SIPPING
Internet-Draft
Intended status: Best Current Practice
Expires: September 6, 2007

J. Rosenberg
Cisco

March 5, 2007

Identification of Communications Services in the Session Initiation Protocol (SIP)
draft-rosenberg-sipping-service-identification-01

Status of this Memo

Copyright Notice

Abstract

This document considers the problem of how SIP endpoints can support a multiplicity of distinct SIP services within the context of a single user agent. The principle problem to be addressed is that of dispatching of incoming requests to the right services, and how service contexts are matched up between calling and called parties. This document proposes the usage of service URN and service URI to

solve the problem.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [2] defines mechanisms for initiating and managing communications sessions between agents. These agents can be entities such as hardphones, softphones, or gateways to other networks, such as the PSTN. These agents are addressed by SIP URI, and in particular, a SIP Address-of-Record or AOR.

However, in practice, the entities participating in a call can be more complicated. An agent might be inside of a cell phone, supporting traditional telephony, Push-To-Talk, and voice and data content as part of an interactive game. Furthermore, the servers within the network itself might provide additional functions, such as call screening or call recording. These functions are often referred to as 'services', 'features' or 'applications'. Their usage raises questions on how users invoke them, how they are identified, and how interoperability between them is provided.

Section 3 defines the problem in more detail. Section 4 defines concepts and terminology. Section 5 introduces requirements for the solution. Section 6 overviews the solution, and Section 7 defines detailed procedures for user agents, while Section 8 defines procedures for proxies.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant STUN implementations.

3. Problem Statement

Consider a device that allows the user to select two applications. One of these applications is a traditional telephony application, which lets the user make and receive phone calls using telephone numbers. The second application is a two-player chess game that utilizes voice commands to move pieces. When one player says "Queen to D7", the software on their phone recognizes this and moves the piece. At the same time, the user's voice is sent to the other player, where it is both rendered to the user, and interpreted locally in order to move the piece.

If the user should make a call using the telephony application, the

result would be a SIP INVITE for a single voice media stream. Interestingly, if the user launches the game application, the same is true - the result would be a SIP INVITE for a single voice media stream. However, depending on which application the caller selected, the appropriate application at the called party must also be selected. It would be nonsensical for the user to invoke the telephony application and be connected to the gaming application on another user's device. The user interface for the gaming application would not function properly, and the overall experience would be poor. What is needed is some kind of way to differentiate these two applications.

A similar problem arises in the invocation of outbound applications that reside in the network. Consider once more our user with the telephony and gaming application. The user wishes to make a telephony call, but wants the call to be recorded. The recording option is available on a call-by-call basis. The recording application itself resides on a server in the domain of the caller, and acts as a back-to-back user agent (B2BUA) in order to perform the call recording (there are alternative models involving the application interaction framework [9] and conferencing [12], but the specific mechanism is not relevant to the discussion here). When the user initiates the call, how do they signal to their outbound proxy that the call needs to pass through the recording application?

In both cases, the problem at hand is the identification and invocation of applications which reside either on the endpoint (in the first example) or in the network (in the second example).

4. Concepts and Terminology

The problem of identifying and invoking services within SIP is not a new one. The problem has been considered extensively in the context of presence. In particular, the presence data model for SIP [14] defines the concept of a service as one of the core notions that presence describes. Services are described in Section 3.3 of RFC 4479, which has this to say on the topic:

3.3. Service

> Each presentity has access to a number of services. Each of these represents a point of reachability for 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).

It is difficult to give a precise definition for service. One reasonable approach is to model each software or hardware agent in the system as a service. If a user starts a softphone application on their PC, then that represents a service. If a user has a videophone device, then that represents another service. This is effectively a physical view of services. This definition, however, starts to fall apart when a service is spread across multiple software agents or devices. For example, a SIP URI representing an address-of-record can be routed to a softphone or a videophone, or both. In that case, one might attempt instead to define a service based on its address on the network. This definition also falls apart when modeling devices or applications that receive calls and dispatch them to different "helpers" based on potentially complex logic. For example, a cellular telephone might house multiple SIP applications, each of which can "register" different handlers based on the method or even body type of the request. Each of those applications or handlers can rightfully be considered a service, but it doesn't have an address on the network distinct from the others.

Because of this inherent difficulty in precisely defining a service, the data model doesn't try to constrain what can be considered a service. Rather, anything can be considered a service so long as it exhibits a set of key properties defined by this model. In particular, each service is associated with characteristics that identify the nature and capabilities of that service, with reach information that indicates how to connect to the service, with status information representing the state of that service, and relative information that describes the ways in which that service relates to others associated with the presentity.

As a consequence, in this model, services are not explicitly enumerated. There is no central registry where one finds identifiers for each service. Consequently, each service does not have a single "service" attribute with values such as "ptt" or "telephony". That doesn't mean that these consolidated monikers aren't useful; indeed, they represent an essential summary of what the service is. Such summarization is useful in creating icons that allow a user to choose one service over another. A watcher is free to create such summarization information from any of the information associated with a service. The reach information often provides valuable information for creating such a summarization. Oftentimes, the scheme of the URI is synonymous with the view of what a service is. An "sms" URI [14] clearly indicates SMS, for example. For some URIs, there may be many services available, for example, SIP or tel [15], in which case the scheme is less meaningful as a way of creating a summary. The reach information could also indicate that certain application software has to be invoked (such as a videogame), in which case that aspect of the reach information would be useful for generating an iconic

representation of the game.

Building upon this, we can model a user agent as containing a SIP processing layer ontop of which sit a number of different SIP services, as shown in Figure 2

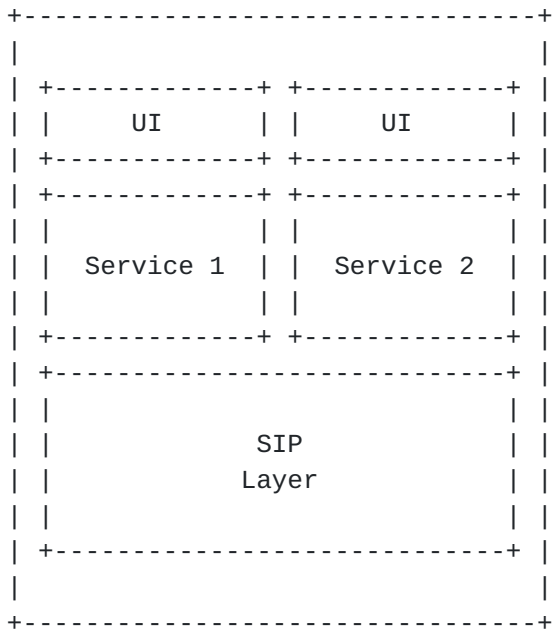

Figure 2

The role of the SIP layer is to parse incoming messages, handle the SIP state machinery for transactions and dialogs, and then dispatch request to the appropriate service. The dispatching operation is based on any number of criteria in the SIP message itself. For example, the method might be used to dispatch the request. A messaging application on the phone would be dispatched when a MESSAGE request arrives. Similarly, when a user interacts with the device, they would select a specific service, and then use that service to initiate communications. The service would then request the SIP layer to send an appropriate message, depending on what was needed. Each service has a user interface (UI) that dictates how it interacts with the user.

SIP uses URI, and in particular, SIP URI, to identify resources within the system. The Address-of-Record, or AOR, identifies the user that is the originator or target of the request. The Globally Routable User Agent URI (GRUU) [4] identifies a specific instance of a user agent. In the model of Figure 2, there is still but one user

agent, and thus a single GRUU. However, we have effectively introduced a layer of hierarchy into the system. Within a particular UA instance, there can be one or more service instances. Each service instance can be addressed by a URI. This URI is formed by adding a parameter, at the discretion of the UA, to the GRUU. The resulting URI is called a service instance URI.

When a service spans multiple devices and multiple SIP UA instances, the aggregate set is represented by a service URI. Typically, a domain will need to construct such a URI, and bind it to the various service instances that can be reached through the service URI.

In addition, each service may or may not be a well-known service. A well-known service is identified by a service URN [5]. The URN refers to the set of assumptions and processing requirements within the service layer that define how a request is processed. For example, a service URN of urn:service:games:voice-chess could be used to identify the voice chess application described in Section 3. In this case, the URN would need to be standardized, and there would be agreement that the "context" is that voice is interpreted by speech recognition for the purposes of performing chess moves, and a specific set of phrases would be agreed upon. It is also possible to have vendor specific services, which would be identified using a URN such as "urn:service:vnd:example.com:foobar", which refers to the foobar service produced by a specific vendor.

It is extremely important to note that this name refers to the additional logic that is required in the processing of a SIP session in order for it to be successfully utilized. SIP assumes that the "normal" service is multimedia communications - the exchange of real time media between the users which generated and received the requests for the purpose of communications between humans or automata which act like a human. The chess example is more than this, because the media is additionally consumed by an automata that is looking to do specialized processing, and because the media is not primarily for human communication, its for controlling moves on a chess board. Because of this, a traditional communications applications has no well-known service associated with it.

5. Requirements

REQ 1: It shall be possible for an incoming request to be dispatched to the correct service on a device.

REQ 2: When multiple services reside on a single device, sharing a single SIP layer, it must not require multiple registrations. This is primarily a performance and overhead requirement.

REQ 3: It shall be possible to support cases where sessions initiated from a particular service purposefully fail unless they can be connected to a matching service for the called party.

REQ 4: It must be possible for services to "match" based on proprietary and well-known identifiers.

REQ 5: It must be possible for a user to initiate a session without knowledge of any information about the recipient except for their AOR.

REQ 6: The mechanism must allow a presence server to determine the services on the phone, without requiring advanced knowledge of those services.

REQ 7: It must be possible to support cases where sessions initiated from a service on the caller side connect to a different service on the other side in cases where the session is meaningful when the notions of service on each side do not match.

6. Overview of Operation

The proposed solution to the problem is relatively straightforward.

The essential problem is that there are cases where a session cannot take place correctly unless the terminating party implements a certain piece of service logic. This is directly analagous to the case where a session cannot take place correctly unless the terminating party implements a certain SIP extension correctly; the problem is just occurring at a different layer in the stack based on the model of Figure 2. Consequently, the proposed solution is similar. The Require header field is utilized, and the option tag is just the service URN for the well-known service, suitably escaped.

When a request with a Require header arrives at the home proxy of the UAS, the home proxy utilizes implicit preferences, as described in RFC 3841 [3]. This will prefer routing of the request to contacts which have indicated support for those extensions. The UAS itself uses the Require header field to dispatch the request to the correct service instance.

In addition, the user agents make use of UA loose routing [6] and GRUU [4], and add an implementation-specific parameter to their GRUU

for each service instance on a device. This allows future out-of-dialog and mid-dialog requests to be targeted at the right service instance, and provides a simple mechanism for dispatch in the device, based entirely on the URI.

When a UAC wants the request to be processed by an application prior to reaching the terminating proxy, it includes the service URN in Route headers that get appended to the route set for the request. For example, if a UA wants a call to be recorded, it would include a service URN like "urn:service:comm:recording" to the bottom-most Route header. This will cause an originating proxy to resolve the service URN to a URI for an application server, and then proxy the request there.

In order to discover available services on a device, presence can be used. A UA would just SUBSCRIBE to the presence of the AOR, and get back a document that contains a service element for each service available for that user. The presence document includes the service URN for any well-known services (noting again that this is only needed when other information, such as method or media types, are insufficient to define the service). The service URN can then be used for creating summary information about the service. The URI present in the contact for that service is the service instance URI or service URI. The former is published by the UA to the network in a presence document, and the latter may be constructed by the network when composing documents together.

7. UA Behavior

7.1. Registration

When a UA supports numerous services, it SHOULD generate a single registration representing the entire UA instance. The UA MUST utilize GRUU [4] and UA loose routing [6]. If any of the services on the UA are well-known services, the UA SHOULD include their URNs as option tags in the extensions media feature tag in the Contact header field parameter.

> The media feature tags can help the network construct presence documents when the UA doesn't publish them separately (though this is recommended as described below). They are also used for routing of requests.
>
> NOTE: An alternative design would be to have each service instance be a separate registered Contact. This would be more helpful for presence, though not needed if a PUBLISH is used. However it has the drawback of adding more state to the network and exposing

"internal" routing within the UA to outside of the UA. It would mean that the proxy would need to know the dispatch logic, which is more likely to be known by the UA.

7.2. Publication

If the UA supports presence, it SHOULD PUBLISH [7] a presence document for itself. This document SHOULD include a service (represented by a tuple) for each service instance. The contact of each tuple SHOULD be derived from the GRUU, and constructed by adding a UA-defined parameter to the GRUU for each service instance. The parameter MUST be different for each service instance, and SHOULD persist over time. The UA SHOULD include information that identifies what the service is, including supported methods and media types, when those are important for its definition. For services that require well-known logic, the agent SHOULD include the service URN amongst the extensions listed for that service.

The UA can do a better job constructing the presence document than the registrar. This is because the UA knows what mechanisms are used to dispatch requests to each service, and knows what well-known service URN are associated with each service. Having an explicit contact for each service allows a UA to unambiguously be reached based on a service selection made by a watcher. This is important, since choice is a key concept provided by presence [14].

7.3. Session Initiation

A UA can initiate a session either directly, or using presence.

When using presence, the UA would start with the AOR for the target. It subscribes to the presence state for the AOR [8]. The result will be a presence document that includes a tuple for each service. The services will include information that describe them with sufficient information for the user to choose one. This may include well-known service URN associated with each service. When the user selects a service for the target user, the UA will generate an INVITE to the contact listed there. Since this contact is a service instance URI or service URI, the request will be routed towards the target UA and explicitly identify the desired service by the URI alone.

Of course, when the UA selected the service to contact, the request would have been initiated from a matching service on the device. In that case, if the initiating request is associated with a required well-known service, the corresponding escaped service URN MUST appear as an option tag in the Require header field.

When initiating a session directly, the user will select a service on the phone and then request communications by entering or selecting the target AOR. If the service from which the request is being made is associated with a required well-known service, the corresponding escaped service URN MUST appear as an option tag in the Require header field.

> This has an important procedural side effect. Based on the rules of the SIP change processs [15], the Require header field can only contain option tags defined in standards track documents. Otherwise, the resulting protocol cannot be considered SIP. This also means that a UA can only require well-known services when they are IETF defined. Otherwise, the resulting protocol is proprietary. This was purposefully done to help temper usage of the mechanism, which can cause significant interop problems if abused.

RFC 3261 uses the 'token' construct for option tags. The service URN is a valid token with the exception of the colon (:). Consequently, when used as an option tag, a service URN MUST be escape coded by replacing the colon with an exclamation point (!).

When initiating a session from a presence document, there is no need for the UA to insert any Require header fields or otherwise add any content to the request beyond what is implied by the contact URI. This does not prevent a UA from inserting one when the UA does in fact require that a specific well-known service be present.

The Contact header field of a dialog forming request SHOULD be formed by taking the GRUU, and adding a URI parameter (at the discretion of the UA) which identifies the particular service invoking the request. The resulting URI is called the service instance URI.

If a UA wants the network to pass the request through application servers that provide specific processing, the UA MUST include a service URN for that service as the bottom-most Route header. The service URN that are available to the UA are learned through mechanisms outside the scope of this specification, and can include configuration [10] for example. If the UA wants the request to be processed by multiple applications, it MUST include a Route header value for each service URN. The UA SHOULD order them based on desired order of invocation, if known.

7.4. Receipt of a Request

When a UA receives a request, it MAY use any content of the request in order to determine which service on the device is appropriate for handling the request. This includes the method, media types, and

required extensions, including any service URN that might be present in the Require header field. The specific means by which a service "registers" itself with the underlying SIP layer to drive the dispatch logic is a matter of local implementation and outside the scope of this specification.

Of course, if the UAS doesn't understand one of the option tags in the Require header field, it will generate a 420 response and include the list of unsupported option tags, including those which happen to be service URN. This is helpful for diagnosing interoperability problems due to incompatible services.

Once the request is delivered to the service instance for processing, any response SHOULD include the service URI derived from the GRUU in the Contact header field.

8. Proxy Behavior

8.1. Request Targeting

When a home proxy receives a request and uses the location service to route the request, it SHOULD follow the procedures defined in RFC 3841 [3] for preference and capability matching. These SHOULD be done even if the request did not contain an Accept-Contact or Reject-Contact header field. When neither was present, the proxy will construct implicit preferences based on the rules in Section 7.2.2 of RFC 3841.

In addition, a proxy SHOULD construct an explicit preference for extensions when the request contains a Require header field. For each option tag in the Require header field, the proxy adds a term to the conjunction of the following form:

(sip.extension=[option tag])

This would include any option tags that were service URN. The result will be that calls get routed to devices which understand the required service.

8.2. Application Invocation

When a proxy receives a request where the next Route header field value after the proxy itself contains a service URN, the proxy MUST resolve the service URN to a SIP URI that can be used to perform that service. The specific mechanism for resolution is outside of the scope of this specification. It can include standardized resolution services such as DDDS [16] or LoST [11], or can be done through local

configuration.

If there are more than one consecutive Route header field values with service URN, a proxy MAY resolve all of them, and MAY reorder them based on localized knowledge of the required invocation sequence. This is particularly important when the proxy is aware of additional applications that need to be invoked, for which it needs to add additional Route header field values.

9. Guidelines for Using Service URN

This document introduces the concept of using a service URN to identify well-known logic that is required in order to successfully process a request. Care must be taken in the usage of this mechanism, or serious interoperability problems can occur.

For example, consider an extreme example whereby the vendor of a UA defines a service URN for each version of their software, under the assumption that the logic in the UA represents a well-known service. If multiple vendors do this, a request from one vendor's device will fail to interoperate with the devices from any other vendor, even if they would be interoperable otherwise.

Consider a more realistic case where a service provider chooses to utilize a well-known service URN for voice telephony and another one for video telephony. There is nothing unique about the actual service logic used to realize each. However, calls made from the video telephony application include a Require header field, requiring the usage of video telephony on the other side. If the call should reach a device that supports only voice, such as a PSTN gateway, the call will automatically fail. However, had existing SIP negotiation techniques been utilized (in this case, the ability to reject media streams), the call would have succeeded.

It is for this reason that the well-known service URN in the Require header field are restricted in several ways. Firstly, they are meant specifically and exclusively for usage in cases where some service logic must be present and matching on both the originating and terminating sides in order for any type of reasonable communications to exist. Secondly, it is limited to capabilities that cannot be negotiated or indicated by other SIP techniques (such as support for a specific media type). One metric for this is that, absent the option tag in the Require header, a request to initiate the session would be identical to a request to or from a different service that is not actually interoperable.

Furthermore, the SIP change process forbids the usage of vendor

proprietary option tags in the Require header field. This means that IETF standardization is required for the definition of service URN that would be used with the mechanism proposed here.

10. Guidelines on Namespace Structure

The service URN [5] creates a basic namespace in which services can be registered. When a new service is added, care should be taken to make sure it is as general purpose as possible while still preserving interoperability. When variations are possible, but for which interoperability exists, these SHOULD be registered using subservices.

This specification requests IANA to create the "vendor" top-level service for vendor specific services. Each sub service MUST be constructed by taking the domain name of the vendor (example.com for example), and following that by a vendor-defined subservice that identifies their service. For example, if vendor example.org wants to create a service called foo, its service URN would be "urn:service:vendor.example.org.foo". Note that these subservices are not IANA registered, and that vendor-defined service URN are not IANA registered SIP option tags.

11. Security Considerations

This specification makes use of option tags and URI to facilitate routing of a request to the appropriate service instance. An attacker in the network could modify these fields to cause the request to be routed to the wrong service instance, which would worsen user experience and possibly cause an interoperability failure. Such an attack would require a man-in-the-middle to modify SIP requests. An attacker capable of such modifications can launch far more disruptive attacks by manipulating other fields, such as Contact or the SDP. Consequently, such attacks do not seem likely.

12. IANA Considerations

This specification registers a new service URN label per the guidelines in Section 4 of [5]. This represents vendor-proprietary services. Allocation of subservices is done using hierarchical allocation [13] and requires no IANA action.

Here is the information to be added to the table of service URN:

Service: vendor

Specification: RFC XXXX [[NOTE TO RFC-EDITOR: Please replace XXXX with the RFC number of this specification.]]

Brief Description: Vendor proprietary service tree

13. Example

Consider our example from Section 3. A user, joe@example.com, starts their chess application and wishes to play with bob@example.com. Joe's INVITE would look like:

```
INVITE sip:bob@example.com SIP/2.0
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK99a
From: Joe <sip:joe@example.com>;tag=n88ah
To: Bob <sip:bob@example.com>
Call-ID: 1j9FpLxk3uxtma7@host.example.com
CSeq: 1 INVITE
Supported: gruu
Require: urn!service!chess
<allOneLine>
Contact:
<sip:joe@example.com
  ;gr=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>
  ;service=chess
</allOneLine>
Content-Length: --
Content-Type: application/sdp

[SDP Not shown]
```

Note that the request contains a Require header field with the service URN. The Contact header field contains a GRUU, and Joe's UA has added a parameter, "service=chess" to this URI. This parameter is used only by Joe's UA for dispatching the request to the chess application when a request is sent to that URI.

In another example, a Joe receives a presence document indicating that the chess service is supported for Bob:

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
 xmlns:dm="urn:ietf:params:xml:ns:pidf:data-model"
 xmlns:rp="urn:ietf:params:xml:ns:pidf:rpid"
 xmlns:caps="urn:ietf:params:xml:ns:pidf:caps"
 xmlns:su="urn:ietf:params:xml:ns:pidf:urn-caps"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 entity="sip:bob@example.com">
 <tuple id="sg89ae">
  <status>
   <basic>open</basic>
  </status>
  <dm:deviceID>mac:8asd7d7d70</dm:deviceID>
  <caps:servcaps>
   <caps:extensions>
    <caps:supported>
     <su:urn>urn:service:chess</su:urn>
    </caps:supported>
   </caps:extensions>
  </caps:servcaps>
  <contact>sip:bob@example.com
  ;gr=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf7>
  ;service=chess</contact>
 </tuple>
 <dm:person id="p1">
  <rp:activities>
   <rp:on-the-phone/>
  </rp:activities>
 </dm:person>
 <dm:device id="pc122">
  <rp:user-input>idle</rp:user-input>
  <dm:deviceID>mac:8asd7d7d70</dm:deviceID>
 </dm:device>
</presence>
```

Joe's UA notices that the chess service is available by the service URN, and it renders an icon representing that service. When Joe selects it, the chess application launches and generates an INVITE. Note that the chess application itself will include a Require header field, since chess has to be supported on the far end to proceed with the call:

```
<alloneline>
INVITE sip:bob@example.com
  ;gr=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf7>
  ;service=chess SIP/2.0
</alloneline>
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK99a
From: Joe <sip:joe@example.com>;tag=n88ah
To: Bob <sip:bob@example.com>
Call-ID: 1j9FpLxk3uxtma7@host.example.com
CSeq: 1 INVITE
Supported: gruu
Require: urn!service!chess
<allOneLine>
Contact:
<sip:joe@example.com
  ;gr=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>
  ;service=chess
</allOneLine>
Content-Length: --
Content-Type: application/sdp

[SDP Not shown]
```

14. Acknowledgements

This document is based on discussions with Paul Kyzivat and Andrew Allen, who contributed significantly to the ideas here.

15. References

15.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] 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.

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

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

October 2006.

[5] Schulzrinne, H., "A Uniform Resource Name (URN) for Services", draft-ietf-ecrit-service-urn-05 (work in progress), August 2006.

[6] Rosenberg, J., "Applying Loose Routing to Session Initiation Protocol (SIP) User Agents (UA)", draft-rosenberg-sip-ua-loose-route-00 (work in progress), October 2006.

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

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

15.2. Informational References

[9] Rosenberg, J., "A Framework for Application Interaction in the Session Initiation Protocol (SIP)", draft-ietf-sipping-app-interaction-framework-05 (work in progress), July 2005.

[10] Petrie, D. and S. Channabasappa, "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-10 (work in progress), January 2007.

[11] Hardie, T., "LoST: A Location-to-Service Translation Protocol", draft-ietf-ecrit-lost-04 (work in progress), February 2007.

[12] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

[13] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[14] Rosenberg, J., "A Data Model for Presence", RFC 4479, July 2006.

[15] Mankin, A., Bradner, S., Mahy, R., Willis, D., Ott, J., and B. Rosen, "Change Process for the Session Initiation Protocol (SIP)", BCP 67, RFC 3427, December 2002.

[16] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS", RFC 3401, October 2002.

Author's Address

Jonathan Rosenberg
Cisco
Edison, NJ
US

Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).