SIMPLE
Internet-Draft
Expires: March 29, 2005

J. Rosenberg
dynamicsoft

September 28, 2004

A Data Model for Presence
draft-ietf-simple-presence-data-model-00

Status of this Memo

Copyright Notice

Abstract

This document defines the underlying data model and data processing operations used by Session Initiation Protocol (SIP) for Instant Messaging Leveraging Presence Extensions (SIMPLE) presence agents. The data model provides guidance on how to map various communications systems into presence documents in a consistent fashion. The data processing operations described here include composition, privacy filtering, and watcher filtering.

Table of Contents

1. Introduction

Presence conveys the ability and willingness of a user to communicate across a set of devices. RFC 2778 [1] defines a model and terminology for describing systems that provide presence information. RFC 3863 [3] defines an XML document format for representing presence information. In these specifications, presence information is modeled as a series of tuples, each of which contains a status, communications address, and other markup. However, neither specification gives guidance on exactly what a tuple is meant to model, and how to map real world communications systems (and in particular, those built around the Session Initiation Protocol (SIP) [4]) into a presence document.

In particular, several important concepts are not clearly modeled or well delineated by RFC 2778. These are:

Service: A communications service, such as instant messaging or telephony, is a system for interaction between users that provides certain modalities.

Device: A communications device is a physical component that a user interacts with in order to make or receive communications. Examples are a phone, PDA or PC.

Person: A person is the end user, and for the purposes of presence, is characterized by states, such as "busy" or "sad" which impact their ability and willingness to communicate.

This specification defines these concepts more fully by means of a presence data model, and concretely defines how to take real world systems and map them into presence documents using that model.

Furthermore, in SIMPLE systems, presence documents are processed extensively by presence user agents, presence agents, and watchers. Other specifications, such as the presence event package [5] and the PUBLISH method [12] document the protocol interfaces for moving presence documents between these entities. However, none of these

specifications define the behaviors these elements can exhibit in terms of processing those documents. This specification defines those procedures, including composition, privacy filtering, and watcher filtering, in more detail.

2. Definitions

This document makes use of many new terms, which are defined here.

Device: A device models the physical environment in which services manifest themselves for users. Devices have characteristics which are useful in allowing a user to make a choice about which communications service to use.

Service: A service models a form of communications that can be used to interact with the user.

Person: A person models the human user and their states that are relevant to presence systems.

Presentity: A presentity combines devices, services and person information for a complete picture of a user's presence status on the network.

Presentity URI: A URI that acts as a unique identifier for a presentity, and provides a handle for obtaining presence information about that presentity.

Data Element: One of the device, service, or person parts of a presence document.

Composition: The act of combining a set of presence and event data about a presentity into a coherent picture of the state of that presentity.

Raw Presence Document: The result of an initial composition operation, before privacy and watcher filtering operations have been applied.

Collection: The process of obtaining the set of event state that is necessary for performing the composition operation.

Merging: Merging is an operation that allows a presence server to combine together a set of different services or devices into a single composite service or device.

Privacy Filtering: The act of applying permissions to a presence document for the purposes of removing information that a watcher is not authorized to see.

Watcher filtering The act of removing information from a presence document at the request of a watcher, to reduce the information

flowing to that watcher.

Pivot: A presence attribute used to select a set of services or devices that are to be combined as part of a composition operation.

View: A view represents a stream of presence documents generated by a presentity after composition and authorization policies have been applied. Depending on how these policies are structured, each watcher to a presentity may get a different view, or they may all get the same view.

Status: Dynamic information about a service, person or device.

Characteristics: Static information about a service, person or device. Useful in providing context that identifies the service or device as different from another service or device.
A service or characteristic. It represents a single piece of presence information.

Publication: The act of pushing a piece of event state, including presence, to a state agent, such as a presence server.

Back end subscription: A subscription made from a state agent, such as a presence server, to a source of presence, for the purpose of collecting event state in order to perform composition.

Device View: A presence document obtained by composing together services with the same value of the device ID attribute.

Presentity View: A presence document obtained by composing together all services into a single tuple.

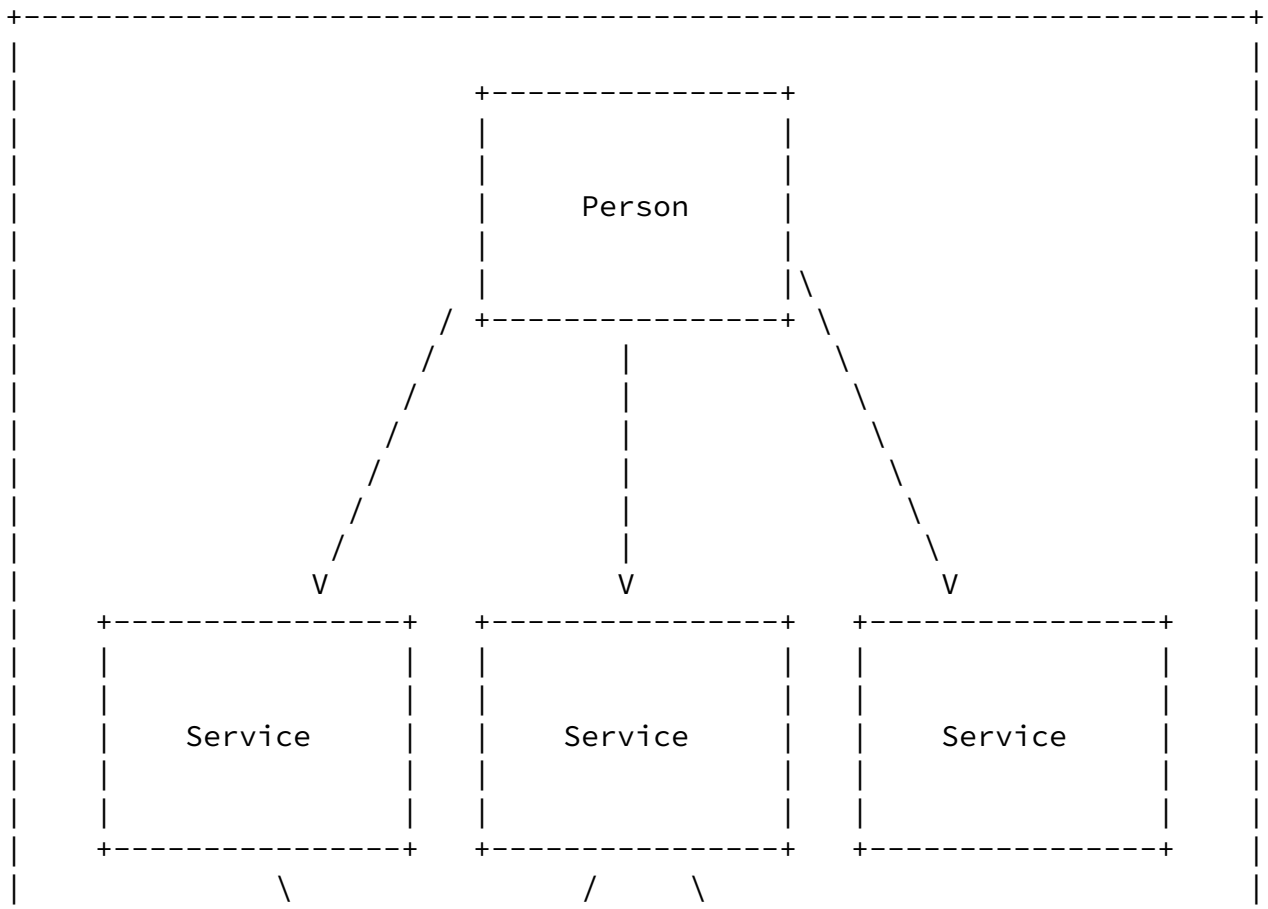
Service View: A presence document whereby the compositor has not combined together services, or it has combined them, but not used the device ID as a pivot.

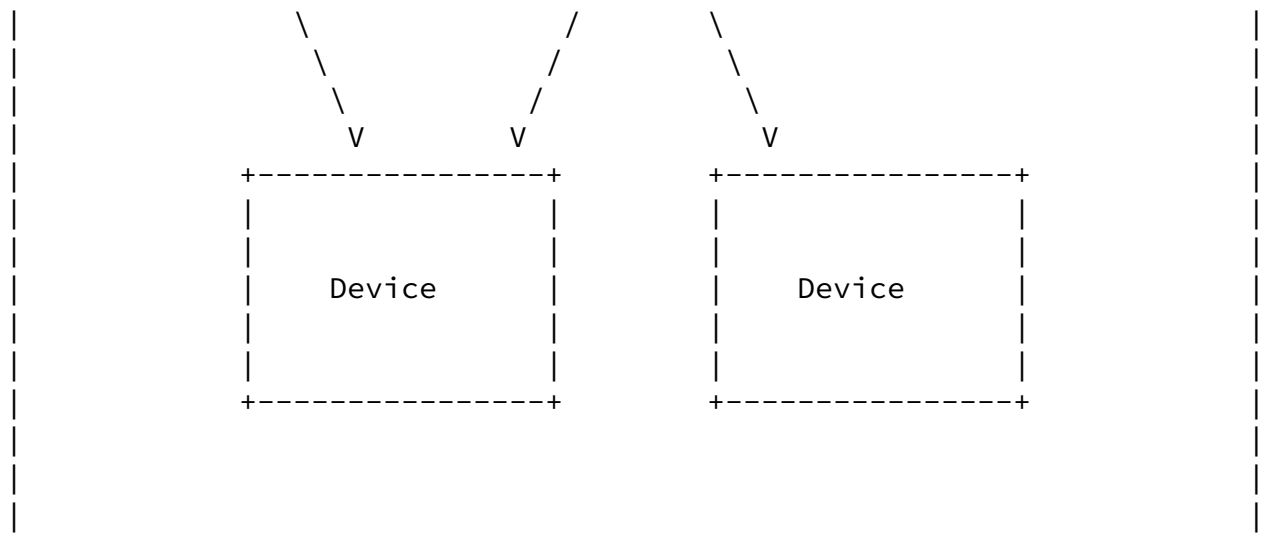
Splitting: Splitting is the process of taking a single service or device data element, and splitting into two services or devices.

Reporting: When a service or device publishes presence data about itself, it is called reporting. Reporting is in contrast to overriding, where a software agent publishes about a different service or device.

Overriding: When a service or device publishes presence information about a different service or device, in an attempt to correct or modify that data.

3. The Model


Person
Service
Service
Service


Device
Device

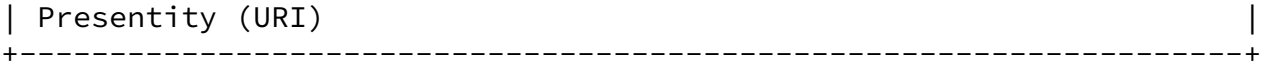


Figure 1

The data model for presence is shown in Figure 1. The model seeks to describe the presentity, which is identified by a URI, called the presentity URI. There are three elements in the model. They are the person, the service, and the device. Each of these data elements contains information (called attributes) that provide a description about the service, person, or device. It is central to this model that each attribute is affilitated with the service, person, or device because they describe that service, presentity or device. This is in contrast to a model whereby the attributes are associated with the service, presentity, or device because they were reported by that service, presentity, or device. As an example, if a cell phone reports that a user is in a meeting, this would be done by including an attribute as part of the person information, indicating a status of "in-a-meeting". The presence information may also include information on the cell phone as a device. However, even though it is that device which is reporting that the user is in a meeting, the busy indicator is not associated with the device, it is associated with the user.

The identifier for the presentity is a URI. For each unique presentity in the network, there is a unique URI. This URI is independent of any of the services or devices that they possess. However, the URI is not just a name - it represents a resource that can be subscribed to, in order to find out the status of the user. As such, it can either be a SIP URI for the user, to which SUBSCRIBE

requests can be directed, or else it can be a pres URI [10]. When the URI is a SIP URI, it will often be the address-of-record for the user, to which SIP calls can be directed. This equivalence is not mandated by this specification, but is a recommended configuration for easing the burden of remembering and storing identifiers for users.

3.1 Person

The person data element models information about the user whom the presence data is trying to describe. This information consists of characteristics of the user, and their status.

Characteristics of a person are the static information about a user that does not change under normal circumstances. Such information might include physical characteristics, such as age and height. Status information about a presentity represent the dynamic information about a user. These typically are things the *user* is doing, places the *user* is at, feelings the *user* has, and so on. Examples of typical person status are "in a meeting", "on the phone", "out to lunch", "happy" and "writing Internet Drafts". The line between static status information and dynamic status information is fuzzy, and it is not important that a line be drawn. The model does not differentiate in a meaningful way between these two types of attributes.

In the model, there can only be one person element per presentity. It is possible for the person data element to model "users" that are in fact multiple people, for example, a customer support desk. Nothing in the model mandates that the entity being modeled is actually composed of a single user. This specification only mandates that the result of the modeling activity be a single person data element, which appears to a consumer of the document to be a single user.

3.2 Service

Each presentity has access to a number of services. Each of these represent a form of communications that can be used to interact with the user. Examples of services are telephony (that is, traditional circuit-based telephone service), push-to-talk, instant messaging, Short Message Service (SMS), and Multimedia Message Service (MMS). When a service is accessible over a communications network, it is represented with a URI that can be "hit" in order to access the service. However, some services are not accessible over a communications network (such as in-person communications or a written letter), and as such, may not utilize a URI.

Each service is adorned with characteristics that describe the nature of the service that will be experienced when a watcher invokes that URI. Examples of such characteristics are the type of media that might be used, the directionality of communications that are permitted, the quality of the service, and so on. These characteristics are important when multiple services are indicated. That is because the purpose of listing multiple services in a presence document is to give the watcher a *choice*. That is, the presentity is explicitly offering the watcher an opportunity to contact them using a multiplicity of different services. To help the watcher make a decision, the presence document includes characteristics of each service which help differentiate the services from each other, and give the watcher the context in which to make a choice.

In this model, services are not explicitly enumerated. That is, there is no "service" attribute with values of "ptt" or "telephony". Rather, the service is identified in one of two ways. In many cases, the URI scheme is a clear indicator of service. An "sms" URI [19] clearly indicates SMS, and a "tel" URI [22] clearly indicates telephony [[OPEN ISSUE: Does the tel URI really indicate telephony??]]. For some URIs, there may be many services available, for example, SIP. For those services, each service has a set of characteristics, each of which has a well-defined meaning, such that a system can unequivocally determine whether or not the service has that characteristic. This is discussed in more detail in [7]. [[OPEN ISSUE: What attributes do we use to determine that the service is in-person communications or written communications?]] [[OPEN ISSUE: Do we fold roach-simple-service-features into this document?]]

One important characteristic of each service is the list of devices on which that service executes. Each device is identified uniquely by a device ID. As such, the service characteristics can include a list of device IDs. A presence document might also contain a information on each device, but this is a separate part of the document. Indeed, the information on each device might not even be present in the document. In that case, the device IDs listed for each service are nothing more than correlation identifiers, useful for determining when two services run on the same device. The benefit of this model is that information on the devices can be filtered out of a presence document, yet the service information, which includes the device IDs, remains useful and meaningful.

It is perfectly valid for a presence document to contain just a single service. This is permitted even if the presentity actually has multiple services at their disposal. The lack of multiple services in the document merely means that the presentity is not offering a choice to the watcher. In such a case, the service

characteristics are less important, but may be helpful in allowing a watcher to decide if they wish to communicate at all.

The URI is an important part of the service. When the watcher makes a decision about which service of the presentity they wish to access, the watcher invokes the URI associated with that service. It is not necessary for the watcher to add additional information to a message generated by invoking that URI in order to reach the service described in the document. Specifically, it is not necessary for a watcher to add SIP caller preferences [14] in order to request routing of the request to a service with the characteristics described in the document. As a result, each service in the presence document will have a different URI.

The URI represents a weak form of contract; the presentity tells the watcher that, if the watcher invokes the URI as included in the presence document, the watcher might be connected to a service described by the characteristics included in the presence document. It is important to stress that this is not a guarantee in any way. It cannot be a guarantee for two reasons. Firstly, the service in the document might actually be modelling a number of actual services used by the user, and it may not be possible to connect the watcher to a service with all of the characteristics described in the presence document. Secondly, the preferences of the presentity always take precedence. The caller might ask to be connected to the video service, but it is permissible to connect them to a different service if that is the wish of the presentity.

The URI also plays another important role - it acts as the unique identifier for the service. When two presence user agents publish information about a service, the service URI is what indicates that the service information they are publishing is for the same or different services. For this reason, it is important that each PUA use unique service URIs, and that these service URIs also be persistent over time. These properties are readily met by using the Globally Routable User Agent URI (GRUU) [11] as the service URI. This is discussed further below.

Each service is also associated with a priority, which represents the preference that the user has for usage of one service over another. This does not mean that, when a watcher wishes to communicate with the presentity, that they should always use the service with the highest priority. If that were the case, there would be no point in

including multiple services in the presence document. Rather, the priority says, "If you, the watcher, cannot decide which of these to use, or if it is not important to you, this is the order in which I would like you to contact me. However, I am giving you a choice." The priorities are relative to each other, and have no meaning as

absolute numbers. If there are two services, and they have priorities of 1 and .5 respectively, this is identical to giving them priorities of .2 and .1 respectively.

Each service also has a status. Status represents dynamic information about the availability of communications using that service. This is in constrast to characteristics, which describe fairly static properties of the various services. The simplest form of status is the basic status, which is a binary indicator of availability for communications using that service. Other status information might indicate more details on why the service is available or unavailable. For example, a telephony service might have additional status to indicate that the user is on the phone, or that the user is handling 3 calls for that service.

Services inherently have a lot of dyanmic state associated with them. For example, consider a wireless telephony service (i.e., a cell phone). There are many dynamic statuses of this service - whether or not the phone is registered, whether or not its roaming, which provider it has roamed into, its signal strength, how many calls it has, what the state of those calls are, how long the user has been in a call, and so on. As another example, consider an IM service. The statuses in this service include whether or not the user is registered, how long they have been registered, whether they have an IM conversation in progress, how many IM conversations are in progress, whether the user is typing, to whom they are typing, and so on.

However, not all of this dynamic state is appropriate to include within a service data element of a presence document. Information is included only when it has a bearing on helping the watcher decide whether or not to initiate communications with that service, or helping them decide when to initiate it, if not now. As an example, whether a cell phone has roamed or not does not pass this litmus test. Knowing this is not likely to have an impact on a decision to use this service.

3.3 Device

Devices model the physical operating environment in which services execute. Examples of devices include cell phones, PCs, laptops, PDAs, consumer telephones, enterprise PBX extensions, and operator dispatch consoles.

The mapping of services to devices are many to many. A single service can execute in multiple devices. Consider a SIP telephony service. Two SIP phones can register against a single address-of-record for this service. As a result, the SIP service is

associated with two devices. Similarly, a single device can support a multiplicity of services. A cell phone can support a SIP telephony service, an SMS service, and an MMS service. Similarly, a PC can support a SIP telephony service and a SIP videophone service.

Devices are identified with a device ID. A device ID is a URI that is a globally and temporally unique identifier for the device. In particular, a device ID is a URN. The URN has to be unique across all other devices for a particular presentity. However, it is also highly desirable that it be persistent across time, globally unique, and computable in a fashion so that different systems are likely to refer to the device using the same ID. With these properties, differing sources of presence information based on device status can be combined together.

Unfortunately, due to the variety of different devices in existence, it is difficult for a single URN scheme to be used. For those devices that already make use of MAC addresses as a unique identifier, the UUID URN [20] makes a good choice. It is expected that, in the future, other URN schemes will be defined that are meaningful for other device types. [[OPEN ISSUE: On more detailed read, it seems that if the UUID URN is based on MAC, it also includes a time-based random part, which would make it less useful for the requirements here. We may need a MAC URN.]]

Though this document does not mandate a particular implementation approach, the device ID is most useful when all of the services on the device have a way to obtain the device ID, and get the same value for it. This would argue for its placement as an operating system

feature, and operating system developers interested in implementing this specification are encouraged to provide APIs that allow applications to obtain the device ID. Absent such APIs, applications which report presence information about their devices will have to generate their own device IDs. This leads to the possibility that the applications may choose different device IDs, using different algorithms or data. In the worst case, these may mean that two services which run on the same device, do not appear to.

Like services and person data elements, device data elements have static characteristics and dynamic status. Characteristics of a device include its physical dimensions and capabilities - the size of its display, the speed of its CPU, and the amount of memory. Status information includes dynamic information about the device. This includes whether or not the device is powered on or off, the amount of battery power that remains in the device, the geographic location of the device, and so on.

The characteristics and status information reported about a device

are for the purposes of choice - to allow the user to choose the service based on knowledge of what the device is. The device characteristics and status cannot, in any reliable way, be used to extract information about the nature of the service that will be received on the device. For example, if the device characteristics include the speed of the CPU, and the speed is sufficient to support high quality video compression, this cannot be interpreted to mean that video quality would be good for a video service on that device. Other constraints on the system may reduce the amount of CPU available to that service. If there is a desire to indicate that higher quality video is available on a device, that should be done by including service characteristics that say just that. The speed of the CPU might be useful in helping the watcher differentiate between a device that is a PC and one that is a cell phone, in the case where the watcher wishes to call the user's cell phone.

Similarly, if there is dynamic device status (such as whether the device is on or off), and this state impacts the state of the service, this is represented by adjusting the state of the service. Unless a consumer of a presence document has apriori knowledge indicating otherwise (note that presence agents often do), the state of a device has no bearing on the state of the service.

Just like services, there is no enumeration of device types - PCs, PDAs, cell phones, etc. Rather, the device is defined by its characteristics, from which a watcher can extrapolate whether the device is a PDA, cell phone, or what have you.

It is important to point out that the device is a *model* if the underlying physical systems in which services execute. There is nothing that says that this model cannot be used to talk about systems where services run in virtualized systems, rather than real ones. For example, if a PC is executing a virtual machine, and running services within that virtual machine, it is perfectly acceptable to use this model to talk about that PC as being composed of two separate devices.

4. Motivation for the Model

Presence is defined in [5] as the ability, willingness or desire to communicate across a set of devices. The core of this definition is the conveyance of information about the ability, willingness or desire for communications. Thus, the presence data model needs to be tailored around conveying information that achieves this goal.

The person data element is targeted at conveying willingness and desire for communications. It is used to represent information about the user themselves that affects willingness and desire to

communicate. Whether or not I am in a meeting, whether or not I am on the phone - each of these says something about my willingness to communicate, and thus makes sense for inclusion in a presence document.

The service component of the data model aims to convey information on the ability to communicate. The ability to communicate is defined by the services by which a user is reachable. Thus, including them is essential.

How do devices fit in? For many users, devices represent the ability to communicate, not services. Frequently, users my statements like, "call me on my cell phone" or, "I'm at my desk". These are statements for preference for communications using a specific device, as opposed to a service. Thus, it is our expectation that users will

want to represent devices as part of the presence data.

Furthermore, the concept of device adds the ability to correlate services together. The device models the underlying platform that supports all of the services on the phone. Its state therefore impacts all services. For example, if a presence server can determine that a cell phone is off, this says something about the services that run on that device - they are all not available. Thus, if services include indicators about the devices on which they run, device state can be obtained and thus used to compute the state of the services on the device.

The data model tries hard to separate device, service, and person as different concepts. Part of this differentiation is that many attributes will be applicable to some of these, but not others. For example, geographic location is a meaningful attribute of the person (the user has a location) and of a device (the device has a location), but not of a service (services don't inherently have locations). Based on this, geographic location information should only appear as part of device or person, never service. Furthermore, it is possible and meaningful for location information to be conveyed for both device and person, and for these locations to be different. The fact that the presence system might try to determine the location of the person by extrapolation from the location of one of the devices is irrelevant from a data modeling perspective. Person location and device location are not the same thing.

5. Encoding

Information represented according to the data model described above needs to be mapped into an on-the-wire format for transport and storage. The Presence Information Document Format [3] is used for representation of presence data.

The <presence> element contains the presence information for the presentity. The "entity" attribute of this element contains the presentity URI.

The existing <tuple> element in the PIDF document is used to represent the service. This is consistent with the original intent of RFC 2778 and RFC 3863, and achieves backwards compatibility with implementations developed before the model described here was

complete. The <contact> element in the <tuple> element is used to encode the service URI. Presence attributes representing dynamic status appear as children to the <status> element, and attributes representing static characteristics appear directly as children of <tuple>. It is not critical that a clean separation between dynamic and static information be made. It is only important that each presence attribute be specified to appear in either <status> or <tuple>.

This specification introduces the <person> element, which can appear as a child to <presence>. There can only be one instance of this element per document. It contains any number of child elements that indicate characteristic information, followed by the <status> element, which contains any number of elements containing dynamic status information.

This specification also introduces the <device> element, which can appear as a child to <presence>. There can be zero or more instances of this element per document. Each one has a mandatory "device-id" attribute which contains the URN for the device ID. Like <person>, it contains any number of elements containing characteristic information, followed by the <status> element, which contains any number of elements containing dynamic status information.

A client that receives a PIDF document containing the <device> and <person> elements will ignore them. Furthermore, since the semantics of service as defined here are aligned with the meaning of a tuple as defined in RFC 2778 and RFC 3863, documents incorporating the concepts defined in this model are compliant with older implementations.

It's important to note that the mapping of the presence data model into a PIDF document is merely an exercise in syntax.

5.1 XML Schema

5.1.1 Person

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<xs:schema targetNamespace="urn:ietf:params:xml:ns:pidf:person"
```

```
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="urn:ietf:params:xml:ns:pidf:person"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:complexType name="personCharacteristicAbstractType" abstract="true">
  <xs:annotation>
   <xs:documentation>Abstract type for characteristics
   for person</xs:documentation>
  </xs:annotation>
 </xs:complexType>
 <xs:complexType name="personStatusAbstractType" abstract="true">
  <xs:annotation>
   <xs:documentation>Abstract type for status for person</xs:documentation>
  </xs:annotation>
 </xs:complexType>
 <xs:element name="personCharacteristic"
  type="personCharacteristicAbstractType">
  <xs:annotation>
   <xs:documentation>Person characteristic
   element (abstract)</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="personStatus" type="personStatusAbstractType">
  <xs:annotation>
   <xs:documentation>Person status element (abstract)</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="person">
  <xs:annotation>
   <xs:documentation>Contains information
   about the human user</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="personCharacteristic" minOccurs="0"
    maxOccurs="unbounded"/>
    <xs:element name="status">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="personStatus" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

5.1.2 Device

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:pidf:device"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="urn:ietf:params:xml:ns:pidf:device"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:complexType name="deviceCharacteristicAbstractType" abstract="true">
  <xs:annotation>
   <xs:documentation>Abstract type for characteristics for
   device</xs:documentation>
  </xs:annotation>
 </xs:complexType>
 <xs:complexType name="deviceStatusAbstractType" abstract="true">
  <xs:annotation>
   <xs:documentation>Abstract type for status for device</xs:documentation>
  </xs:annotation>
 </xs:complexType>
 <xs:element name="deviceCharacteristic"
 type="deviceCharacteristicAbstractType">
  <xs:annotation>
   <xs:documentation>Device characteristic element
   (abstract)</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="deviceStatus" type="deviceStatusAbstractType">
  <xs:annotation>
   <xs:documentation>Device status element (abstract)</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="device">
  <xs:annotation>
   <xs:documentation>Contains information about the device</xs:documentation
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="deviceCharacteristic" minOccurs="0"
    maxOccurs="unbounded"/>
    <xs:element name="status">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="deviceStatus" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
```

```
      </xs:element>
     </xs:sequence>
```



```
      <xs:attribute name="device-id" type="xs:anyURI" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:schema>
```

6. Publication

Publication is defined as the process by which an event publication agent (EPA) pushes event state into the network [12]. In this section, we consider how an EPA for the presence event package would generate the presence document it will publish.

6.1 Reporting

Reporting is the process whereby a service publishes about itself, an agent on a device publishes about the device, or an agent representing the human user publishes person information elements. Reporting is in contrast to overriding, where a software agent attempts to publish information about a different service or device.

An EPA for presence (also known as a Presence User Agent (PUA)) computes the presence document as if it had full knowledge of the state of the presentity. That is, it represents the complete view of user presence as understood by that PUA. Frequently, the PUA is a software agent that acts as a service, and will therefore be authoritative for the service information it reports. It is anticipated that services will also frequently report information on device and person status as well, as this information is sometimes collected by applications representing services. It is possible that devices can themselves publish information about a presentity, and that software agents representing the person, and not their services, can also publish presence documents. For the remainder of this discussion, we assume that the entity doing the publishing is a service.

When a document is created by such a PUA, the presentity URI (encoded in the "entity" attribute) will typically be a SIP URI, and equal to the AOR of the presentity. This will also usually be the same as the

request URI in the PUBLISH request itself, but it need not be so. The URI serve different purposes. As described in [12], the request URI serves as a means to route the request to the appropriate event state compositor, and identity the target of the publication. As such, it is primary a means for targeting the document. The entity about whom presence is reported is always taken from the "entity" of the presence document.

A PUA will also publish the services it knows about, and the device

it's associated with. The service URI needs to be a unique identifier that defines the service as far as the PUA is concerned. That URI should be a GRUU, as discussed above. The device ID for the device is obtained from the local operating system.

6.2 Overriding

Overriding is the process whereby a PUA attempts to publish information in an explicit attempt to have that information take the place of information published by a different PUA for the same presentity.

The motivating use case for this feature is as follows. A user has an office PC and a home PC, both of which run an Instant Messaging (IM) application. While at work, they set the status of their IM application to "in a meeting". This information is reported in publications produced by the PUA on their office PC. When the user arrives at home, they realize that their office PC is still reporting out-of-date information, and they would like to correct it. As such, the user would like their home PC to publish data that overrides the information being published by their office PC.

In this specific example, the office PC will be publishing a document with a person information element indicating that the user is in a meeting and a service information element indicating availability for IM communications. The service URI is equal to the GRUU for that client. The home PC will be publishing a document with a person information element indicating that the user is at home and a service information element indicating availability for IM service. That IM service uses a different service URI than the one at work, since the two are running on separate UA instances. This presents the presence server with conflicting person information elements for the same

presentity.

Overriding is ultimately an attempt by a publisher to force the composition processing in a presence server to resolve a conflict in a particular way. Ideally, this is done by having a software agent directly set the composition policy that will be used, and then publishing information which will be known to "win" the conflict resolution. In the absence of directly controllable conflict resolution policies, Section 7.2.2 provides guidelines on resolving conflicts for service, device and person information. Publishers can attempt to make use of these guidelines to cause an override to occur.

In most cases, the information that needs to be overriden will be person information. In the example above, the stale information is the status "in a meeting", which is a property of the person

information element. Service information is most usually "self reported" - that is, reported by an agent providing that service. That agent will likely be authoritative for the service, and it is unlikely that some other service needs to provide more up to date information. The situation is more complicated for devices. At the time of writing, most devices did not contain separate agents that published information about themselves; the publication happens from the software agent providing the service. This does present the possibility that conflicting or incorrect information could be reported about a device, neccesitating an override. Since a human being is authoritative about the person information elements, it is likely that any software agent that reports it will have incorrect information. It is for this reason that person information elements are expected to be the most common target for overrides.

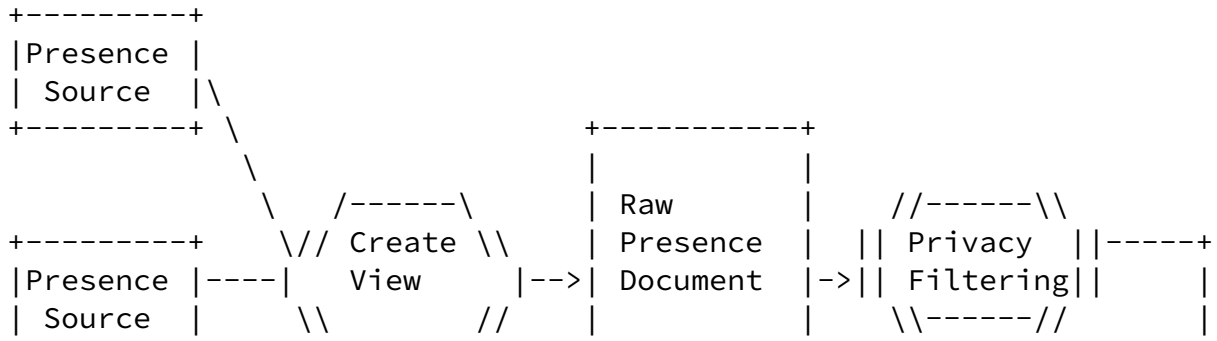
Fortunately, overriding person information is easy. The guidelines in Section 7.2.2 recommend that, absent policy or meta-data guiding otherwise, the most recently reported status wins. An agent wishing to override the person status can therefore just publish a person information element for the presentity. It only needs the presentity URI to do so.

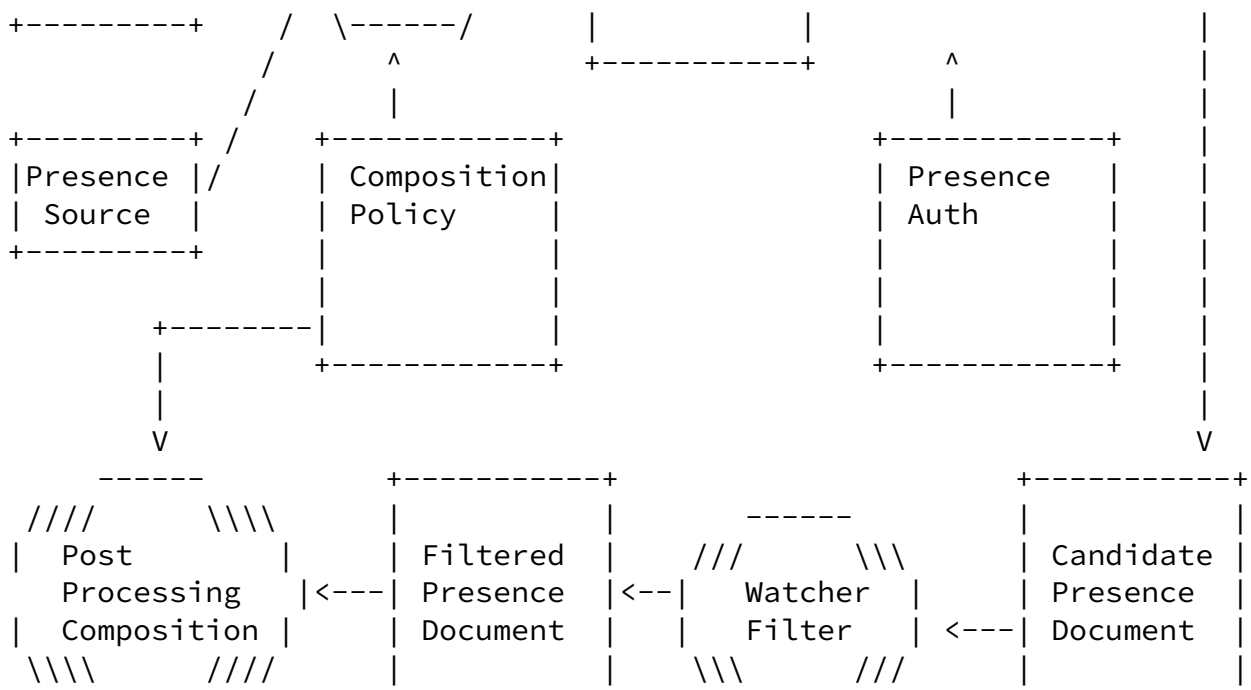
Overriding service and device information elements is more complicated, since it requires the service URI or device ID published by that service or device. The composition operation will often

modify the service URI and device ID before the presence document is distributed to watchers. The result is that a normal watcher of presence information will not have enough information at its disposal to perform an override. At the time of writing, it was anticipated that new event packages could be defined to facilitate this discovery, should the need really arise in practice.

7. Presence Server Processing

In this section, we outline the processing a server does on presence documents. The basic flow of operations is shown in Figure 4.





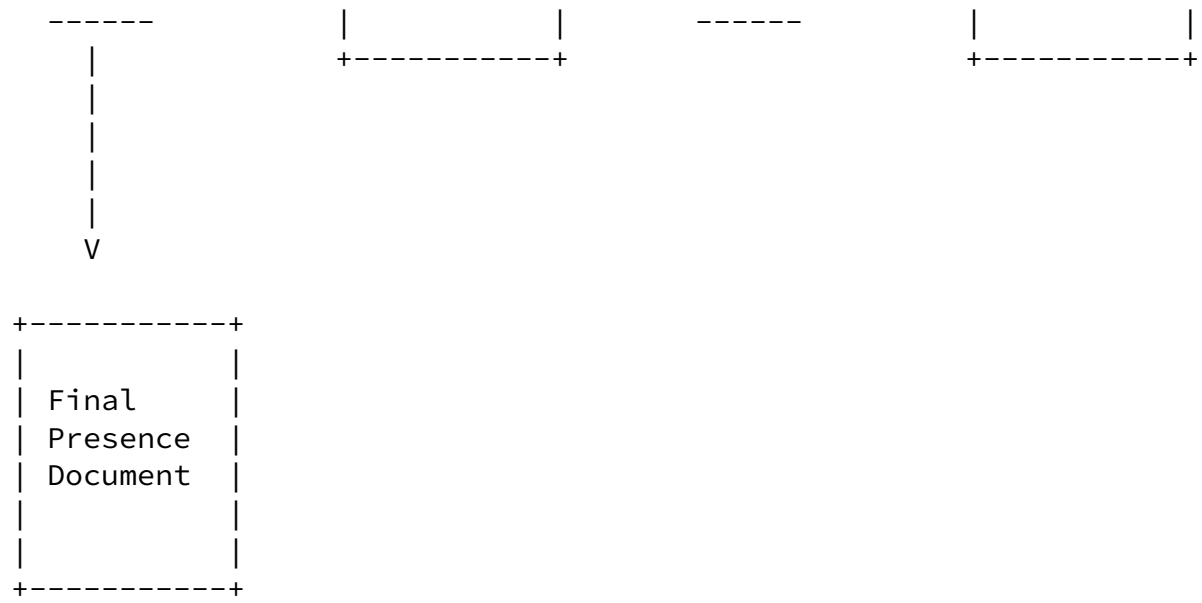


Figure 4

7.1 Collection

The first step is the process of collection. Collection is defined as the process of obtaining the set of event state that is necessary for performing the composition operation that creates the initial view. A view is defined as the particular stream of presence documents seen by a watcher after the application of policy. In this case, the initial view is the view of the presentity before the application of privacy and watcher filtering.

The event state that is collected includes all of the presence documents that have been published for the presentity. This, by definition, is the set of documents whose "entity" attribute in the <presence> element in the presence document is the same as that of the presentity. However, it may also include other presence documents for other presentities, in cases where the presence server knows that the state of one presentity is a function of the state of another. An example is the helpdesk presentity, whose state is a function of the state of the users in the help desk.

In addition to presence events, other event state can be used as

well. As an example, registration state [2] has a bearing on presence, as does dialog state [21], and the state of non-SIP systems, such as traditional telephony equipment, layer 2 devices, and so on. This state can be obtained by a presence server in several ways. Firstly, publishers for that state can send PUBLISH requests for it to the presence server. In another approach, the presence server acts as a watcher, and subscribes to the event state for the resources it needs. This is referred to as a back-end subscription.

Each of these non-presence events can then be converted into a piece of presence state (presentity, device or service information) based on local policy. For example, if the presence server has somehow obtained information that says that the user's cell-phone is on, this can be converted into device state (using the device ID of the phone) along with service state, if the presence server knows about the services on the device.

Registration state is of particular importance. It can be obtained by a presence server by having the presence server co-located with the registrar, or by having the presence server subscribe to the registration event package for the user [2]. Each registered contact is considered a service. The service URI (expressed in the <contact> element in each tuple of the presence document) is obtained from the GRUU for each contact, if it exists, else it is set to the Contact URI from the registration. Service parameters can be extracted from any callee capabilities provided in the registration [13]. The presentity URI is set to the address-of-record. This mapping has the advantage that it is readily correlated to any service information that might also be PUBLISHed explicitly by that UA. As such, a UA that registers should also PUBLISH its state, in the event the presence server cannot access registration information.

Once the non-presence event state is converted into pieces of presence state, the compositor will have, at its disposal, a set of presence data, each of which is for the same presentity.

7.2 Composition

The next step in the process is the composition operation, which produces the raw presence document, also known as the initial view,

from the document sources. This document is "raw" because it contains more information than any watcher might actually see. Privacy and watcher filtering may eliminate some of the data from the document.

The means by which composition is done is a matter of local policy. However, there are some general tools and techniques that merit discussion.

7.2.1 Correlation

A key part of composition is using information in one presence document, describing a person, service or device, to affect information in another. As an example, if the presence server has a document indicating that the user has a telephony service that is busy, the server can use this to extract information about the person - that they are on the phone. Similarly, if one document indicates that a device with ID 1 is off, and another document that indicates a telephony service is running on the device with ID 1, the server can determine that the telephony service is closed.

The way in which the various input data impact each other are a matter of local policy. However, a key to performing such combination operations is the usage of a correlation identifier that can match a service, device, and person together across input sources. The presence document provides the service URI, presentity URI and device ID as correlation identifiers. All three of these identifiers have uniqueness and temporal persistence properties that make them useful for purposes of correlation. Indeed, its not just that the identifiers have temporal persistence; its that they have a common value that is used persistently across different sources. In the example above, the device ID of 1 is useful for correlating the device state to the service state, if, and only if, the source indicating the device state uses the same device ID as the source indicating the service state. This makes selection of the device ID a critically important operation.

7.2.2 Conflict Resolution

In some cases, there may be multiple sources that provide conflicting information about a service, person, or device. In this case, "conflicting" means that there are multiple person data elements that say different things, multiple service data elements for the same service (where the same service is defined as two services with the

same service URI), or multiple device data elements with the same device ID.

Conflicting person information is very likely. The typical situation is described in Section 6.2, where a user wishes to change a stale status set by another software agent no longer under their direct control.

Ultimately, how to resolve conflicts is under the control of the composition policy governing the operation of the server. Here, we discuss approaches that would be typical and appropriate to use.

One way in which conflicts can be resolved is by measuring the likelihood that the information from each source is accurate. In this simple case, the person data element is reported from two IM clients. However, one IM client may report an idle indicator for the device, whilst the other (the home IM client) reports that it is not idle. The presence server can use this information to believe the device which is not idle.

More generally, when a source publishes information, it publishes its "world view", including information it thinks it knows about the person, about the service it is providing, and the device it runs on. The fact that all of these are reported together in a presence document is key. It provides additional context that can be used to determine the level of accuracy of a source for particular information. For example, if a cell phone reports that the user is in a meeting, the cell phone's document will include, in addition to the person status, cell phone device and cell phone service information. Simimlarly, if a calendaring application acts as a source, and indicates that the user is in a meeting, it would include only information about the meeting. The presence server might decide to trust the information that reports *just* the meeting, more than a cell phone that reports a meeting.

The presence server may also know the source of the presence data, based on authenticated identities. For example, in the case above, the calendaring application may have a separate identity it uses to authenticate itself to the presence server. The presence server can be configured to know that the owner of that particular authenticated identity is a calendar application, and therefore, it can trust its information on meeting status information more than another source. [[OPEN ISSUE: do we want a <source> attribute that can be used to explicitly define information about the publisher of the information?? How would this be authorized??]].

Without such additional meta-data, the conflict can be resolved by a

simple freshness metric. The presence source which has most recently

begun reporting information for a specific service, device or person data element, wins. It is imporant not to confuse the time at which a status is initially reported, from when it is refreshed. The former occurs when the status of the person, device or service changes, and the latter occurs for subsequent publications which do not change the value.

Conflicts of services or devices are less likely to occur in the model presented here, due to the unique nature of the service URI and device ID. However, it is possible. Indeed, a client might explicitly choose to publish with the same service URI as another client, if its goal is to explicitly override the service of the other. Using the same service ID is the "hint" to the presence server that conflicting data exists, and one needs to be chosen.

7.2.3 Merging

Merging is an operation that allows a presence server to combine together a set of different services or devices into a single composite service or device. Two services are different if they have different service URIs, and two devices are different if they have different device IDs. This operation is a common one in composition operations.

The merging process involves three steps. The first is to select the set of services or devices to merge. The second is to combine the characteristics and status of each. The third is to generate a composite service URI or device ID.

One way to identify the set of services that will be combined is by defining a "pivot". A pivot is a particular attribute (either characteristic or status) of a service that is used as the selector. All of the services in the raw presence document for whom the pivot attribute has the same value, are all combined together, and the resulting service will clearly have that value for that particular pivot attribute. If the raw presence document has three distinct values for the pivot attribute, the presence document, after combination, will have three services.

For example, if the video prescaps [15] attribute is used as the

pivot, then all services that support video will be combined, and all of those that don't will be combined. The resulting presence document after merging will have two services - one with a characteristic of video, and one with a characteristic of no-video.

An important pivot is the SIP address-of-record. When a PUA publishes a presence document, it includes its GRUU as the service URI in the <contact> element in the tuple. If the presence server

has access to registrar data, it can determine the AOR associated with that GRUU (if there is one). By using the implicitly provided AOR as a pivot, the presence server can combine together all of the services which are reachable through the same AOR.

Once the set of services or devices is selected, the next step is to combine their characteristics and status information. How the characteristics and status are combined will vary for each attribute. For many attributes, if the value is the same across all services, the combination operation is easy - use that value. If the attribute differs across services, it is a matter of local policy as to how they are combined.

As an example, consider the <basic> status as defined in [3]. The most sensible combination operation is the boolean OR operation. That is, a composite service is said to be available as long as one of its component services is available.

The final step, combining service URIs, is more complicated. If the service URIs are GRUUs within the same AOR, they can easily be combined by using the AOR as the result of the combination function. Indeed, even if the presence server is not combining multiple services together, it might make sense to change the GRUU to the AOR in the presence document delivered to a watcher. If the service URIs are SIP URIs but are not GRUUs, the presence server may need to create a URI which represents the collection of services. Requests made to that URI could fork to the set of services that were combined together. If the service URIs are not even the same URI scheme, for example, a mailto and a tel URI, there is little that can be done. In such a case, the <contact> URI should be removed from the document. There are some cases where URIs with distinct URI schemes can be combined. For example, if one service has a tel URI, and the other has a SIP URI, a combined service can be represented by a SIP

URI generated by the presence server. If the watcher generates a request towards this SIP URI, the proxy server could fork the request to the original tel URI and the original SIP URI. This works in this specific case (sip and tel URI combination) because SIP requests can sensibly be directed to a tel URI. These cases aside, it is generally not a good idea to combine services together that have radically different URIs.

The merging operation takes place for devices identically to the way it takes place for services. Fortunately, combining of device IDs is a bit less complicated than combining service URIs. The server can manufacture new device IDs that represent a "virtual" device that represents a collection of other devices.

It is perfectly valid for the merging operation to eliminate all

devices from the final document, or to eliminate the person data element. However, for a presence document to be meaningful, it has to contain at least one service data element (encoded using a <tuple>).

If a presence document is obtained by using the device ID within each service element as a pivot, the result is a device view - there is a single service in the document for each device. If all of the services are composed together, so that the final document has a single service, it is called a presentity view. A service view is used to describe documents where services are either uncombined, or are combined using a pivot other than the device ID.

7.2.4 Splitting

Splitting is the process of taking a single service or device data element, and splitting into two services or devices. This is useful when the presence server or presence user agent wishes to model a complex application (such as a voice, video and IM enabled client) by a multiplicity of distinct services.

The process of splitting involves taking the attributes (both status and characteristics) for the service, and determine which of the component services that attribute will describe. In some cases, a single attribute will be split so that it is present in both components. For example, if the composite service has an idle

indication, meaning that the service has not been used in some time, the component services would both inherit the same value for the idle indicator. In other cases, an attribute gets assigned only to one service, or in other cases, its value is changed as it is split up. The way in which this is done is a matter of local policy.

In all cases, it is important to remember that the purpose of having multiple services or devices described in a document is to give the watcher choice about what service to use. Therefore, the splitting operation should result in multiple services that have sufficient characteristics associated with them to differentiate them to a watcher.

Splitting of a service URI is a relatively simple operation. The entity performing the split creates two new service URIs, each of which, should a request be received for that URI, would get translated to, or routed to, the composite service URI. If a presence user agent is performing the split, it can use the grid parameter of the GRUU to manufacture an infinite supply of URIs that all get routed to itself. If a presence server is doing the split, it can manufacture an entirely new URI (in conjunction with the domain owner, of course) as needed.

When a service is split, there is usually no reason to split the device as well. The component services all run on the same device, and there is much benefit to indicating that this is the case. For example, it would allow a presence server that is compositing the presence document for the presentity, to determine that all of the component services are inactive if the device should fail.

7.3 Privacy Filtering

Once the merging operation has been applied, the next step is to perform privacy filtering. Privacy filtering is a process by which information is removed or transformed in a raw presence document, for the purposes of withholding sensitive information about the presentity. Typically, the filtering operation runs at the bequest of the presentity, in order to protect their own privacy. However, privacy filtering can be instantiated by the operator, in order to execute domain filtering policies, or even third parties that are authorized to specify filtering.

The exact privacy filtering operation that takes place depends on the identity of the watcher, and can also depend on other variables, such as time of day, the weather in Helsinki, and so on. The set of information that dictates the way in which privacy filtering is executed is called authorization policy.

Authorization policy is expressed using the document format defined in [9].

7.4 Watcher Filtering

Watcher filtering is the process by which information is further removed from the presence document. However, it is the watcher which specifies the information subset that they would like to receive. Watcher filtering is accomplished by including filter documents in subscription requests. These filters are then bound to the subscription, and applied to any presence document generated during the lifetime of that subscription.

Filters are described using the document format defined in [16].

7.5 Post-Processing Composition

After the privacy and watcher filtering operations have been applied, the resulting presence document may contain service or device elements which no longer contain enough information to differentiate one from another. As discussed above, the purpose of having multiple services or devices described in a document is to give the watcher choice about which service to invoke. If the services defined in a

document all appear the same, differing only in the service URI, there is no reason for a user to choose one over another. In such a case, composition rules, and in particular, merging of services, will need to be done. The result is the final presence document that can be delivered to watchers.

8. Extending the Presence Model

When new presence attributes are added, any such extension has to consider the following questions:

1. Is the new attribute applicable to person, service or device data elements? If it is applicable to more than one, what is its

meaning in each context? An extension should strive to have each attribute concisely defined for each area of applicability, so that a source can clearly determine to which type of data element it should be applied.

2. Is this new attribute a dynamic status, or a static characteristic? Characteristics are information that describe information about devices, services or a person that help provide context for a consumer of the document to make a decision about whether communications is desired in one place or another. They are therefore descriptive in nature.

9. Example Presence Documents

In this section, we give examples of different physical systems, present the model of that system using the concepts described here, and then show the resulting presence document.

Some of the examples and content in this section are lifted from [6].

9.1 Basic IM Client

In this scenario, a provider is offering a service very similar to the instant messaging services offered today by the public providers like AOL, Yahoo, and MSN. In this service, each user has a "screen name" that identifies them in the service. A single client, generally a PC application, connects to the service at a time. When the client connects, this fact is made available to other watchers of that user in the system. The user has the ability to set a textual note that describes what they are doing, and this note is seen by the watchers in the system. The user can set one of several status messages - such as busy, in a meeting, etc., which are pre-defined notes that the system understands. If a user does not type anything on their keyboard for some time, their status changes to idle on the screens of the various watchers of the system. The system also indicates the amount of time that the user has been idle.

Whenever a user is connected to the system, they are capable of receiving instant messages. A user can set their status to "invisible", which means that they appear as offline to other users. However, if an IM is sent to them, it will still be delivered.

This system is modeled by representing each client in the system with three data elements - a person element, a service element, and a device element. The person element describes the state of the user, including the note and the pre-defined status messages. These represent information about the human user, so they are included in the person element. The service tuple represents the IM service. No characteristics are included. The service URI published by the client is set to the client's GRUU. The device element is used to model the PC. The device element includes the idle indicator, since the idleness refers to usage of the *device*, not the service. Inclusion of the idle indicator in a service tuple is permitted, but would imply something different - that the user hasnt used the service (i.e., has not used their IM client) in some time. [[OPEN ISSUE: Need to determine what the real meaning of idle is.]]

The document published by the client would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:someone@example.com">
      <tuple id="sg89ae">
        <device-id>mac:8asd7d7d70</device-id>
        <prescaps>
          <methods>
            <method>MESSAGE</method>
            <method>OPTIONS</method>
          </methods>
        </prescaps>
        <status>
          <basic>open</basic>
        </status>
        <contact>sip:gruu-someone-1@example.com</contact>
      </tuple>
      <person>
        <status>
          <activities>
           <activity>on-the-phone</activity>
          </activities>
        </status>
      </person>
      <device device-id="mac:8asd7d7d70">
        <status>
          <idle/>
        </status>
      </device>
    </presence>
```

It is worth commenting further on the value of having a separate device element just to convey the idle indicator. As described above, the idle indication of interest is really an indicator that the device is idle. By making that explicit, the idle indicator can be used by the presence server to affect the state of other services running on the same device. For example, let say there is a voip application running on the same device. This application reports its presence information using the example below. Since it reports that it runs on the same device, the presence server can use the status of the service to further refine the idle indicator of the device. Specifically, if the user is using their voip application, the presence server knows that the device is in use, even if the IM application reports that the device is idle. Typically, idleness is determined by lack of keyboard or mouse input, neither of which might be used during a voip call.

In a more simplistic case, reporting the idle indicator as part of

the device status allows that indicator to be used for other services on the same device. Taking, again, the example of the voip application on the same device, if the voip application does not report any device information, and a watcher is not provided information on the IM service, the presence document sent to the watcher can include the device status. Because of the usage of the device IDs and the device information, the presence server can correlate the device status as reported by the IM application with the voip service, and use them together.

9.2 VoIP Application

In this example, consider a SIP network. The user has a SIP AOR of sip:user@example.com. The user has a single SIP PC client that they run on their office machine. This is a simple SIP softphone, supporting audio only.

The PC client publishes a presence document that has a single tuple, representing the service. It does not include any presentity or device elements, although it does include a device-id as part of its service characteristics.

The document published by the client would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:user@example.com">
      <tuple id="sg89ae">
        <device-id>urn:mac:8asd7d7d70</device-id>
        <prescaps>
          <methods>
            <method>INVITE</method>
            <method>OPTIONS</method>
            <method>BYE</method>
            <method>ACK</method>
            <method>CANCEL</method>
          </methods>
          <audio/>
        </prescaps>
        <status>
```

```
            <basic>open</basic>
          </status>
          <contact>sip:gruu2@example.com</contact>
        </tuple>
      </presence>
```


9.3 Cellphone

In this example, the user has a cellphone. This cellphone has an SMS client and a SIP push-to-talk client running. The phone also has a switch that allows the user to select "silent mode". This information is also used by the phone as an indicator of business. As it turns out, the SMS and PTT applications on the phone are totally separate, and each publishes its own information. Indeed, both happen to publish information about the silent mode switch.

The SMS application models itself with three elements - a service element, representing the actual SMS service, a device element, modeling the phone, and a presentity element, modeling the user. Similarly, the PTT app has three elements, representing the PTT service, device, and presentity.

If the user is in a PTT call, the PTT application might generate a document that looks like this. Note the inclusion of the busy element as part of the presentity state, which was set based on the silent switch:

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:someone@example.com">
      <tuple id="sg89ae">
        <device-id>urn:esn:600b40c7</device-id>
        <prescaps>
          <methods>
            <method>INVITE</method>
            <method>OPTIONS</method>
            <method>BYE</method>
            <method>ACK</method>
            <method>CANCEL</method>
          </methods>
          <audio/>
          <duplex>half</duplex>
        </prescaps>
        <status>
          <basic>closed</basic>
        </status>
        <contact>sip:gruu-aa@example.com</contact>
      </tuple>
      <person>
        <status>
          <activities>
           <activity>on-the-phone</activity>
           <activity>busy</activity>
          </activities>
        </status>
```

```
      </person>
      <device device-id="urn:esn:600b40c7">
        <prescaps>
          <mobility>mobile</mobility>
        </prescaps>
      </device>
    </presence>
```

The SMS application would publish a document that looks like this:


```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:someone@example.com">
      <tuple id="sg89ae">
        <device-id>urn:esn:600b40c7</device-id>
        <status>
          <basic>open</basic>
        </status>
        <contact>sms:1234567</contact>
      </tuple>
      <person>
        <status>
          <activities>
           <activity>busy</activity>
           <activity>on-the-phone</activity>
          </activities>
        </status>
      </person>
      <device device-id="urn:esn:600b40c7"/>
    </presence>
```

The presence server now has two presence documents for a single user. It has conflicting information for the person and device elements. It knows it has two services, both of which run on the same device. It merges the two devices into one, and unions the information it has for them. It keeps the services as separate, but changes the PTT URI from the GRUU to the AOR. It merges the person information together, unioning that as well. The resulting raw presence document, after composition, would look like:

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:someone@example.com">
      <tuple id="sg89ae">
        <device-id>urn:esn:600b40c7</device-id>
        <status>
          <basic>open</basic>
        </status>
        <contact>sms:1234567</contact>
      </tuple>
      <tuple id="sg89ae">
        <device-id>urn:esn:600b40c7</device-id>
        <prescaps>
          <methods>
            <method>INVITE</method>
```



```
            <method>OPTIONS</method>
            <method>BYE</method>
            <method>ACK</method>
            <method>CANCEL</method>
          </methods>
          <audio/>
          <duplex>half</duplex>
        </prescaps>
        <status>
          <basic>closed</basic>
        </status>
        <contact>sip:someone@example.com</contact>
      </tuple>
      <person>
```

```
        <status>
          <activities>
           <activity>busy</activity>
          </activities>
        </status>
      </person>
      <device device-id="urn:esn:600b40c7">
        <prescaps>
          <mobility>mobile</mobility>
        </prescaps>
      </device>
    </presence>
```

10. Security Considerations

The presence information described by the model defined here is very sensitive. It is for this reason that privacy filtering plays a key role in the processing of presence data, as described above. Presence systems based on this model need to provide such a privacy capability, and furthermore, need to protect the integrity and confidentiality of the data.

11. Acknowledgements

This document is really a distillation of many ideas discussed over a long period of time. These ideas were contributed by many different participants in the SIMPLE working group. Henning Schulzrinne initially described the "pivot" operation described above for composition. Brian Rosen deserves credit for the "presentity view". Aki Niemi, Paul Kyzivat, Cullen Jennings, Ben Campbell, Robert Sparks, Dean Willis, Adam Roach, Hisham Khartabil, and Jon Peterson contributed many of the concepts that are described here. A special

thanks to Steve Donovan for discussions on the topics discussed here.

12 Informative References

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

[2] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[3] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W. and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.

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

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

[6] Sparks, R., "SIMPLE Presence Document Usage Examples", draft-sparks-simple-pdoc-usage-00 (work in progress), October 2003.

[7] Roach, A., "Identification of Services in RPID (Rich Presence Information Data)", draft-roach-simple-service-features-00 (work in progress), February 2004.

[8] Schulzrinne, H., Gurbani, V., Kyzivat, P. and J. Rosenberg, "RPID: Rich Presence: Extensions to the Presence Information Data Format (PIDF)", draft-ietf-simple-rpid-03 (work in progress), March 2004.

[9] Rosenberg, J., "Presence Authorization Rules", draft-ietf-simple-presence-rules-00 (work in progress), May 2004.

[10] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.

[11] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-02 (work in progress), July 2004.

[12] Niemi, A., "An Event State Publication Extension to the Session Initiation Protocol (SIP)", draft-ietf-sip-publish-04 (work in progress), May 2004.

[13] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[14] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[15] Lonnfors, M. and K. Kiss, "User agent capability presence status extension", draft-ietf-simple-prescaps-ext-01 (work in progress), May 2004.

[16] Khartabil, H., "An Extensible Markup Language (XML) Based Format for Event Notification Filtering", draft-ietf-simple-filter-format-02 (work in progress), August 2004.

[17] 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-02 (work in progress), July 2004.

[18] Schulzrinne, H., "CIPID: Contact Information in Presence Information Data Format", draft-ietf-simple-cipid-03 (work in progress), July 2004.

[19] Wilde, E. and A. Vaha-Sipila, "URI scheme for GSM Short Message Service", draft-wilde-sms-uri-06 (work in progress), July 2004.

[20] Mealling, M., "A UUID URN Namespace", draft-mealling-uuid-urn-03 (work in progress), March 2004.

[21] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-04 (work in progress), February 2004.

[22] Schulzrinne, H., "The tel URI for Telephone Numbers", draft-ietf-iptel-rfc2806bis-09 (work in progress), June 2004.

[23] Isomaki, M., "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Manipulating Presence Document Contents", draft-ietf-simple-xcap-pidf-manipulation-usage-01 (work in progress), June 2004.

Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.