SIMPLE
Internet-Draft
Expires: December 28, 2006

J. Rosenberg
Cisco Systems
June 26, 2006

A Processing Model for Presence
draft-rosenberg-simple-presence-processing-model-02

Status of this Memo

Copyright Notice

Abstract

This document defines the underlying data processing operations used by Session Initiation Protocol (SIP) for Instant Messaging Leveraging Presence Extensions (SIMPLE) presence agents and presence user agents. The data processing operations described here include composition, privacy filtering, and watcher filtering.

Table of Contents

1. Introduction 3
2. Definitions 3
3. Publication 4
3.1. Reporting 4
3.2. Overriding 5
4. Presence Server Processing 7
4.1. SIP Subscription Processing 7
4.2. Presence Document Processing 7
4.2.1. Collection 10
4.2.2. Composition 11
4.2.3. Privacy Filtering 17
4.2.4. Watcher Filtering 17
4.2.5. Post-Processing Composition 18
5. Security Considerations 18
6. Acknowledgements 18
7. Informative References 18
Author's Address 20
Intellectual Property and Copyright Statements 21

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. [6] defines a data model for modeling communications systems using that document format.

This specification is a companion to the data modeling specifications described above. Rather than describing the meaning of the underlying presence data, it describes the processing operations used by presence agents in processing that data. Other specifications, such as the presence event package [4] and the PUBLISH method [8] 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. By providing a model for those operations, consistent interpretation of authorization policies and composition policies across implementations can be achieved. This allows for consistent user experiences.

2. Definitions

Subscription Authorization Decision: The process by which a server determines whether a subscription should be placed into the accepted, rejected or pending states.

Presence Document Generation Process: The flow of operations followed by a presence server that takes a set of presence sources, and based on various policy documents, produces the presence document sent to a particular watcher.

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.

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. Publication

Publication is defined as the process by which an event publication agent (EPA) pushes event state into the network [8]. In this section, we consider how an EPA for the presence event package would generate the presence document it will publish.

3.1. 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 [8], 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.

3.2. 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 4.2.2.2 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 4.2.2.2 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.

4. Presence Server Processing

In this section, we outline the processing done by a presence server. This processing is broken into two components - the SIP subscription processing and the presence document processing.

4.1. SIP Subscription Processing

For the most part, processing of SIP subscriptions is described in RFC 3856 [4]. That specification does leave some hooks for policy-based decisions in subscription handling, as discussed in Section 6.6.2 of RFC 3856. In particular, when the presence server receives a SUBSCRIBE request, it needs to make an immediate decision to put that subscription into one of three states - rejected, successful, and pending. That decision, called the subscription authorization decision, is governed by a Presence Authorization document. This document, discussed below in more detail, also contains information that guides privacy filtering when the subscription is accepted.

Users can change their presence authorization documents at any time. As a result, a user could change those policies to alter the state of the subscription. Changes in the document need to take effect immediately.

4.2. Presence Document Processing

Once a subscription has been accepted, presence documents are delivered to the watcher through notifications. This requires the presence server to generate a presence document for the watcher. The process for doing that is called the presence document generation process. This process is invoked by the presence server under several conditions:

Subscription Transition to Accepted: When the subscription itself transitions to the accepted state, the presence server needs to generate the current state of the presentity and place this in a NOTIFY to the watcher. To do this, the presence document generation process is invoked.

SUBSCRIBE refreshes: Once in the accepted state, SUBSCRIBE refreshes on the SIP dialog request that the server generate a notification containing the current state of the presentity. To do this, the presence document generation process is invoked.

Source changes: When one of the sources of presence information for a user changes, the result may change the state of the presentity. To determine the new state, the server invokes the presence document generation process.

Policy changes: When one of the policy documents governing the presence document generation process changes, the result may change the state of the presentity. To determine the new state, the server invokes the presence document generation process.

The basic steps in the presence document generation process are shown in Figure 1. This is a logical flow, and does not require a server to implement exactly these steps every time the process is invoked.

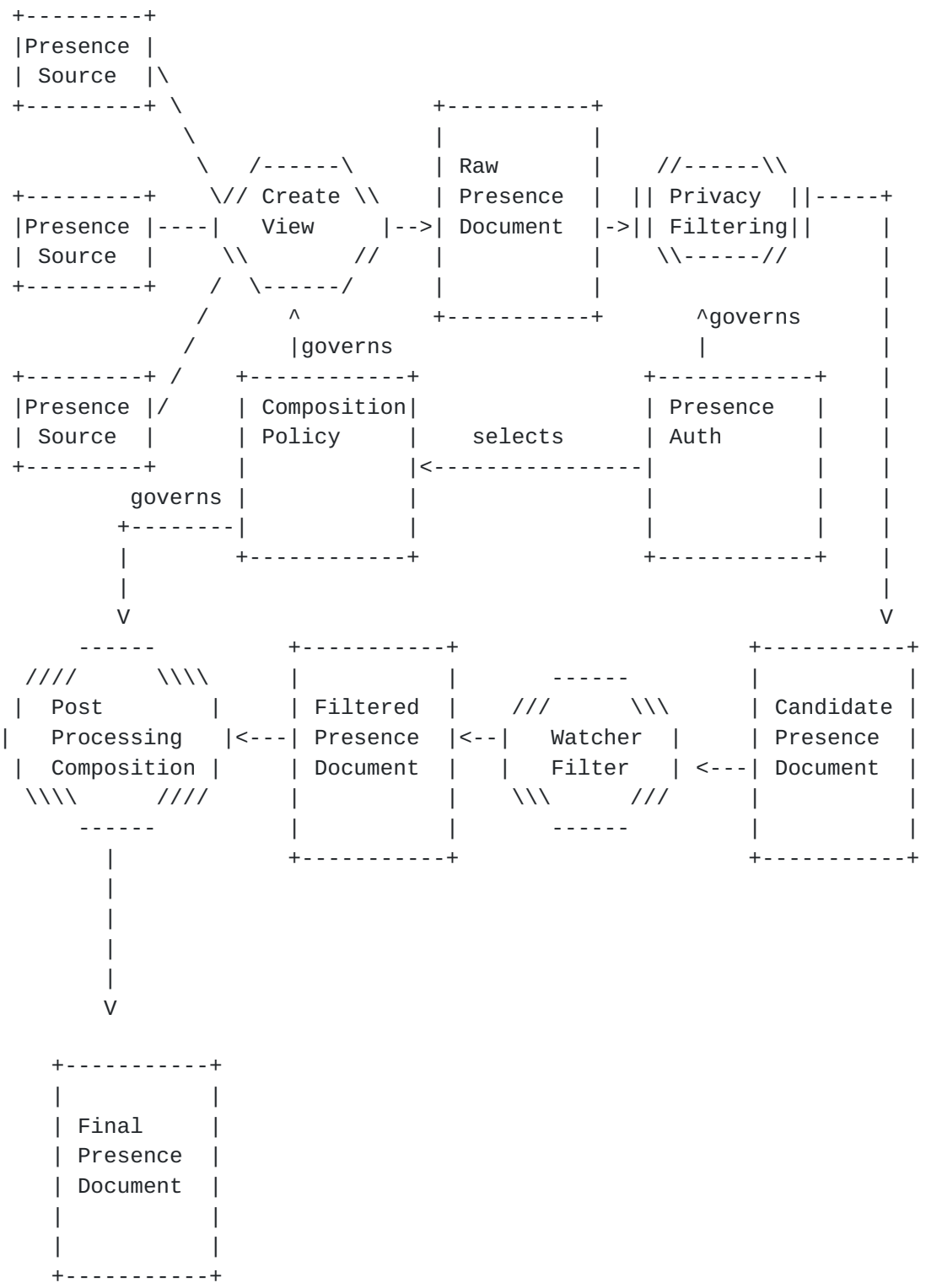
Presence Source
Presence Source
Presence Source
Presence Source
Create View
Raw Presence Document
Privacy Filtering
governs
Composition Policy
selects
Presence Auth
governs
Post Processing Composition
Filtered Presence Document
Watcher Filter
Candidate Presence Document
Final Presence Document

Figure 1

4.2.1. 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 [12], 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 [9]. 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.

4.2.2. 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.

Composition is governed by the rules defined in the composition policy. The composition policy is a document selected by the presence authorization document. Since different composition policies may have differing implications on privacy, the authorization rules themselves need to select the composition policy. As an example, "polite blocking", defined in RFC 2779 [5], is actually the selection of a specific composition policy - one which generates a view that falsely represents the watcher as unavailable. The decision as to whether to invoke this composition policy or not is dictated by the authorization document.

The authorization rules applicable to the presence document generation process can themselves depend on the current state of the presentity, which is derived from the initial view in the raw presence document. This results in a seemingly circular decision - the composition policy to generate the raw presence document is based on authorization policies that are selected by the value of the raw presence document. However, as discussed below, this problem is avoided by using a specific composition policy (the default policy) to compute the view used to assist in the selection of the authorization policy. That authorization policy can select a different composition policy to generate the document actually sent to a watcher.

Composition policies can be complex and rich. However, there are some general tools and techniques that merit discussion.

4.2.2.1. 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.

4.2.2.2. 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 3.2, 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.

4.2.2.3. 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 [10] 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.

4.2.2.4. 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.

4.2.2.5. Default Composition Policy

Unless a user specifies otherwise through an explicit composition policy statement, it is RECOMMENDED that presence servers follow the default composition policy described here. By following this

default, the processing of the presence server becomes more predictable by users and their agents, allowing them to set their presence status in ways that result in the desired predictable output. If a different default is used, users may be surprised by the results of their actions.

TODO: place default policy here

4.2.3. 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 [7].

These rules describe how a series of authorization documents are matched to the subscription, combined together, and then applied. This matching process is based on conditions described in each authorization document. These conditions can include the presence state of the presentity itself. The presence state used to determine these authorization policies is different than the presence state sent to the watcher. To compute this presence state, the presence server runs the presence document generation process using the default composition policy described above, and then stops the process once the raw presence document is generated. This raw presence document is used for any presence states needed to select the authorization policies applicable to the watcher.

4.2.4. 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 [11].

4.2.5. 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.

5. Security Considerations

6. 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.

7. 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., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[5] Day, M., Aggarwal, S., Mohr, G., and J. Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779, February 2000.

[6] Rosenberg, J., "A Data Model for Presence", draft-ietf-simple-presence-data-model-07 (work in progress), January 2006.

[7] Rosenberg, J., "Presence Authorization Rules", draft-ietf-simple-presence-rules-07 (work in progress), June 2006.

[8] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

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

[10] Lonnfors, M. and K. Kiss, "Session Initiation Protocol (SIP) User Agent Capability Extension to Presence Information Data Format (PIDF)", draft-ietf-simple-prescaps-ext-06 (work in progress), January 2006.

[11] Khartabil, H., "An Extensible Markup Language (XML) Based Format for Event Notification Filtering", draft-ietf-simple-filter-format-05 (work in progress), March 2005.

[12] Rosenberg, J., Schulzrinne, H., and R. Mahy, "An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP)", RFC 4235, November 2005.

Author's Address

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.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.