

# SIP and Presence

Hannu-Pekka Rajaniemi  
University of Helsinki  
Email: hprajani@cs.helsinki.fi

Kliment Yanev  
University of Helsinki  
Email: yanev@cs.helsinki.fi

**Abstract**—This paper describes the presence features that have been developed for SIP. The focus is on SIMPLE but other solutions are also mentioned, though not in detail. The distinguishing features of SIMPLE presence are discussed in detail. Attention is paid to the relationship and interoperation between SIMPLE and existing standards in the field, namely SIP and IMPP. The data format used in SIMPLE presence is presented in detail, with a focus on the features in it that make SIMPLE very suitable for mobile devices and equipment containing sensors that generate rich and automated presence data.

## CONTENTS

<b>I</b>	<b>Introduction</b>	1
<b>II</b>	<b>A SIMPLE History</b>	2
<b>III</b>	<b>SIP and SIMPLE</b>	2
<b>IV</b>	<b>SIMPLE and IMPP</b>	3
<b>V</b>	<b>PIDF</b>	3
V-A	PIDF Extensions . . . . .	4
V-A.1	CIPID . . . . .	4
V-A.2	User Agent Capability . . . . .	4
V-A.3	RPID . . . . .	5
V-A.4	Partial Presence (PIDF-diff) . . . . .	6
V-A.5	Timed Presence . . . . .	7
V-A.6	Location data . . . . .	8
<b>VI</b>	<b>Privacy and Security</b>	8
VI-A	Privacy . . . . .	8
VI-B	Security . . . . .	9
<b>VII</b>	<b>Implementations</b>	9
VII-A	GAIM . . . . .	9
VII-B	Cockatoo . . . . .	10
VII-C	Microsoft Live Communications Server 2005 (LCS2005) . . . . .	10
VII-D	Google talk . . . . .	10
	<b>References</b>	10

## I. INTRODUCTION

Presence information is an important part of modern digital personal communication. All the systems that used to provide just basic instant messaging functionality have moved to various extents into the area of presence. Presence, (which

might be better called pre-sense, since it gives information prior to a communication attempt) allows a user to know in advance what kind of context another user is in and what kind of communication, if any, is appropriate. Without this information, it is hard to determine whether a communication attempt will be successful. This is particularly important on mobile devices which follow the user into various situations, as well as computer-based messaging architectures where the user may or may not be at the device at the time of communication. Indeed, 30-70% of cellular phone calls (which up to now have had no presence information associated with them) fail[3]. This is an issue that can be easily resolved with a good presence framework.

Once SIP, the Session Initiation Protocol [1] had established itself as a signaling protocol, it was observed that the facilities provided by the core protocol were not sufficient for applications such as mobile devices and computer-based clients providing instant messaging and presence. SIP's presence framework was limited to a very basic status display, and its instant messaging capability was limited to whatever proprietary protocol client applications chose to use, since nothing of instant messaging was specified on the signaling protocol level. This was a big problem as the interoperability of instant messages across clients could not be guaranteed. At the same time mobile devices and complex personal computer-based clients were becoming more aware of their environment, and some of this environment information was considered useful for remotely displayed presence information. On the PC client side user expectations firmly tied services such as instant messaging, presence and the concept of a contact list, to the communication types that SIP was originally meant to handle, such as voice over IP. Success of proprietary VoIP systems such as Skype, which combines the look and feel of an instant messaging client with VoIP functionality, has shown that these features are necessary in modern communication systems. In fact, the rather current SIP client Gizmo uses XMPP to handle instant messaging and presence even though its main functionality is based on SIP. These examples showed that core SIP was not sufficient for mobile phone applications nor complex PC-based clients. Since those applications are exactly the ones that SIP was designed for, efforts were started on the project of extending SIP to provide the required extra functionality. These efforts crystallized into the Internet Engineering Task Force's SIMPLE working group[22]. SIMPLE stands for SIP Instant Messaging and Presence Leveraging Extensions and the working group's purpose is to design a set of backwards

compatible extensions to SIP. These extensions provide the capabilities of today's instant messaging and presence systems, as well as capabilities suited for mobile devices. They are also meant to integrate well with SIP.

SIMPLE, contrary to what its name implies, is composed of several loosely related parts, some implementing instant messaging services, others focusing on presence services and yet others combining the previous two in the concept of a contact list. There are also defined policies for managing the distribution of private information so that the amount of information that is published can be controlled. This paper focuses on the presence aspects of SIMPLE, the features that distinguish it from other currently available presence architectures and specifically in the context of mobile devices and rich presence information services.

The second section of this paper deals in detail with the motivation for and chain of events that led to the development of SIMPLE. The third section discusses the way SIMPLE's presence works together with and on top of the core SIP architecture. Section four describes the way the key features of SIMPLE implement the IMPP presence model and the main differences between the model and its implementation in SIMPLE. Section five goes over the format that SIMPLE uses to transport presence data and its main advantages and disadvantages as compared to other competing formats. Section six describes the complicated permission system that SIMPLE uses to limit access to presence information to precisely defined recipients as well as the security system that is used to enforce those permissions and prevent leaks of presence data to unauthorized third parties. Finally, section seven tells of the current implementations of SIMPLE (though as of the time of writing the list is rather short) and the features that they do and do not implement.

## II. A SIMPLE HISTORY

When the first version of ICQ was released by a small Israeli IT firm Mirabilis in November 1996 a new Internet hype term Instant Messaging was introduced. ICQ became quickly a worldwide phenomenon as in just six months it acquired 700,000 users. At that time it was also the sole IM software available for everyone. Competition quickly ensued; AOL Instant Messenger (AIM), MSN Messenger and Yahoo Messenger quickly entered the field and by 1998 AIM had overtaken ICQ in popularity. The fate of Mirabilis was sealed when AOL bought it and ensured a dominant market position for AIM by integrating ICQ into it.

However each of the networks used different proprietary closed source protocols and there was no interoperability between them. In 1999 MSN tried to access AIM network but AOL saw this as a security risk and blocked this quickly. After this there has been no official interoperability until October 2005 when MSN and Yahoo announced that their networks would interoperate by the summer of 2007.

As it became obvious that none of these closed source protocols would be opened for non-reverse engineered interoperability IETF took upon itself to create an open instant

messaging protocol. In 1998 IETF formed the Instant Messaging and Presence Protocol (IMPP) Working Group which was charged with the task of developing an open common standard instant messaging protocol. The initial goal of the working group was to define a definitive set of requirements for an instant messaging and presence protocol, which they did with RFC documents 2778 and 2779, dated February 2000 [17], [20].

As for working protocols the group came up with several proposals such as PRIM (Presence and Instant Messaging Protocol), APEX (Application Exchange) and SIP (Session Initiation Protocol). SIP is a signaling protocol but it was proposed that SIP could be extended to instant messaging and presence handling too. This proposal was later named SIMPLE (SIP Instant Messaging and Presence Leveraging Extensions). As it became obvious that a single working group was not enough, the IMPP working group was closed after creating an abstract model and requirements for IM protocols. Working groups were created for each proposed protocol to further their development.

Later both PRIM and APEX working groups expired as they lost their initial support. SIMPLE has prevailed and is still in active development. However the progress has been slow as propositions for standards with deadlines in May 2004 are still in development. Only parts of SIMPLE have been standardized and there is still a lot of work to be done.

Today the only competing protocol is an XML based protocol called XMPP (Extensible Messaging and Presence Protocol). XMPP was first developed by Jeremie Miller since 1998 as a protocol for the Jabber network. XMPP has since been standardized by IETF and is still in development and managed by Jabber Software Foundation.

## III. SIP AND SIMPLE

As SIMPLE is an extension to SIP, it was a critical requirement that it be backwards compatible with non-SIMPLE SIP implementations. As such it uses SIP as a transport, encapsulating presence notifications and instant messages inside SIP message bodies. One component of SIP that is extensively used in implementing presence is the SUBSCRIBE/NOTIFY [7] functionality. This functionality is used to implement the publication of presence information so that the server handles the distribution of information to all interested (SUBSCRIBED) parties.

The way SUBSCRIBE/NOTIFY works is by having a User Agent (UA) send a SUBSCRIBE request, which is a soft subscription (it has a timer and expires if it is not renewed). For the time that the subscription is active, the User Agent is sent a NOTIFY whenever the status of the entity they have subscribed to changes. This is used to push presence updates to all subscribers.

SIMPLE defines event packages for the purpose of presence notification in [8]

In theory any SIP server that supports SUBSCRIBE/NOTIFY can serve as a presence server. However, the complex privacy-management rules that SIMPLE requires

need much more than a basic SIP server can provide. Thus to make full use of the protocol, strong server-side support is absolutely necessary. Sensitive presence information should not be published if the server does not understand the value of the information and the necessity for control to its access. The SIMPLE specification defines several extensions [12], [11] to the SIP protocol that are to be supported by servers, clients and gateways to implement the full functionality of SIMPLE. Perhaps the most important of these is the capability for a client to ask a server what capabilities it supports. This allows clients to make informed decisions about what the server can do for them and what level of service they can expect. Another important extension is the extension for resource lists, which allow a server to store a contact list. This is useful for two reasons - a user's client does not need to keep track of its contact list, so they can move to another client installation without needing to transfer data; also, the client can then SUBSCRIBE to an entire contact list and be notified when any of the entities there change status. Yet another important extension is the possibility to do filtering of presence according to permission lists[10]. Both permission lists and contact lists are managed using an implementation of XML Configuration Access Protocol (XCAP) [9].

It is likely that the extensions defined in SIMPLE will be implemented in many future SIP servers due to the benefits they bring to users. Since the extensions keep compatibility with SIP servers there is no loss of functionality if a SIP server implements the extensions.

#### IV. SIMPLE AND IMPP

The IMPP protocol, defined in [17] and [20], defines a set of requirements and a basic model for instant messaging and presence services in general. While the protocol itself is not too widely used or implemented, the requirements are sensible and extensible, and thus have been the basis for several instant messaging and presence systems, including XMPP and SIMPLE.

SIMPLE implements the Common Profile for Presence (CPP) model defined in [19] and fulfills all the requirements outlined in [20]. Thus it is compatible with the IMPP model, but SIMPLE extends the basic model in quite a few ways, and thus while a SIMPLE-IMPP gateway is easy to implement, the most important features that SIMPLE adds to the field cannot be used on the IMPP side.

The Common Profile for Presence's implementation and extension are described in more detail in the chapter on the Presence Information Data Format.

#### V. PIDF

The Presence Information Data Format (PIDF) is defined in [2] and is the reference format for all IETF presence implementations. It is designed after the Common Profile for Presence (CPP) and is the common data format for all CPP compliant systems. PIDF has been designed from the very start to be extensible and flexible, and thus suitable for both current

and future presence systems. Several extensions to PIDF will be discussed below.

The PIDF covers and extends the minimal model of IMPP Presence. Each presence data entry consists of one or multiple tuples that contain a basic status and optionally an address and other data. The basic status is either open, closed, or some other externally defined value. Additionally, PIDF's extension of the IMPP model allows for a timestamp of the current information, which tells how recent the data is, and a priority assigned to each contact address to indicate the preferred method of communication.

Here is an example of a PIDF presence entry:

```
<?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>
  </tuple>
  <note>
    I'll be in Tokyo next week
  </note>
</presence>
```

## A. PIDF Extensions

PIDF is a very extensible format, and being based on XML allows the trivial addition of namespaces to the basic structure. This possibility is in very wide use within the framework of SIMPLE. The most important presence features of SIMPLE are provided by extensions to PIDF. Several of those are outlined below:

1) *CIPID*: CIPID, Contact Information in Presence Information Data Format [4] is an extension to PIDF. PIDF has only an optional `<contact>` element in which user can put a priority value and URL of the contact address CIPID adds multiple new optional elements for presenting contact information more diversely. CIPID defines a set of elements that can be easily combined into a buddy list but it does not extend the formal way of presenting the list of tuples, or devices, a presentity may have listed. Propriety instant messaging systems have had CIPID-like information in some form or another from the beginning.

The new elements in CIPID are `<card>`, `<display-name>`, `<homepage>`, `<icon>`, `<map>` and `<sound>`.

The element `<card>` has an URI that points to a business card either in vCard- or LDIF format.

`<display-name>` includes a name identifying the person that the presentity wants to be shown by the watcher user interface. Alternatively the element can include a tuple with the same purpose. The watcher can choose to either use the defined display name or use another more to his liking.

Naturally the `<homepage>` element has an URI pointed to the presentity's home page.

`<icon>` has an URI pointing to an icon image that a person wants to represent himself. Typical image formats (JPEG, PNG, GIF) are recommended for support. The watcher can choose if he wants to use `<icon>` or not.

The `<map>` element has an URI link to a map that a person has defined. The map can be either an HTML client-side image map, an image or a geographical information system (GIS) document. Again, typical image formats are recommended for support.

The `<sound>` element is used to inform the watcher that a person related to that sound has come online. The sound objects in the element can be for example of the suggested formats MIDI or MP3 that are referenced to with an URL.

2) *User Agent Capability* : [18] defines a mechanism for SIP user agents to transport their technical capabilities and characteristics to other user agents. This is done in the contact header field. As plain PIDF is a minimalistic protocol aimed for interoperability it has no SIP specific features. User Agent Capabilities Extension [5] (henceforth UACE) to PIDF introduces an extension that can convey SIP specific media feature tags. These tags contain information on what types of media in what way can a tuple handle. They can only define

the order of priority does the presentity want watchers to use his devices (a tuple per device).

UACE namespace is defined as `urn:ietf:params:ns:pidf:caps`. The new elements are: `<audio>`, `<application>`, `<control>`, `<video>`, `<text>`, `<message>`, `<type>`, `<automata>`, `<class>`, `<duplex>`, `<description>`, `<event-packages>`, `<priority>`, `<methods>`, `<extensions>`, `<schemes>`, `<actor>`, `<isfocus>` and `<languages>`. These are followed by any number of elements extended from other namespaces.

The elements are divided into two main categories: Service capability elements and device capability elements. The prior are defined under the root tag `<servcap>` and the latter under `<devcaps>`. Device capability elements are `<mobility>`, `<priority>` and `<description>`. Service capabilities are defined by all tags but the `<mobility>` tag.

Service elements `<audio>`, `<application>`, `<control>`, `<video>`, `<text>` and `<message>` define the basic services a tuple is defined to handle. All of them are booleans. The `<application>` element is for application mime types and `<control>` element is for conference controlling mechanisms etc. Other conference related elements are `<isfocus>` and `<automata>` which defines if a tuple is an automata, ie. voicemail server.

`<priority>` defines the priority of the devices defined in tuples that others should use when contacting a presentity. The tuples are also divided in two classes, personal and business by `<class>`. This combined with priority gives a watcher quickly a sense of how to contact the presentity in question. More definitions is still provided with the `<actor>` element by which a presentity can define for example principal, attendant, msg-taker, information or other values defined in SIP Media Tags and its extensions for any devices. By using these others can easily choose by which device contact the presentity at any time.

Other service elements define the duplex status of a device (full, half, receive-only, send-only), supported SIP Event packages and Method types along with supported SIP extensions and URI schemes. Also all the supported MIME types can be defined with `<type>` element and `<language>` defines the languages for communication.

The only real device capability element is `<mobility>`. This is used to define what kind of a device a tuple represents. As an example a device can be fixed if it's a desktop machine or mobile if it's a cell phone. Other device capability elements are more general ones used in both service and device capability definitions for giving a priority and a verbal description for the device in question.

Compared to basic PIDF this extension gives a richer and more flexible way of presenting presence information to others. Using UACE gives implementers an easier task of negotiating a connection with SIP as there is already information available for each device on a presentity's list of contact methods.

3) *RPID*: The Rich Presence Information Data format [14] defines a large body of properties that can be included in a presence information entry. They define logical properties such as user mood and activity, as well as physical properties such as location and environment. This information is designed with mobile devices in mind, and provides very specific information about both the device and the user. This information is managed using a sophisticated permission framework that is in fact usable for any kind of presence information.

This information is, in some ways, a fusion between presence information and data found in calendar systems. This is a powerful combination, as it provides enough flexibility to satisfy a person's presence requirements in both personal and work-related situations.

Here are a few of the properties that *RPID* provides and the capabilities and benefits of each:

- **Place-is**: This property describes the properties of the place the user is located at. This includes information about the sound, video, and text communication suitability to the situation. Level of noise affects the quality of voice communication and can be determined automatically. Also, if the place the user is at requires silence this can be communicated with a *place-is* property. Video communication might be impractical or uncomfortable, and the place may be dark or filming may be forbidden. Text entry can be uncomfortable or inappropriate, for example while driving. This property allows the user to communicate the type of communication preferred before the communication takes place. Many of these properties can be determined automatically from measurements or from other properties.
- **Place-type**: This property defines the type of place that the user is, selected from a list of possibilities. This can be determined from user input or from things like calendar data. Based on this it is possible to determine what kind of communication, if any, is appropriate for the situation. Example values include "classroom", "airport", "public-transport". The set of values is fixed but includes an "other" option with an optional text label that allows users to add more values. Additionally, *RPID* itself can be extended to add more values. This can be determined using various location-based services such as cell identification in a mobile phone.
- **Privacy**: This property describes the types of communication that are unlikely to be noticed/intercepted by others in the area. This way a potential contact can determine what method of communication to use if they wish the communication to remain private. This in turn assists the potential contact in picking a time that is suitable for the specific communication they intend.
- **Mood**: This property describes the mood that the user is in at the moment. This can be used by a potential contact to figure out how the user would react to various forms of communication in their current state. For example, if the user sets a value of "bored" or "lonely" to the mood property, it is likely that they would not mind being contacted.

On the other hand, if the user sets their mood to "angry" or "stressed" then other wishing to contact them would think about their attitude when approaching that user. This is somewhat related to the activity property below. It is possible that using biometric sensors can allow this kind of information to be determined automatically, but currently it depends on user input.

- **Relationship**: This property describes who, if not the user themselves, is likely to receive the message. If a friend or family member is using the user's mobile device or computer at the time, the value could be set to "family" or "friend". A potential contact would then be aware that someone else is likely to receive the communication and thus consider whether they want this potential recipient is capable of communicating that particular message to the intended recipient. This prevents private information from spreading to unintended recipients.
- **Activities**: This property describes what the user of the system is doing at that particular moment. The value is selected from a predefined list but may be extended using either other namespaces or the "other" value with a description attached. It is intended to communicate to a potential contact what the user is doing at the moment so that the contact can make an informed decision on whether they may interrupt. This kind of information is often seen in collaborative calendaring systems, where a user's calendar appointments are displayed to others so they can coordinate communication. Indeed, the easiest way to get this information is to use the user's calendar as a source.
- **Sphere**: This is a powerful property that allows the user to separate their life into several "spheres", for example work, hobbies, and personal time. It makes it extremely easy to define rules that only apply in some situations. For example, at work it might be desired for workmates to be able to see calendar entries, location and activity, but once the workday is over it could be desired to allow only a small set of friends and/or family to see this information. Thus, setting the sphere to "work" or "personal" would switch between these rulesets. This property is meant to describe the "role" that the person is in at the moment.
- **User-input**: This property tells the amount of time that has gone since the user last used the service. This helps determine whether a user is likely to respond to communication immediately or not.

These properties, and several others that *RPID* provides, are very well suited for systems that contain a lot of information on the user and the situation they are in, such as context-aware systems. The data aggregated from sensors, user input, logs, personal calendars and other sources can be presented in a systematic and controlled way which is both human- and machine-readable. This is an enabler for context-aware applications which can adapt themselves to the user's situation and provide the most suitable service. Mobile applications in particular are very well suited for this kind



```

    version="568">
<p:add sel="presence/note"
  pos="before">
<tuple id="ert4773">
  <status>
    <basic> open </basic>
  </status>
  <contact priority="0.4">
    mailto:pep@example.com
  </contact>
</tuple>
</p:add>
<p:replace sel="*/tuple[@id='r1230d']
  ... /status/basic/text()">
  open
</p:replace>
<p:remove sel=
  "*/d:person/r:activities/r:busy"
  ... ws="after"/>
<p:replace sel=
  "*/tuple[@id='cg231jcr']
  ... /contact/@priority">
  0.7
</p:replace>
</p:pidf-diff>

```

An updated local composition presence document after applying the patches:

```

<?xml version="1.0" encoding="UTF-8"?>
<p:pidf-full xmlns=
  "urn:ietf:params:xml:ns:pidf"
  ... version="568">

<tuple id="cg231jcr">
  <status>
    <basic> open </basic>
  </status>
  <contact priority="0.7">
    im:pep@example.com
  </contact>
</tuple>

<tuple id="r1230d">
  <status>
    <basic> open </basic>
  </status>
  <ci:homepage>
    http://example.com/~pep/
  </ci:homepage>
  <ci:icon>
    http://example.com/~pep/icon.gif
  </ci:icon>
  <ci:card>
    http://example.com/~pep/card.vcd
  </ci:card>

```

```

  <contact priority="0.9">
    sip:pep@example.com
  </contact>
</tuple>

<tuple id="ert4773">
  <status>
    <basic> open </basic>
  </status>
  <contact priority="0.4">
    mailto:pep@example.com
  </contact>
</tuple>

<note xml:lang="en">
  Full state presence document
</note>

<dm:person id="p123">
  <r:activities>
    <r:on-the-phone/>
  </r:activities>
</dm:person>

<dm:device id="u600b40c7">
  ...
</dm:device>
</p:pidf-full>

```

PIDF-diff is not automatically usable by systems that implement the basic CPP (Common Profile for Presence) which defines PIDF. PIDF-diff will not cause any interoperability problems as servers and gateways can simply choose not to use it if PIDF-diff is not implemented.

5) *Timed Presence*: Normally one thinks of presence information as a description of a current status of a user. This certainly makes sense for most applications of presence and is in fact the way presence is implemented in pretty much every other presence system in existence. However, there are applications where it makes sense to express more than just the present (pun not intended) state. If one thinks of calendar systems, it is common for calendar entries to indicate what a user will be doing, when, and for how long. Such a system might also indicate the recent history of that user's events. As mentioned above, the SIMPLE presence framework is a fusion of sorts between calendaring systems and traditional presence. As such, it is useful for it to include a method of expressing past intervals, durations, and future intervals in relation to a presence property. This makes most sense for properties such as activity which are often scheduled and known in advance, but its use is conceivable for nearly all the other properties allowed by PIDF and RPID. This method is defined in [16], the Timed Presence extension to PIDF. The validity interval information can define either a past or a future time interval

which is defined to start at a given time and optionally also to end at a given time. If the ending time is not given, the interval is not allowed to cover the present. Time periods may overlap each other, though that is undesirable since it can be confusing. However, since overlapping appointments are a reality, this confusion is just a reflection of the real world and cannot be avoided.

Using this data, it is possible to make an informed decision on how long the user will be unavailable and thus schedule a suitable time for contact. Also, it is possible to guess the current status of a user from past data even if current data is not available.

Here is what a timed presence element looks like:

```
<presence xmlns=
"urn:ietf:params:xml:ns:pidf"
  xmlns:xsi=
    "http://www.w3.org/2001/
      XMLSchema-instance"
  xmlns:ts=
    "urn:ietf:params:xml:ns:
      pidf:timed-status"
  xsi:schemaLocation=
    "urn:ietf:params:xml:ns:
      pidf pidf.xsd
  urn:ietf:params:xml:ns:
      pidf:data-model
      data-model.xsd
  urn:ietf:params:xml:ns:
      pidf:timed-status
      timed-status.xsd"
  entity=
    "pres:someone@example.com">

  <tuple id="c8dqui">
    <status>
      <basic>open</basic>
    </status>
    <ts:timed-status
      from=
        "2005-08-15T10:20:00.000-05:00"
      until=
        "2005-08-22T19:30:00.000-05:00">
      <ts:basic>closed</ts:basic>
    </ts:timed-status>
    <contact>
      sip:someone@example.com
    </contact>
  </tuple>
  <note>
    I'll be in Tokyo next week
  </note>
</presence>
```

6) *Location data*: While not specifically part of the SIMPLE work, a related working group called GEOPRIV has developed an extension to PIDF for transporting location data [15]. Location data is likely to become more important and pervasive in the future and it makes sense to define a way of its expression since it is closely related to the properties defined above. Also, location data can facilitate location-based services, can be gathered automatically with no effort from the user, and is a valuable bit of information in determining the situation the user is in. On the other hand, it is very sensitive information that should be strictly controlled. Indeed, the GEOPRIV working group [21] has defined a set of requirements [13] for services that carry sensitive presence data.

## VI. PRIVACY AND SECURITY

When Internet services were first designed the entire system was based on trust. At this point it is clear that universal trust is not applicable to open networks and thus the design of all new protocols must take data security into account. The concept of security is to prevent unauthorized access to both information and the facilities for changing it. Thus the communication is guarded from third parties.

Privacy management deals with a different problem. Instead of trying to prevent third parties from accessing the information, privacy management is the method of controlling how much information is shared among already authenticated and thus trustworthy contacts. This is a big issue for presence information since the data formats defined above allow for the specification of information that is extremely private and not suited for indiscriminate sharing. The feeling of invasion of personal space can completely put people off from using powerful presence tools such as SIMPLE. Thus the amount of data distributed and the entities that it is distributed to must be very strictly controlled.

### A. Privacy

SIMPLE follows the requirements outlined in the GEOPRIV specification [13] for the kind of privacy rules presence systems must support. The key concept is that of a rule set containing rules for what information may and may not be published to a specific entity. The rules are combined such that the maximum visibility of all the rules applies. A rule can then define a transformation that changes the visibility of data, or cause the subscription to the presence service to be denied entirely, or cause the server to accept the subscription and generate a presence document that tells the requesting entity that the information is unavailable. The rules are matched according to authenticated addresses, so a server can know that another server has authenticated whomever is requesting the information.

The ruleset [10] has one of three possible values for each address, that determine whether subscriptions from that address are to be accepted. These are considered before any transformations take place. The subscription can either be

blocked, confirmed, allowed or polite-blocked. Polite-block means that the subscription is accepted and then a nearly empty presence document is sent to the requesting entity, containing only the message that the information is not available. If the subscription is blocked, all subscriptions from that address are rejected. A confirmed subscription means that the subscription is put in a pending state and the user's input is required to proceed. An allowed subscription means that the system responds to the subscription and sends presence data according to the transformation rules outlined below. Block is the default value. Interestingly, blocking a subscription in one rule does not guarantee that no data will be sent there. If there are several rules about a particular address, the maximum permissions that they give will be chosen. Thus if the same address has three separate rules, one for allow, one for block and one for confirm, the maximum of them, in this case allow, will be used.

Once the subscription has been established, transformation rules come into play. There are transformation that toggle the visibility of groups of properties and also the individual properties within them. For each property defined in PIDEF, there is a value that can be set to true or false in a rule. If set to true, it allows the particular property to be seen by those matching the rule. If set to false, it deletes the property from the presence document if it is present. The exception is the user-input property, where it is possible to set it to show no data, all data, just active/idle status or active/idle status and the idle threshold. A true/false value can also be assigned to the visibility of other elements defined in other namespaces, so the permission system is usable without modification for control of the visibility of other properties. These could include, for example, location data, information on nearby people etc. Further extensions to the scale of the presence framework do not need to coordinate changes with the permission system to be able to use it. However, filtering rules that actually alter data need to be implemented at another level. One workaround is to provide several levels of detail for data as separate elements, and then only give access to one of them. This could be useful in location information, where the precision of the location shown is determined by the level of trust. This problem is being worked on in the GEOPRIV IETF working group.

The ruleset is edited using the XML Access Configuration Protocol defined in [9]. The protocol defines methods of adding, altering and removing rules. The rules are meant to be stored on a server that would do the filtering, but that is not required in the specification. Since some servers might not support filtering, there are two internet drafts that define ways of asking a server what its capabilities are. [12], [11] This is of great importance since if a server does not support privacy management the client must be aware of this and not publish any information that is sensitive.

## B. Security

Unlike privacy, which is managed mostly manually and defined using individual rules for known contacts, security features must work transparently in the background. It is

security that ensures the rules are enforced between contacts and that third parties cannot access nonpublic information nor forge information about another or deny others service. The SIMPLE working group has taken great care that no security issues remain unresolved. Every single publication has a carefully written section on security. The most important of these in the context of presence (that most other documents directly reference) are section 9 in [8] and section 6 in [2].

The latter defines the Presence Information Data Format, and imposes requirements on those that implement it. Every implementation of RPID is expected to support S/MIME for the encryption and integrity verification of sensitive information. Additionally, a behavior is defined for situations where an old timestamp is encountered, thus preventing replay attacks.

In the former document, the SIP event package for presence, critical protocol-level issues are resolved. The issue of confidentiality, that is that the user does not have to reveal who they are subscribed to or who is subscribed to them, has been resolved by requiring hop by hop encryption and end to end encryption. TLS is the recommended protocol for hop by hop encryption, and S/MIME for end to end encryption. The authenticity of a subscriber can be verified using HTTP Digest, which all clients are required to implement. TLS and SIPS provide integrity and authenticity services. Additionally, S/MIME can be used to secure SUBSCRIBE and NOTIFY messages. The document also recommends proxies authenticate their users to prevent network abuse and spam and verify their identity. Replay attacks are preventable by using timestamps within a signed (S/MIME) message. This prevents denial of service attacks such as a constant retransmission of an offline status, which can prevent a user from being reachable. Denial of service attacks that use subscription floods can be prevented by removing subscriptions that are unacknowledged or not desired. Also, since authorization policies usually filter out requests from unknown users, a subscription flood will not work unless the target is authorized to access many people's presence information. Denial of service attacks towards a server can be prevented using either a 4-way handshake or the 503 back-off response code. Both of these mechanisms are part of SIP and thus defined in [1].

Since most of SIMPLE is still at the Internet draft level at the time of writing, new security and privacy considerations might emerge. However, given the current track record of the SIMPLE working group, they are likely to be noticed, documented and resolved rapidly.

## VII. IMPLEMENTATIONS

### A. GAIM

As a Google Summer of Code project, the popular IMP client GAIM [25] gained basic support for SIP/SIMPLE thanks to Thomas Butter's effort. At the time of writing, basic status setting and instant messaging is supported, and the client can communicate with other popular SIP clients over a standard SIP server. It is unknown whether the more advanced features of SIMPLE will be supported in the future. The current SIP/SIMPLE support will be included in the 2.0.0 version

of GAIM. Not much can be said about it since it is not yet in wide use, and the lack of any servers that fully support SIMPLE makes it hard to evaluate.

### B. Cockatoo

Cockatoo [24] is a Google Summer of Code project. It implements a SIP/SIMPLE extension to Mozilla Thunderbird that enables users to phone other users by one click from the address book or to reply to e-mails by phone. Calls can be made both to the normal network and to sip addresses. This is done by implementing adding a sip address to address book cards. Presence information is transported as defined in [8] (SUBSCRIBE / NOTIFY - extensions). Later on SIMPLE features such as MESSAGE will be implemented too.

### C. Microsoft Live Communications Server 2005 (LCS2005)

Microsoft released Live Communications Server 2005 this year. LCS05 is aimed for corporations and is a multi-purpose communications server that supports audio, video and text conferencing and instant messaging. LCS05 is interoperable with all the major IM networks including MSN, AOL and Yahoo! IM. It uses the Microsoft RTC platform which uses SIP/SIMPLE protocol for instant messaging. Unfortunately, the protocol is extended in incompatible ways. RTC is used by all Microsoft real-time communication applications such as Messenger and Office Live Meeting. [23]

SIMPLE is not yet finished. For example buddy list management has not yet been developed fully and this has led to the problem with LCS05 and RTC: Microsoft has extended SIMPLE with their own protocol payloads and exchange sequences. This means that even if LCS05 is SIP/SIMPLE based if you want to create truly LCS05 interoperable clients you have to use the RTC Client API that hides the modified SIP protocol stack by forcing development and connectivity through a higher-level approach. In practice this affects also the server side and LCS05 is not directly interoperable with other SIMPLE implementations as the protocol exchange between servers has been modified. In fact, the extensions break compatibility even if supported, so a client has to treat a LCS05 server differently from a normal SIP/SIMPLE server.

### D. Google talk

Google has stated that their IMP client Google Talk [26] will support SIP/SIMPLE. However, as that implementation is still forthcoming, nothing can be said about it at the time of writing.

## REFERENCES

- [1] Rosenberg, J., Schulzrinne, H., Camarillo, H., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [2] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.
- [3] Helsinki University Context Project website, url:<http://www.cs.helsinki.fi/group/context/>, ref. November 2005.
- [4] Schulzrinne, H., "CIPID: Contact Information in Presence Information Data Format", draft-ietf-simple-cipid-04 (work in progress), June 2005.
- [5] Lonnfors, M., Kiss, K., "User Agent Capability Extension to Presence Information Data Format", draft-ietf-simple-prescaps-ext-05 (work in progress), October 2005
- [6] Lonnfors, M., Leppanen, E., Khartabil, H., Urpalainen, J., "Presence Information Data format (PIDF) Extension for Partial Presence", draft-ietf-simple-partial-pidf-format-05, October 2005
- [7] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.
- [8] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.
- [9] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-ietf-simple-xcap-03 (work in progress), July 2004.
- [10] Rosenberg, J., "Presence Authorization Rules", draft-ietf-simple-presence-rules-02 (work in progress), February 2005.
- [11] Rosenberg, J., "An Extensible Markup Language (XML) Representation for Expressing Policy Capabilities", draft-ietf-simple-common-policy-caps-00 (work in progress), July 2005.
- [12] Rosenberg, J., "An Extensible Markup Language (XML) Representation for Expressing Presence Policy Capabilities", draft-ietf-simple-pres-policy-caps-00 (work in progress), July 2005.
- [13] Schulzrinne, H., "A Document Format for Expressing Privacy Preferences", draft-ietf-geopriv-common-policy-05 (work in progress), July 2005.
- [14] Schulzrinne, H., "RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)", draft-ietf-simple-rpid-09 (work in progress), September 2005.
- [15] Peterson, J., "A Presence-based GEOPRIV Location Object Format", draft-ietf-geopriv-pidf-lo-03 (work in progress), September 2004.
- [16] Schulzrinne, H., "Timed Presence Extensions to the Presence Information Data Format (PIDF) to Indicate Presence Information for Past and Future Time Intervals", draft-ietf-simple-future-04 (work in progress), June 2005.
- [17] Day, M., Rosenberg, J., and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000
- [18] Schulzrinne, H. Rosenberg, J., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.
- [19] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.
- [20] Day, M., Aggarwal, S., Mohr, G., and J. Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779, February 2000.
- [21] IETF GEOPRIV Charter, url:<http://www.ietf.org/html.charters/geopriv-charter.html>, referenced November 2005.
- [22] IETF SIMPLE Charter, url:<http://www.ietf.org/html.charters/simple-charter.html>, referenced November 2005.
- [23] Microsoft RTC Description, url:[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rtrcclnt/rtrc/client\\_settings.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rtrcclnt/rtrc/client_settings.asp), referenced November 2005.
- [24] Cockatoo project, url:<http://cockatoo.mozdev.org/>, referenced November 2005.
- [25] The GAIM project, url:<http://gaim.sf.net/>, referenced November 2005
- [26] The Google Talk developer FAQ, url:<http://www.google.com/talk/developer.html>, referenced November 2005