as it does not need to reflect presence inputs and does not change over time.)

Composition at the presence agent is just one component of providing useful and correct information to the watcher. We assume that composition is algorithmic, although manual composition by the presentity is theoretically possible. Given the automated nature of composition, there may well be situations where the best course of action is to expose the underlying data to the watcher, even though it may be contradictory. Indeed, in many cases, a mechanical composer may not even be able to detect whether information is contradictory or not.

The goals of composition are to remove information that is either stale, contradictory or redundant and to possibly generate inferred presence state. Stale information has been superseded by other, newer information. Contradictory information makes two statements about the presentity that cannot both be true. Redundant presence information provides information that is no longer of interest. For example, a presentity may decide to drop information about services whose status is closed if there are open services and may drop a service record referring to another person via a <relationship> element if the presentity itself is available. Inferred presence state uses presence elements or external information to derive new information. Location information seems particularly suitable for such inferences. For example, a location away from home might generate the activity indication 'away' or specific geospatial locations might be mapped to particular location types or activities.

Composition is not designed to reduce the size of notification messages or to protect information for privacy. Various compression schemes and partial notification [10] are better suited to reduce message sizes. Privacy filtering [8] has the role of tailoring information to individual recipients, based on the presentity's privacy policy.

In our model, the composer is reactive. In other words, it only creates a new presence document if one of the publishers updates parts of the presence document. An active composer could, for example, generate a new presence document after a certain time interval has elapsed or when timed presence information [8] is transitioning from the future to the presence.

Rather than hard-coding composition policies or forcing users to write a composition program in a general-purpose programming language, we propose to abstract composition rules into an XML-based policy language. Such a language makes it easier to predict the outcome of composition operations and makes it possible to switch servers or service providers and still maintain the same presence semantics.

B. Scoping Composition

In our model, presence takes a presence document, made up of a set of <tuple>, <person> and <device> tuples, each tuple consisting of one or more elements, and creates another valid presence document based on this information. First, we describe possible operation on these tuples, proceeding from the highest granularity to sub-element operations. In all cases,

[1] This material is based on the Internet Draft [15].

```
<dm:person id="p1">
   <rpid:activities from="2005-05-30T12:00:00+05:00"
      until="2005-05-30T17:00:00+05:00">
      <rpid:note>Far away</rpid:note>
      <rpid:away/>
   </rpid:activities>
   <rpid:class>calendar</rpid:class>
   <rpid:mood>
     <rpid:angry/>
     <rpid:other>brooding</rpid:other>
   </rpid:mood>
   <rpid:place-is>
      <rpid:audio>
         <rpid:noisy/>
      </rpid:audio>
   </rpid:place-is>
   <rpid:place-type><rpid:residence/></rpid:place-type>
   <rpid:privacy><rpid:unknown/></rpid:privacy>
   <rpid:sphere>bowling league</rpid:sphere>
   <rpid:status-icon>http://example.com/play.gif</rpid:status-icon>
   <rpid:time-offset>-240</rpid:time-offset>
   <dm:note>Scoring 120</dm:note>
   <dm:timestamp>2005-05-30T16:09:44+05:00</dm:timestamp>
 </dm:person>
```

Fig. 4. Rich presence document format

operations can be concatenation (union), merging (combining elements from several tuples into one) or selection (one-of-N).

In the tuple-level approach to composition, the integrity of individual tuples is maintained. The two possible operations are union and selection. For a union operation, tuples from different presence sources, of the same kind and with different tuple identifiers are simply all copied to the composed presence document. This is the default composition policy described in the data model documents. Also, it appears likely that <device> tuples for different devices are simple collected, possibly removing unreferenced devices, i.e., devices that are not referred to by any service (<tuple>).

For selection, only a subset of tuples for each type are copied to the composed document, with all others being discarded.

It is also conceivable that tuples are concatenated, but that elements from some of the tuples are removed, e.g., because they are considered less reliable.

For element-level operations, one can either include multiple instances of the same element type, where allowed, or create a new element that combines values from multiple elements.

Some elements can contain timing information indicating the range of time that the information is believed to be valid. It is probably not a good idea to combine elements that cover different, although maybe overlapping, time intervals.

C. Types of Information Sources

Presence information can be contributed by many different sources, either directly, by publishers using SIP PUBLISH requests or by a presence agent acting as a watcher receiving NOTIFY requests. We focus here on the semantic source of the data, i.e., how it was derived, not how it was injected into the presence system.

For simplicity, we do not try to assess the veracity of the presence document. In order to evaluate the usefulness of a presence document, we only care whether the presentity would want the information to appear that way, not whether this corresponds to observable facts. Thus, a presence document is correct in that sense if it indicates that the presentity is in a meeting even though the presentity has actually gone fishing if the presentity would like the rest of the world to believe that he is at work. It may, however, well be the case that composition policies find it easier to maintain the truth than keep lies consistent across sources of presence information.

We can distinguish the following sources of presence data: reported current, reported scheduled, measured, and derived information.

Reported current information has been provided by the presentity within processing time delays of the current time. A presentity can update status information manually, by setting any of the element in a presence document. 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.

For *reported scheduled* information, a presentity indicates its plans for the future rather than the present, e.g., in a

calendar. The reliability of this information depends largely on the diligence of the user in updating calendars and similar sources.

Measured device information uses observed user behavior on communication devices, such as the act of placing or receiving calls or typing. The main source of error is that a device may not be able to tell whether the presentity itself is using the device or some other person.

Presence information *measured by sensors* reflects the status of the presentity, e.g., its location, type of location, activity or other environmental factors. In quantum mechanical fashion, it is sometimes difficult to ascertain both the measured variable and the identity of the presentity. 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, cleaning staff or a dog. A GPS sensor cannot detect whether the cell phone is being used by the presentity or has been borrowed by the presentity's spouse.

Presence information might be *derived* indirectly from other sources of data. For example, the basic open/closed status might be algorithmically derived from a variety of other, watcher-visible or not, elements.

D. Information Conflict

The fundamental problem addressed by composition is information conflict, i.e., multiple sources (publishers) have 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, lack of update diligence and sensor failure.

Location divergence occurs if the publisher collecting the information is not be co-located 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.

Update diligence may be lacking. 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.

Sensors can fail for a variety of reasons. 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.

1) Detecting Information Conflict: For mechanical composition, we would like to be able to detect information conflicts so that appropriate processing logic can remove inaccurate information. Information conflicts can be classified as to whether they are detectable in a single element or only across elements and how easy it is to detect them.

We distinguish single-element from multi-element conflicts. Single- element conflicts occur if two elements, say <activities> in RPID, in two sources cannot both be true or are highly unlikely to be true, without having to inspect any other element. A multi-element conflict occurs if only the combination of multiple elements indicates a conflict.

Multi-element conflicts often have location, and properties known for this location, as the common element. For example, certain geospatial locations are known not to contain certain types of places. Thus, both the location and the <place-type> information are, by themselves, each credible and possible, but are detectably wrong once considered together. These conflicts can be detected if location or time can be mapped to reliable information from external sources.

We distinguish three types of information conflict: obvious, probable and undetectable, described in turn below.

For some pieces of presence information, information conflicts are obvious and readily detectable. For example, under the one-person- per-presentity assumption and common assumptions of physics, 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. In RPID, the <place-is>, <place-type>, <privacy>, <relationship>, <time-offset>, and <user-input> elements have exclusive values, although in some cases, below the element level. For example, the <privacy> field has information for both audio and video, and thus two sources may report different information for <privacy> and still both be correct as long as they refer to different media types.

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. One would hope, however, that combinations such as "steering, sleeping" are rarely true, although "sleeping, meeting" indicates that there are few activities that completely rule out others.

Thirdly, 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.

E. Composition Operations

Based on the exploration of common presence document operations earlier, we now discuss a variety of practical semantic composition operations and policies that may be

useful. They are worded as possible choices or operations that can be applied, in a roughly sensible order of execution.

1) Default Policy: Union: The default composition policy is designed to lose no information, at the expense of presenting possibly contradictory information to watchers.

This composition policy performs a union with replacement. Newly published elements replace earlier elements with the same 'id' attribute. We assume that each source chooses their own 'id' values.

Other than this, all elements are simply enumerated as is, sorted by type (person, tuple, device). Elements within the <person>, <tuple> and <device> elements are not modified at all, except possibly annotated with a source description and time stamp. This policy can also be seen as providing input to the following steps.

2) Tuple Discarding: The next steps discards whole tuples, for example closed contacts, i.e., all <tuple>s (service) with a basic status of 'closed', or old tuples, i.e., with a time stamp or time range older than a given threshold are discarded.

3) Combine Services Tuples with Identical Contacts: Next, a composer may combine all <tuple>s with identical contacts, where identical is defined for each URI schema.

4) Ambiguity Resolution: For elements in <person> or <tuple> where only one value makes sense, but there are a number of source tuples, there are a number of possible approaches, including to choose recent or trustworthy tuples, to omit contradictions, to choose by sphere, to give certain values precedence or base the decision on user location.

For the "choose recent tuple" approach, the composer chooses elements from the most recent tuple only or from tuples no more than a certain age. Simply choosing the most recently published value is likely to cause flip-flopping between dueling publishers.

Alternatively, the composer can pick values from the most trustworthy tuple, typically based on the source or type of reporting. For example, it might rank sources in the order "reported current", "measured device information", "measured by sensors", "reported scheduled", and finally "derived".

A conservative approach omits any information where two source tuples contradict each other. Only the intersection of enumerated sets, with their associated notes, is used.

Tuple values may be used to guide the composition. For example, tuples belong to a certain sphere may be given precedence. In general, tuples containing certain values may be selected over others. For examples, activity indication of "meeting" might win over "sleeping" or a tuple for the person itself over one for a relationship. More specific values may displace less specific ones, e.g., for activities or location information.

Finally, if the location information is trustworthy, it may be possible to rule out certain values, e.g., for <place-type>. However, this appears to be generally unlikely.

As long as different sources report non-contradictory information, these tuples can be merged into one tuple. Such merging hides their origin from multiple sources, which can be considered both a feature and a drawback. A composer may discard <device> tuples that are not referenced by any service <tuple>.

F. Inference

Inference adds information, rather than removing redundant or incorrect information. In some cases, it may be possible to infer information, either based on general properties of the location, likely behavior at certain times of the day or statistically-inferred behavior of the presentity. For example, the composer could deduce that the activity is "sleeping" if the time at the presentity's current location is 2 am.

IV. Privacy

A. Protecting Privacy

Presence information, and in particular rich presence, is highly privacy-sensitive. Existing systems generally only allow access control by requesting permission from watchers before they are allowed to subscribe. However, with richer presence, information needs to be more finely tailored to the recipient. Current proposals [16, 17] allow users to specify a set of rules that grant access to individuals based on their identity, the location and current sphere of the presentity, and the time of day. These rules are designed to be simple and *privacy-safe*, i.e., taking away a rule can only reduce the level of access, not increase it.

The privacy rules are designed to allow other applications to build on a common foundation, the common policy rules. Currently, there are two such extensions, namely for geospatial [18] and for presence information [16]. A rulemaker, typically the presentity, uploads the rules to the presence server, e.g., using XCAP [19]. The rules affect who can subscribe to the presentity or target, the term used in geospatial applications. The subscriber is also known as the watcher or location recipient.

The common privacy rules can be thought of as a set of database rows, each row representing one rule. Each rule consist of a set of conditions that determine when a rule "fires" and enables a set of actions, such as allowing a subscription, and permissions, namely the data that the watcher can receive.

B. Design Choices Made

Unlike for other policy systems, rules are "permit only". Rules only provide permissions rather than denying them. Allowing both 'permit' and 'deny' actions would require maintaining a fixed ordering of rules, which may be difficult to maintain or visualize when rules are updated remotely and incrementally. Because of this property, rule ordering is not important, but making a policy decision requires selecting from of all rules for a particular presentity. (Given the database implementation and indexing, implementations typically will not have to inspect each row sequentially.)

Permissions are additive. When a watcher requests permission to subscribe, for example, the presence agent selects the rules that match the watcher and other conditions and combines those rules by logically adding up the permissions. Each data type has its own rules of addition, but they generally

involve some form of set union or maximum. Typically, a small set of rules will give limited permissions to a broad set of users, while more specific rules then grant additional rights, e.g., to a particular person or during a particular time of day.

The rule sets upgradeable with later enhancements while maintaining the safety of earlier rules. Naturally, a rules engine that only understands parts of the permission may reveal less information than desired. Rules with unknown conditions never match.

The rule sets try to avoid giving users false assurance. It appears more dangerous to give the user the impression that the system will prevent disclosure automatically, but fail to do so with a significant probability of operator error or misunderstanding, than to force the user to explicitly invoke simpler rules. For example, rules based on weekday and time-of- day ranges seem particularly subject to misinterpretation and false assumptions on part of the rule maker. (For example, a non- technical rule maker would probably assume that the rules are based on the timezone of his current location, which may not be known to other components of the system.)

C. Conditions

The common policy rules allow restricting matching based on identity, sphere and validity. The identity rules support simple matching for a single authenticated individual (<one> element) or a set of individuals (<many> element), where all but a designated set of individuals are allowed. This type of blacklisting is not without problems, as it can be easily circumvented if an annoying individual can easily create new identities. For example, for email, some systems deliver email of the form 'user+'*something* to the user. An example is shown in Fig. 5.

```
<rule id="f3g44r1">
  <conditions>
    <sphere value="work"/>
    <identity>
       <many>
         <except domain="example.com"/>
         <except domain="example.org"/>
         <except id="sip:alice@bad.example.net"/>
         <except id="sip:bob@good.example.net"/>
         <except id="tel:+1-212-555-1234"/>
         <except id="sip:alice@example.com"/>
       </many>
    </identity>
    <validity>
      <from>2003-12-24T17:00:00+01:00</from>
      <until>2003-12-24T19:00:00+01:00</until>
    </validity>
  </conditions>
  <actions/>
  <transformations/>
</rule>
```

Fig. 5. Example privacy rules

D. Actions

In presence and other event systems, the system needs to decide whether to accept the subscription or not. SIMPLE can block a subscription, but also request presentity confirmation from the user. The user would be notified of a new subscriber, via the event list package [20]. The user could then add a new rule to the privacy policy that either admits or blocks the subscription.

In addition to accepting or blocking the subscription, a presentity can also politely block the new subscriber. The new subscriber gets a presence notification that contains no useful information and indicates that the presentity is unavailable. Polite blocking reduces information leakage, as the act of refusing a subscription after some time period indicates to the watcher that the human being is indeed around, as he was able to change the privacy rules.

E. Transformations

The presence privacy rules [16] specify which elements in the presence data format will be shown to a particular watcher, allowing to select devices, services and persons or subsets of that information. Fig. 6 shows a simple example that exposes "sip" and "mailto" services, all person identities, as well activities. A new attribute with the name "foo" is also delivered, so that presence data can be locally extended without having to change the presence rule definition.

```
<cr:transformations>
 <provide-services>
  <service-uri-scheme>sip</service-uri-scheme>
  <service-uri-scheme>mailto</service-uri-scheme>
 </provide-services>
  <provide-persons>
   <all-persons/>
  </provide-persons>
 <provide-activities>true</provide-activities>
 <provide-user-input>bare</provide-user-input>
 <provide-unknown-attribute
 name="new:foo">true</provide-unknown-attribute>
</cr:transformations>
```

Fig. 6. Example transformation rules

V. Presence as System Glue

Due to its interdomain nature, presence turns out to be an ideal system "glue" that allows loosely coupled systems to exchange information. For example, while networked calendaring systems exist, they have not been very successful at coordinating individuals in different organizations. One can think of presence information as an automatically updated diary and allow others to subscribe to subsets of that information, facilitating the creation of group calendars or automated meeting schedulers.

Rather than presenting presence information to human users, it may be at least as interesting to use it as input for service creation. In particular, call routing applications using feature scripting languages such as CPL and LESS [21,22] can benefit from tailoring their action to the current environment of the user. As a example of the most common routing problem today, knowledge, via rich presence, that the presentity is in a theater could cause the call routing algorithm to send all but the most urgent calls to voice mail or an assistant.

VI. The Future of Presence

While presence, either in combination with IM or voice communications, is likely to reduce the annoyance of being

inappropriately interrupted by phone calls, it has scaling problems, privacy issues and does not address calls made by strangers. Presence does not scale well as the number of watchers increases or the granularity of the presence information decreases. For example, if scheduled activities such as meetings and phone calls as well location changes trigger notifications, each presentity may well send about 50 notifications each day, as each activity may generate a start and end notification.[2] Most active users of IM/presence clients have dozens of contacts, so that a system can generate hundreds of messages each day. Almost all of these are ignored by the recipients and simply lead to a Christmas-tree-like display of red and green status lights. The wasted bandwidth is a particular concern for wireless devices.

While the privacy rules discussed earlier allow fine-grained control over who can see one's status, it does not address the more subtle privacy issues caused by the ability to observe somebody continuously and possibly mine the data for patterns. For many watchers, individuals may be comfortable with them checking in occasionally before making a call, but not to be observed in one's coming and goings throughout the day.

Finally, many calls are placed by relative strangers who are unlikely to be on the watcher list or be admitted to that list. For example, the author has a personal address book of more than 2,600 entries, mostly individuals that have received email messages. It is clearly unrealistic to distribute presence information to all of these, on the off chance that one of them may decide to call or visit.

Thus, blindly distributing presence information to a long watcher list is unlikely to be a good system design both from a network efficiency and human factors perspective. If the role of presence is to prevent disruptive communications, a polling model may be more appropriate. For example, a system could be set up so that anybody who has received email from the presentity is allowed to do take a limited number of samples per day of the user's availability without manual subscription approval. In addition, a user may permit a subscription that sends one notification message if the presentity was busy during the initial subscription or includes the predicted next availability based on calendaring information, to facilitate "make urgent call when user gets off the phone". This model can be implemented by having short-duration subscriptions, as each new subscription triggers an initial notification, and by limiting the number of such subscription renewals over time. One can imagine a phone that automatically triggers a subscription after dialing and provides the caller feedback such as "The person you are trying to reach is in a meeting until 3.30 pm. Do you want to proceed?"

VII. General Event Handling

While almost all the attention in this paper has focused on event notification applied to presence, event notification clearly has much broader applicability and can be considered one of the fundamental building blocks for distributed applications. For example, many other telephony services, such as call transfer [24] and message waiting [25], can be modeled or enhanced with event notification. It is likely that the SIP model could have been simplified if event notification had been applied even for its basic INVITE session initiation transaction, instead of using provisional (1xx) responses.

SIP-based event notification can also be used in health care [26] and for emergency alerts during natural and man-made disasters [27, 28].

More generally, Internet-scale event notification can be used for a variety of coordination tasks, including network management, workflow management and as a complement to web services. For network management, email, SNMP traps and syslog are commonly used, but have uncertain delivery delays, difficulty with structured data and limited filtering capabilities. The current approach to event notification in web services is largely based on polling, which does not work well for tasks that complete after an unpredictable delay. Compared to other publish/subscribe (event) systems such as Siena [29], SIP events do not offer the same sophisticated in-network replication services and are thus best suited for systems where most notifications are sent to a modest number of users and where notifications are not heavily tailored for different receiver subsets. On the other hand, the privacy, partial notification and authentication mechanisms for SIP events make deployment across administrative domains much more feasible.

VIII. Conclusion

Presence, viewed as a special case of event notification, can offer far more useful services than the simple IM applications do today, helping with group awareness, communication scheduling, attention management and group coordination. These additional features require, however, support for privacy policies, rich presence and automated composition. The effort in the SIMPLE working group is close to offering a comprehensive architecture for advanced presence services. We have implemented [30, 31] parts of this architecture in a communication client.

IX. Acknowledgements

As indicated by the references, the work described here is the collaboration of a large set of individuals working within the IETF SIMPLE and GEOPRIV working groups. In particular, Jonathan Rosenberg and Paul Kyzivat contributed many of the ideas.

References

[1] A. Roach, "Session initiation protocol (SIP)-specific event notification," RFC 3265, Internet Engineering Task Force, June 2002.

[2] J. Rosenberg, "A presence event package for the session initiation protocol (SIP)," RFC 3856, Internet Engineering Task Force, Aug. 2004.

[3] P. Saint-Andre, "Extensible messaging and presence protocol (XMPP): core," RFC 3920, IETF, Oct. 2004.

[4] M. Day, J. Rosenberg, and H. Sugano, "A model for presence and instant messaging," RFC 2778, Internet Engineering Task Force, Feb. 2000.

[2] Surprisingly, there does not appear to be a systematic, recent treatment of presence and instant messaging traffic characteristics, although the Hubbub work is instructive [23].

[5] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson, "Presence information data format (PIDF)," RFC 3863, Internet Engineering Task Force, Aug. 2004.

[6] J. Rosenberg, "A data for presence," Internet Draft draft-ietf-simple-presence-data-model, IETF, Feb. 2005.

[7] H. Schulzrinne, "RPID – rich presence information data format," Internet Draft draft-ietf-simple-rpid-05, Internet Engineering Task Force, Feb. 2005. Work in progress.

[8] H. Schulzrinne, "Future presence: Extensions to the presence information data format (PIDF)," Internet Draft draft-ietf-simple-future-00, Internet Engineering Task Force, Feb. 2004. Work in progress.

[9] J. Peterson, "A presence-based GEOPRIV location object format," Internet Draft draft-ietf-geopriv-pidf-lo-01, Internet Engineering Task Force, Feb. 2004. Work in progress.

[10] A. Niemi, "Session initiation protocol event packages for calendaring," Internet Draft draft-niemi-sipping-cal-events-00, Internet Engineering Task Force, July 2006. Work in progress.

[11] M. Lonnfors and K. Kiss, "User agent capability presence status extension," Internet Draft draft-ietf-simple-prescaps-ext-00, Internet Engineering Task Force, Feb. 2004. Work in progress.

[12] J. Winterbottom, M. Thomson, and H. Tschofenig, "GEOPRIV PIDF-LO usage clarification, considerations and recommendations," Internet Draft draft-ietf-geopriv-pdif-lo-profile-01, Internet Engineering Task Force, July 2005. Work in progress.

[13] H. Schulzrinne, "DHCP option for civil location," internet draft, Internet Engineering Task Force, July 2003. Work in progress.

[14] J. Rosenberg, "A processing for presence," Internet Draft draft-rosenberg-simple-presence-processing-model, Internet Engineering Task Force, July 2005. Work in progress.

[15] H. Schulzrinne, "Composing presence information," Internet Draft draft-schulzrinne-simple-composition-00, Internet Engineering Task Force, July 2005. Work in progress.

[16] J. Rosenberg, "Presence authorization rules," Internet Draft draft-ietf-simple-presence-rules, IETF, Feb. 2005.

[17] H. Schulzrinne, "Common policy," Internet Draft draft-ietf-geopriv-common-policy-04, Internet Engineering Task Force, Feb. 2005. Work in progress.

[18] H. Schulzrinne, H. Tschofenig, J. Morris, J. R. Cuellar, and J. Polk, "Policy rules for disclosure and modification of geographic information," Internet Draft draft-ietf-geopriv-policy-07, Internet Engineering Task Force, Oct. 2005. Work in progress.

[19] J. Rosenberg, "The extensible markup language (XML) configuration access protocol (XCAP)," Internet Draft draft-ietf-simple-xcap-08, Internet Engineering Task Force, Oct. 2005.

[20] A. B. Roach, J. Rosenberg, and B. Campbell, "A session initiation protocol (SIP) event notification extension for resource lists," internet draft, Internet Engineering Task Force, June 2003. Work in progress.

[21] J. Lennox, X. Wu, and H. Schulzrinne, "Call processing language (CPL): a language for user control of Internet telephony services," RFC 3880, Internet Engineering Task Force, Oct. 2004.

[22] X. Wu and H. Schulzrinne, "LESS: language for end system services in Internet telephony," Internet Draft draft-wu-iptel-less, IETF, Feb. 2005. Work in progress.

[23] E. A. Isaacs, A. Walendowski, and D. Ranganthan, "Hubbub: a sound-enhanced mobile instant messenger that supports awareness and opportunistic interactions," in *Conference on Human Factors in Computing Systems*, (Minneapolis, Minnesota), ACM, Apr. 2002.

[24] R. Sparks, "The session initiation protocol (SIP) refer method," RFC 3515, Internet Engineering Task Force, Apr. 2003.

[25] R. Mahy, "A message summary and message waiting indication event package for the session initiation protocol (SIP)," RFC 3842, Internet Engineering Task Force, Aug. 2004.

[26] K. Arabshian and H. Schulzrinne, "A sip-based medical event monitoring system," in *5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (HealthCom)*, (Santa Monica, CA), June 2003.

[27] K. Arabshian and H. Schulzrinne, "A generic event notification system using XML and SIP," in *New York Metro Area Networking Workshop 2003*, Sept. 2003.

[28] H. Schulzrinne and K. Arabshian, "Providing emergency services in Internet telephony," *IEEE Internet Computing*, Vol. 6, pp. 39–47, May 2002.

[29] A. Carzaniga, D. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, Vol. 19, pp. 332–383, Aug. 2001.

[30] X. Wu and H. Schulzrinne, "sipc, a multi-function SIP user agent," in *7th IFIP/IEEE International Conference, Management of Multimedia Networks and Services (MMNS)*, pp. 269–281, IFIP/IEEE, Springer, Oct. 2004.

[31] X. Wu and H. Schulzrinne, "Service learning and service risk management in Internet telephony," in *Conference Record of the International Conference on Communications (ICC)*, IEEE, IEEE, May 2005.