Network Working Group
Internet-Draft
Expires: August 30, 2002

Kutscher
Ott
Bormann
TZI, Universitaet Bremen
March 01, 2002

Session Description and Capability Negotiation
draft-ietf-mmusic-sdpng-04.txt

Status of this Memo

Copyright Notice

Abstract

This document defines a language for describing multimedia sessions with respect to configuration parameters and capabilities of end-systems.

This document is a product of the Multiparty Multimedia Session Control (MMUSIC) working group of the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at mmusic@ietf.org and/or the authors.

Document Revision

$Revision: 4.23 $

Table of Contents

1. Introduction

Multiparty multimedia conferencing is one of the applications that require dynamic interchange of end-system capabilities and the negotiation of a parameter set that is appropriate for all sending and receiving end-systems in a conference. For some applications, e.g. for loosely coupled conferences or for broadcast scenarios, it may be sufficient to simply have session parameters be fixed by the initiator of a conference. In such a scenario no negotiation is required because only those participants with media tools that support the predefined settings can join a media session and/or a conference.

This approach is applicable for conferences that are announced some time ahead of the actual start date of the conference. Potential participants can check the availability of media tools in advance and tools such as session directories can configure media tools upon startup. This procedure however fails to work for conferences initiated spontaneously including Internet phone calls or ad-hoc multiparty conferences. Fixed settings for parameters such as media types, their encoding etc. can easily inhibit the initiation of conferences, for example in situations where a caller insists on a fixed audio encoding that is not available at the callee's end-system.

To allow for spontaneous conferences, the process of defining a conference's parameter set must therefore be performed either at conference start (for closed conferences) or maybe (potentially) even repeatedly every time a new participant joins an active conference. The latter approach may not be appropriate for every type of conference without applying certain policies: For conferences with TV-broadcast or lecture characteristics (one main active source) it is usually not desired to re-negotiate parameters every time a new participant with an exotic configuration joins because it may inconvenience existing participants or even exclude the main source from media sessions. But conferences with equal "rights" for participants that are open for new participants on the other hand would need a different model of dynamic capability negotiation, for example a telephone call that is extended to a 3-parties conference at some time during the session.

SDP [2] allows to specify multimedia sessions (i.e. conferences, "session" as used here is not to be confused with "RTP session"!) by providing general information about the session as a whole and specifications for all the media streams (RTP sessions and others) to be used to exchange information within the multimedia session.

Currently, media descriptions in SDP are used for two purposes:

- to describe session parameters for announcements and invitations (the original purpose of SDP) and

- to describe the capabilities of a system and possibly provide a choice between a number of alternatives (which SDP was not designed for).

A distinction between these two "sets of semantics" is only made implicitly.

This document is based upon a set of requirements specified in a companion document [1]. In the following, we first introduce a model for session description and capability negotiation as well as the basic terms used throughout this specification (section 2). Next, we outline the concept for the concepts underlying SDPng and introduce the syntactical components step by step in section 3. In section 4, we provide a formal definition of the SDPng session description language. Finally, we overview aspects of using SDPng with various IETF signaling protocols in section 5. In Appendix A, we provide basic audio codec and payload type definitions that are subsumed in SDPng libraries in Appendix B and Appendix C.

The next version of this draft will only contain the formal specification of the language itself. Requirements and the description of the system model will be moved to a separate document.

2. Terminology and System Model

Any (computer) system has, at a time, a number of rather fixed hardware as well as software resources. These resources ultimately define the limitations on what can be captured, displayed, rendered, replayed, etc. with this particular device. We term features enabled and restricted by these resources "system capabilities".

> Example: System capabilities may include: a limitation of the screen resolution for true color by the graphics board; available audio hardware or software may offer only certain media encodings (e.g. G.711 and G.723.1 but not GSM); and CPU processing power and quality of implementation may constrain the possible video encoding algorithms.

In multiparty multimedia conferences, participants employ different "components" in conducting the conference.

> Example: In lecture multicast conferences one component might be the voice transmission for the lecturer, another the transmission of video pictures showing the lecturer and the third the transmission of presentation material.

Depending on system capabilities, user preferences and other technical and political constraints, different configurations can be chosen to accomplish the use of these components in a conference.

Each component can be characterized at least by (a) its intended use (i.e. the function it shall provide) and (b) one or more possible ways to realize this function. Each way of realizing a particular function is referred to as a "configuration".

> Example: A conference component's intended use may be to make transparencies of a presentation visible to the audience on the Mbone. This can be achieved either by a video camera capturing the image and transmitting a video stream via some video tool or by loading a copy of the slides into a distributed electronic white-board. For each of these cases, additional parameters may exist, variations of which lead to additional configurations (see below).

Two configurations are considered different regardless of whether they employ entirely different mechanisms and protocols (as in the previous example) or they choose the same and differ only in a single parameter.

> Example: In case of video transmission, a JPEG-based still image protocol may be used, H.261 encoded CIF images could be sent, as

could H.261 encoded QCIF images. All three cases constitute different configurations. Of course there are many more detailed protocol parameters.

Each component's configurations are limited by the participating system's capabilities. In addition, the intended use of a component may constrain the possible configurations further to a subset suitable for the particular component's purpose.

Example: In a system for highly interactive audio communication the component responsible for audio may decide not to use the available G.723.1 audio codec to avoid the additional latency but only use G.711. This would be reflected in this component only showing configurations based upon G.711. Still, multiple configurations are possible, e.g. depending on the use of A-law or u-Law, packetization and redundancy parameters, etc.

In modelling multimedia sessions, we distinguish two types of configurations:

- potential configurations
  (a set of any number of configurations per component) indicating a system's functional capabilities as constrained by the intended use of the various components;

- actual configurations
  (exactly one per instance of a component) reflecting the mode of operation of this component's particular instantiation.

  Example: The potential configuration of the aforementioned video component may indicate support for JPEG, H.261/CIF, and H.261/QCIF. A particular instantiation for a video conference may use the actual configuration of H.261/CIF for exchanging video streams.

In summary, the key terms of this model are:

- A multimedia session (streaming or conference) consists of one or more conference components for multimedia "interaction".

- A component describes a particular type of interaction (e.g. audio conversation, slide presentation) that can be realized by means of different applications (possibly using different protocols).

- A configuration is a set of parameters that are required to implement a certain variation (realization) of a certain component. There are actual and potential configurations.

* Potential configurations describe possible configurations that are supported by an end-system.

* An actual configuration is an "instantiation" of one of the potential configurations, i.e. a decision how to realize a certain component.

In less abstract words, potential configurations describe what a system can do ("capabilities") and actual configurations describe how a system is configured to operate at a certain point in time (media stream spec).

To decide on a certain actual configuration, a negotiation process needs to take place between the involved peers:

1. to determine which potential configuration(s) they have in common, and

2. to select one of this shared set of common potential configurations to be used for information exchange (e.g. based upon preferences, external constraints, etc.).

In SAP-based [9] session announcements on the Mbone, for which SDP was originally developed, the negotiation procedure is non-existent. Instead, the announcement contains the media stream description sent out (i.e. the actual configurations) which implicitly describe what a receiver must understand to participate.

In point-to-point scenarios, the negotiation procedure is typically carried out implicitly: each party informs the other about what it can receive and the respective sender chooses from this set a configuration that it can transmit.

Capability negotiation must not only work for 2-party conferences but is also required for multi-party conferences. Especially for the latter case it is required that the process to determine the subset of allowable potential configurations is deterministic to reduce the number of required round trips before a session can be established. For instance, in order to be used with SIP, the capability negotiation is required to work with the offer/answer model that is for session initiation with SIP -- limiting the negotiation to exactly one round trip.

The requirements for the SDPng specification, subdivided into general requirements and requirements for session descriptions, potential and actual configurations as well as negotiation rules, are captured in a companion document [1].

3. SDPng

This section introduces the underlying concepts of the Session Description Protocol - next generation (SDPng). The focus of this section is on the concepts of the capability description and negotiation language with a stepwise introduction of the various syntactical elements. Note that this section does only examples accompanied by explanations -- a full formal specification is provided in section 4.

3.1 Conceptual Outline

The description language follows the system model introduced in the beginning of this document. We use a rather abstract language to avoid misinterpretations due to different intuitive understanding of terms as far as possible.

The concept of a capability description language addresses various pieces of a full description of system and application capabilities in four separate "sections":

Definitions (elementary and compound); see Section 3.1.1.

Potential or Actual Configurations; see Section 3.1.2.

Constraints; see Section 3.1.3.

Session attributes; see Section 3.1.4.

3.1.1 Definitions

The "Definitions" section specifies a number of basic abstractions that are later referenced to avoid repetitions in more complex specifications and allow for a concise representation. Definition elements are labelled with an identifier by which they may be referenced. They may be elementary or compound (i.e. combinations of elementary entities). Examples of definitions that could occur in "Definitions" sections include (but are not limited to) codec definitions, redundancy schemes, transport mechanisms and payload formats.

Elementary definition elements do not reference other elements. Each elementary entity only consists of one of more attributes and their values. Default values specified in the definition section may be overridden in descriptions for potential (and later actual) configurations. A mechanisms for overriding definitions is specified below.

For the moment, elementary abstractions are defined for media types (i.e. codecs) and for media transports mechanisms. For each transport and for each codec to be used, the respective attributes need to be defined. This definition may either be provided within the "Definitions" section itself or in an external document (similar to the audio-video profile or an IANA registry that defines payload types and media stream identifiers).

It is not required to define all codecs and transport mechanisms in a "Definitions" sections and reference them when specifying potential and actual configurations. Instead, a syntactic mechanism is defined that allows to give some definitions directly in a configurations section.

Examples for elementary definitions:

```
<audio:codec name="audio-basic" encoding="PCMU"
             sampling="8000" channels="1"/>


<audio:codec name="audio-L16-mono" encoding="L16"
             sampling="44100" channels="1"/>
```

The element type "audio:codec" is used in these examples to define audio codec configurations. The configuration parameters are given as attribute values.

Definitions may have default values specified along with them for each attribute (as well as for their contents). Some of these default values may be overridden so that a codec definition can easily be re-used in a different context (e.g. by specifying a different sampling rate) without the need for a large number of base specifications. In the following example the definition of audio-L16-mono is re-used for the defintion of the corresponding stereo codec. Appendix A provides a complete set of corresponding audio:codec definitions of the codecs used in RFC 1890 [4].

```
<audio:codec name="audio-L16-stereo" ref="audio-L16-mono"
             channels="2"/>
```

The example shows how existing definitions can be referenced in new definitions. This approach allows to create simple as well as more complex definitions in an extensible set of reference documents. Section 3.4 specifies the mechanisms for external references.

Besides definitions of audio codecs other definitions such as RTP payload formats and specific transport mechanisms are suitable to be defined in a definition section for later referencing. The following

example shows how RTP payload types are defined using a pre-defined codec.

```
<rtp:pt name="rtp-avp-0" pt="0" format="audio-basic"/>
<rtp:pt name="rtp-avp-11" pt="11" format="audio-L16-mono"/>
```

In this example, the payload type "rtp-avp-11" is defined with payload type number 11, referencing the codec "audio-L16-mono". Instead of referencing an existing definition it is also possible to define the format "inline":

```
<rtp:pt name="rtp-avp-10" pt="10">
 <audio:codec encoding="L16" sampling="44100" channels="2"/>
</rtp:pt>
```

Note: For negotiation between endpoints, it may be helpful to define two modes of operation: explicit and implicit. Implicit specifications may refer to externally defined entities to minimize traffic volume, explicit specifications would list all external definitions used in a description in the "Definitions" section. Again, see Section 3.4 for complete discussion of external definitions.

The "Definitions" section may be empty if all transport, codecs, and other pieces needed to the specify Potential and Actual Configurations (as detailed below) are either included by referencing external definitions or are explicitly described within the Configurations themselves.

3.1.2 Components & Configurations

The "Configurations" section contains all the components that constitute the multimedia application (IP telephone call, real-time streaming application, multi-player gaming session etc.). For each of these components, the potential and, later, the actual configurations are given. Potential configurations are used during capability exchange and/or negotiation, actual configurations to configure media streams after negotiation (e.g. with RTSP) or in session announcements (e.g. via SAP). A potential and the actual configuration of a component may be identical.

Each component is labelled with an identifier so that it can be referenced, e.g. to associate semantics with a particular media stream. For such a component, any number of configurations may be given with each configuration describing an alternative way to realize the functionality of the respective component.

Each configuration (potential as well as actual) is labelled with an

identifier. A configuration combines one or more (elementary and/or compound) entities from the "Definitions" section to describe a potential or an actual configuration. Within the specification of the configuration, default values from the referenced entities may be overwritten. In addition, it is also possible to provide definition elements inline, inside the definition of a configuration.

Note: Not all protocol environments and their respective operation allow to explicitly distinguish between Potential and Actual Configurations. Therefore, SDPng so far does not provide for syntactical identification of a Configurations as being a Potential or an Actual one. The semantics of configurations are to be determined from the requirements of the specific protocol that uses SDPng to express capabilities and configurations.

The following example shows how RTP sessions can be described by referencing payload definitions.

```
<cfg>
  <component name="interactive-audio" media="audio">
    <alt name="AVP-audio-0">
      <rtp:session format="rtp-avp-0">
       <rtp:udp addr="224.2.0.53" rtp-port="7800" rtcp-port="7801"/>
      </rtp:session>
    </alt>

    <alt name= AVP-audio-11">
      <rtp:session format="rtp-avp-11">
       <rtp:udp addr="224.2.0.53" rtp-port="7800" rtcp-port="7801"/>
      </rtp:session>
    </alt>
   </component>
</cfg>
```

For example, an IP telephone call may require just a single component "name=interactive-audio" with two possible ways of implementing it. The two corresponding configurations are "AVP-audio-0" without modification, the other ("AVP-audio-11") uses linear 16-bit encoding. Typically, transport address parameters such as the port number would also be provided. In this example, this information is given by the "rtp:udp" element. Of course, it must be possible to specify other transport mechanisms as well. See Section 3.2 for a discussion of extension mechanisms that allow applications to use non-standard transport (or other) specifications.

During/after the negotiation phase, an actual configuration is chosen out of a number of alternative potential configurations, the actual configuration may refer to the potential configuration just by its

"id", possibly allowing for some parameter modifications.
Alternatively, the full actual configuration may be given.

Instead of referencing existing payload type definitions it is also possible to provide the required information "inline". The following example illustrates this:

```
<cfg>
  <component name="audio1" media="audio">
    <alt name="AVP-audio-0">
      <rtp:session>
       <rtp:pt pt="0">
        <audio:codec name="audio-basic" encoding="PCMU"
                     sampling="8000" channels="1"/>
       </rtp:pt>
       <rtp:udp addr="224.2.0.53" rtp-port="7800" rtcp-port="7801"/>
      </rtp:session>
    </alt>
   </component>
</cfg>
```

The UDP/IPv4 multicast transport that is used in the examples is a simple variant of a transport specification. More complex ones are conceivable. For example, it must also be possible to specify the usage of source filters (inclusion and exclusion), Source Specific Multicast, the usage of multi-unicast, or other parameters such as QoS parameters. Therefore it is possible to extend the definition of transport mechanisms by providing the required information in the element content. An example:

```
<cfg>
  <component name="audio1" media="audio">
    <alt name="AVP-audio-0">
      <rtp:session format="rtp-avp-0">
       <rtp:udp addr="224.2.0.53" rtp-port="7800" rtcp-port="7801">
        <option name="ssm" sender="sender.example.com"/>
       </rtp:udp>
      </rtp:session>
    </alt>
   </component>
</cfg>
```

Additional transport mechanisms and options will be defined in future versions of this document.

3.1.3 Constraints

Definitions specify media, transport, and other capabilities, whereas

configurations indicate which combinations of these could be used to provide the desired functionality in a certain setting.

There may, however, be further constraints within a system (such as CPU cycles, DSP resources available, dedicated hardware, etc.) that limit which of these configurations can be instantiated in parallel (and how many instances of these may exist). We deliberately do not couple this aspect of system resource limitations to the various application semantics as the constraints may exist across application boundaries. Also, in many cases, expressing such constraints is simply not necessary (as many uses of the current SDP show), so additional overhead can be avoided where this is not needed.

Therefore, we introduce a "Constraints" section to contain these additional limitations. Constraints refer to potential configurations and to entity definitions and express and use simple logic to express mutual exclusion, limit the number of instantiations, and allow only certain combinations. The following example shows the definition of a constraints that restricts the maximum number of instantiation of two alternatives (that would have to be defined in the configuration section before) when they are used in parallel:

```
<constraints>
  <par>
    <use-alt ref="AVP-audio-11" max="5">
    <use-alt ref="AVP-video-32" max="1">
  </par>
</constraints>
```

As the example shows, constraints are defined by defining limits on simultaneous instantiations of alternatives. They are not defined by expressing abstract end-system resources, such as CPU speed or memory size.

By default, the "Constraints" section is empty (or missing) which means that no further restrictions apply.

3.1.4 Session Attributes

The fourth and final section of the SDPng syntax addresses session layer attributes. These attributes largely include those defined by SDP [RFC2327] (which are explicitly indicated in the following specification) to describe originator, purpose, and timing of a multimedia session among other characteristics. Furthermore, SDPng includes attributes indicating the semantics of the various Components in a teleconference or other session. This part of the specification is open ended with an IANA registry to be set up to

register further types of components; only a few of the examples are listed here.

A session-level specification for connection information (SDP "c=" line), bandwidth information (SDP "b=" line), and encryption keys (SDP "k=" lines) is deliberately not provided for in SDPng. The relevant information can be specified directly in the Configuration section for individual alternatives.

Session level attributes as defined by SDP still have to be examined and adopted for SDPng in a future revision of this specification.

3.1.4.1 Owner

The owner refers to the creator of a session as defined in RFC2327 ("o=" line). The syntax is as follows:

```
<owner user="username" session-id="session-id" version="version"
       nettype="IN" addrtype="IP4" addr="130.149.25.97"/>
```

The owner element must be present if SDPng is used with SAP. For all other protocols, the owner element is not necessarily required. The attributes listed above match those from the SDP specification; all attributes must be present and they are used following the rules of RFC2327.

The owner element is an empty element.

3.1.4.2 Session Identification

The "session" element is used to identify the session and to provide a description and possible further references. It provides the following attributes:

name: The session name as it is to appear e.g. in a session directory. This is equivalent to the SDP "s=" line.

The session element can contain arbitrary text of any length (but authors are encouraged to keep the inline description brief and provide additional information via URLs using the info element described below. This text is used to provide a description of the session; it is the equivalent of the SDP "i=" lines.

Additionally, the session element can contain other elements of the following types to provide further information about the session and its creator:

info: The info element is intended to provide a pointer to further

information on the session itself. It is an empty element and provides the attribute xlink:href that is used to specify an URI. Info elements are optional, they may occur any number of times.

contact: The contact element provides contact information on the creator of the session. It is an empty element and provides an attribute xlink:href that is used to specify an URI. Any URI scheme suitable to reach a person or a group of persons is acceptable (e.g. sip:, mailto:, tel:). Contact elements are optional, they may occur any number of times.

```
<session name="An SDPng seminar">
    And here comes a long description of the seminar indicating what
    this might be about and so forth. But we also include further
    information -- as additional elements:
    <info xlink:href="http://www.ietf.org/"/>
    <contact xlink:href="mailto:joe@example.com"/>
    <contact xlink:href="mailto:bob@example.com"/>
    <contact xlink:href="tel:+49421218"/>
    <contact xlink:href="sip:joe@example.com"/>
    <contact xlink:href="sip:bob@example.com"/>
</session>
```

3.1.4.3 Time Specification (SDP 't=', 'r=', and 'z=' lines)

The time specification for a session follows the same rules as in SDP. Time specifications are usually only meaningful when used in conjunction with SAP and are optional. SDPng uses the following elements and attributes to specify timing:

The element "time" is used to indicate a schedule for the session; time has two optional attributes:

start: The starting time of the first occurrence of the session as defined in RFC2327.

end: The ending time of the last occurrence of the session as defined in RFC2327.

The time element can contain the following elements:

repeat: This element specifies the repetition pattern for the schedule. There may be zero or more occurrences of this element within the time element. "repeat" has two mandatory and one optional attribute and is an empty element; the attributes are as defined in SDP:

interval: The duration between two start times of the session. This attribute is always present.

duration: The duration for which the session will be active starting at each repetition interval. This attribute is always present.

offset: The offset relative to "start" attribute at which this repetition of the session is start. This attribute is optional; if it is absent, a default value of "0" is assumed.

Formatting of the attribute values follows the rules defined in RFC2327.

zone: The zone element specifies one or more time zone adjustments as defined in RFC2327. This element has zero or more occurrences in the time element. It is an empty element and has two attributes as defined in SDP:

adjtime: The time at which the next adjustment will take place.

delta: The adjustment offset (typically +/- 1 hours).

The example from RFC2327, page 16, expressed in SDPng:

```
<time start="3034423619" stop="3042462419">
  <repeat interval="7d" duration="1h"/>
  <repeat interval="7d" duration="1h" offset="25h"/>
</time>
```

The time element can occur multiple times.

3.1.4.4 Component Semantic Specification

Another important session parameter is to specify - ideally in a machine-readable way but at least understandable for humans - the function of the various components in a session. Typically, the semantics of the streams are implicitly assumed (e.g. a video stream goes together with the only audio stream in a session). There are, however, scenarios in which such intuitive understanding is not sufficient and the semantics must be made explicit.

```
<info name="audio-interactive" function="speaker">
    Audio stream for the different speakers
</info>
```

The above example shows a simple definition of the semantics for the

component "audio-interactive". Further options may be added to provide additional information, e.g. language, and other functions may be specified (e.g. "panel", "audience", "chair", etc.).

3.2 Syntax Definition Mechanisms

In order to allow for the possibility to validate session descriptions and in order to allow for structured extensibility, SDPng relies on a syntax framework that provides concepts as well as concrete procedures for document validation and extending the set of allowed syntax elements.

SGML/XML technologies allow for the creation of Document Type Definitions (DTDs) that can define the allowed content models for the elements of conforming documents. Documents can be formally validated against a given DTD to check their conformance and correctness. XML DTDs however, cannot easily be extended. It is not possible to alter to content models of element types or to add new element types after the DTD has been specified.

For SDPng, a mechanism is needed that allows the specification of a base syntax -- for example basic elements for the high level structure of description documents -- while allowing extensions, for example elements and attributes for new transport mechanisms, new media types etc. to be added on demand. Still, it has to be ensured that extensions do not result in name collisions. Furthermore, it must be possible for applications that process descriptions documents to distinguish extensions from base definitions.

For XML, mechanisms have been defined that allow for structured extensibility of a model of allowed syntax: XML Namespace and XML Schema.

XML Schema mechanisms allows to constrain the allowed document content, e.g. for documents that contain structured data and also provide the possibility that document instances can conform to several XML Schema definitions at the same time, while allowing Schema validators to check the conformance of these documents.

Extensions of the session description language, say for allowing to express the parameters of a new media type, would require the creation of a corresponding XML schema definition that contains the specification of element types that can be used to describe configurations of components for the new media type. Session description documents have to reference the non-standard Schema module, thus enabling parsers and validators to identify the elements of the new extension module and to either ignore them (if they are not supported) or to consider them for processing the

session/capability description.

It is important to note that the functionality of validating capability and session description documents is not necessarily required to generate or process them. For example, endpoints would be configured to understand only those parts of description documents that are conforming to the baseline specification and simply ignore extensions they cannot support. The usage of XML and XML Schema is thus rather motivated by the need to allow for extensions being defined and added to the language in a structured way that does not preclude the possibility to have applications to identify and process the extensions elements they might support. The baseline specification of XML Schema definitions and profiles must be well-defined and targeted to the set of parameters that are relevant for the protocols and algorithms of the Internet Multimedia Conferencing Architecture, i.e. transport over RTP/UDP/IP, the audio video profile of RFC1890 etc.

Section 3.4 describes profile definitions and library definition. A detailed definition of how the formal SDPng syntax and the corresponding extension mechanisms is provided in Section 5.

The example below shows how the definition of codecs, transport-variants and configuration of components as well as a conference description are realized in SDPng.

```
<def>
 <audio:codec name="audio-basic" encoding="PCMU"
              sampling="8000" channels="1"/>

 <audio:codec name="audio-L16-mono" encoding="L16"
              sampling="44100" channels="1"/>

 <rtp:pt name="rtp-avp-0" pt="0" format="audio-basic"/>
 <rtp:pt name="rtp-avp-11" pt="11" format="audio-L16-mono"/>

</def>

<cfg>
  <component name="interactive-audio" media="audio">
    <alt name="AVP-audio-0">
      <rtp:session format="rtp-avp-0">
       <rtp:udp addr="224.2.0.53" rtp-port="7800" rtcp-port="7801"/>
      </rtp:session>
    </alt>

    <alt name="AVP-audio-11">
      <rtp:session format="rtp-avp-11">
```

```
        <rtp:udp addr="224.2.0.53" rtp-port="7800" rtcp-port="7801"/>
       </rtp:session>
     </alt>
    </component>
  </cfg>

  <constraints>
    <par>
      <use-alt ref="AVP-audio-11" max="1">
    </par>
  </constraints>

  <conf>
   <owner user="joe@example.com" id="foobar" version="1" nettype="IN"
                        addrtype="IP4" addr="130.149.25.97"/>
   <session name="An SDPng seminar">
    This seminar is about SDPng...
    <info xlink:href="http://www.ietf.org/"/>
    <contact xlink:href="mailto:joe@example.com"/>
    <contact xlink:href="sip:joe@example.com"/>
   </session>

   <time start="3034423619" stop="3042462419">
    <repeat interval="7d" duration="1h"/>
    <repeat interval="7d" duration="1h" offset="25h"/>
   </time>

   <info name="interactive-audio" function="speaker">
      Audio stream for the different speakers
   <info>

  </conf>
```

Section 5 specifies the formal Schema definition that this example is conforming to.

A real-world capability description would likely be shorter than the presented example because the codec and transport definitions can be factored-out to profile definition documents that would only be referenced in capability description documents.

3.3 Referencing Definitions

This section specifies some generic mechanisms for referencing existing definitions. Referencing existing definition allows to contruct definitions without having to include all parameters inline. By using these mechanisms, complex definitions can be derived by

combining multiple basic mechanisms. Common parameters that occur in different configurations do not have to be repeated but can be defined once and then be referenced as often as they are needed.

3.3.1 The sdpng:use Element Type

The element type "sdpng:use" is a generic reference mechanisms that allows to refer to arbitrary definition within another definition or configuration element. "sdpng:use" is an element type with one mandatory attribute called "href". The value of that attribute is the name of the definition to be referenced. An example:

```
<def>
   <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>
</def>
<cfg>
       <c name="interactive-audio" media="audio">
     <alt name="alt-avp-audio-10">
       <rtp:session format="rtp-avp-10">
            <use href="endpoint-addr-1"/>
       </rtp:session>
     </alt>
      </c>
<cfg>
```

In this example, an element "rtp:udp" is used in the definitions section to define some transport parameters that should later be re-used by referencing this definition using the specified name "endpoint-addr-1". Within the element "rtp:session" in the configurations section the definition is referenced using the "use" element.

An implementation that processes this SDPng document and wants to evaluate the configuration for the alternative "rtp-avp-10" MUST replace the "use" element by the referenced element. If the referenced element contains "use" elements itself, those MUST also be dereferenced.

When applying this algorithm to the sample SDPng document, the following result SDPng document is generated:

```
   <def>
      <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>
   </def>
   <cfg>
          <c name="interactive-audio" media="audio">
           <alt name="alt-avp-audio-10">
             <rtp:session format="rtp-avp-10">
                <rtp:udp name="endpoint-addr-1" rtp-port="7800"
addr="224.2.0.53"/>
             </rtp:session>
           </alt>
         </c>
   <cfg>
```

For the purpose of comparing configurations, both SDPng documents are equal.

3.3.2 Properties

The element type "sdpng:prop" can be used to add properties to definitions. "sdpng:prop" has two attributes:

name: the name of the property

value: the value for the named property

For example:

```
   <def>
      <audio:codec name="g722" encoding="G722" channels="1" sampling="16000"/>
      <rtp:pt name="rtp-avp-9" pt="9" format="g722"/>
      <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>
   </def>
   <cfg>
          <c name="interactive-audio" media="audio">
           <alt name="alt-avp-audio-9-4">
             <rtp:session format="rtp-avp-9">
              <use href="endpoint-addr-1"/>
             </rtp:session>
             <prop name="foo" value="bar"/>
           </alt>
         </c>
   <cfg>
```

For comparing and collapsing elements, all sdpng:prop element that are contained in a parent element (like alt in the example above) MUST be transformed to attributes of the containing element. If the

parent element already provides a corresponding attribute its value MUST be overwritten.

The example above would thus be transformed to:

```
<def>
   <audio:codec name="g722" encoding="G722" channels="1" sampling="16000"/>
   <rtp:pt name="rtp-avp-9" pt="9" format="g722"/>
   <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>
</def>
<cfg>
       <c name="interactive-audio" media="audio">
        <alt name="alt-avp-audio-9-4" foo="bar">
          <rtp:session format="rtp-avp-9">
           <use href="endpoint-addr-1"/>
          </rtp:session>
        </alt>
      </c>
<cfg>
```

The main purpose of the sdpng:prop element type is to provide a mechanism by which attributes of referenced elements can be modified by the referring element. An application for this is described in Section 3.3.4.

3.3.3 Definition Groups

Using the sdpng:group element arbitrary definition can be combined and defined as a group with a specific name. Using this name, the definitions contained in the group can be referenced with the sdpng:use element and embedded into other elements.

An example for the use of the sdpng:group element:

```
   <def>
      <audio:codec name="g722" encoding="G722" channels="1" sampling="16000"/>
      <rtp:pt name="rtp-avp-9" pt="9" format="g722"/>
      <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>

      <group name="g1">
       <prop name="foo" value="bar"/>
       <prop name="xyz" value="uvw"/>
      </group>

   </def>
   <cfg>
          <c name="interactive-audio" media="audio">

           <alt name="alt-avp-audio-9-4">
             <rtp:session format="rtp-avp-9">
              <use href="endpoint-addr-1"/>
             </rtp:session>
             <use href="g1"/>
           </alt>
         </c>
   <cfg>
```

This example shows how a group that has been defined in the definitions section is referenced using the sdpng:use element. The group element contains two sdpng:prop elements.

For comparing and collapsing elements, all references to sdpng:group element MUST be replaced by the content of the corresponding sdpng:group element. The example above would thus be transformed to:

```
   <def>
      <audio:codec name="g722" encoding="G722" channels="1" sampling="16000"/>
      <rtp:pt name="rtp-avp-9" pt="9" format="g722"/>
      <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>

      <group name="g1">
       <prop name="foo" value="bar"/>
       <prop name="xyz" value="uvw"/>
      </group>

   </def>
   <cfg>
          <c name="interactive-audio" media="audio">

           <alt name="alt-avp-audio-9-4">
             <rtp:session format="rtp-avp-9">
              <use href="endpoint-addr-1"/>
             </rtp:session>
             <prop name="foo" value="bar"/>
             <prop name="xyz" value="uvw"/>
           </alt>
         </c>
   <cfg>
```

In this example the content of the sdpng:group element named g1 has been embedded into the alt element that contained the sdpng:use element referencing the group element.

According to the rules in Section 3.3.2 the sdpng:prop elements are transformed in a second step to yield the following final decription:

```
<def>
   <audio:codec name="g722" encoding="G722" channels="1" sampling="16000"/>
   <rtp:pt name="rtp-avp-9" pt="9" format="g722"/>
   <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>

   <group name="g1">
    <prop name="foo" value="bar"/>
    <prop name="xyz" value="uvw"/>
   </group>

</def>
<cfg>
       <c name="interactive-audio" media="audio">

        <alt name="alt-avp-audio-9-4" foo="bar" xyz="uvw">
          <rtp:session format="rtp-avp-9">
           <use href="endpoint-addr-1"/>
          </rtp:session>
        </alt>
      </c>
<cfg>
```

As a general rule, all references MUST be resolved before sdpng:prop elements are processed and transformed into attribute values.

3.3.4 Usage of Child Elements and Attributes of sdpng:use Elements

It is also possible to provide arbitrary other elements within a sdpng:use element (depending on the specific application). All elements that occur in a sdpng:use element MUST be transfomed to child elements of the referenced element when resolving a sdpng:use reference. If the reference already provides child elements, the child elements of the sdpng:use element are added to the list of child elements of the referenced element.

Any existing elements of a referenced element with the same GI as an element in the corresponding sdpng:use element MUST be replaced by the element of the sdpng:use element. This mechanism allows to extend and to change referenced elements in a simple way.

In the following we give an example of using an sdpng:prop element within a sdpng:use element which has the semantics of adding properties to the referenced element. The semantics and processing requirements for the sdpng:prop element are specified in Section 3.3.2.

Example for the usage of an sdpng:use element containing an

sdpng:prop element:

```
<def>
   <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>
</def>
<cfg>
       <c name="interactive-audio" media="audio">
        <alt name="alt-avp-audio-10">
          <rtp:session format="rtp-avp-10">
            <use href="endpoint-addr-1">
             <prop name="foo" value="bar"/>
            </use>
          </rtp:session>
        </alt>
      </c>
<cfg>
```

This will be transformed to:

```
<def>
   <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>
</def>
<cfg>
       <c name="interactive-audio" media="audio">
        <alt name="alt-avp-audio-10">
          <rtp:session format="rtp-avp-10">
             <rtp:udp name="endpoint-addr-1" rtp-port="7800"
addr="224.2.0.53">
               <prop name="foo" value="bar"/>
             </rtp:udp>
          </rtp:session>
        </alt>
      </c>
<cfg>
```

In a second step, the sdpng:prop element would be transformed to an attribute of its parent element (rtp:udp in this case) according to the rules specified in Section 3.3.2.

As an abbreviation, the properties for the referenced element do not have to be specified using sdpng:prop elements within the sdpng:use element but can also specified directly as attributes of the sdpng:use element, as shown in the following example:

```
<def>
   <rtp:udp name="endpoint-addr-1" rtp-port="7800" addr="224.2.0.53"/>
</def>
<cfg>
       <c name="interactive-audio" media="audio">
        <alt name="alt-avp-audio-10">
          <rtp:session format="rtp-avp-10">
            <use href="endpoint-addr-1" name="foo" value="bar"/>
          </rtp:session>
        </alt>
      </c>
<cfg>
```

In this example, the sdp:use element has no child element sdpng:prop but provides the property "foo" directly as an attribute. All attributes of a sdpng:use element other than href MUST be transformed to attributes of the referenced elements.

If the referenced element is a definition group (see Section 3.3.3), any child elements of an sdpng:use element MUST be transformed to child elements of the parent element of the sdpng:use element. Any properties (either explicit sdpng:prop elements or attributes of the sdpng:use element) MUST be transformed to properties of the parent element of the sdpng:use element.

3.4 External Definition Packages

There are two types of external definitions:

Profile Definitions (Section 3.4.1) define rules for specifying parameters that are not covered by the base SDPng specification.

Library Definitions (Section 3.4.2) contain definitions that can be referenced in SDPng documents.

3.4.1 Profile Definitions

In order to allow for extensibility it must be possible to define extensions to the basic SDPng configuration options.

For example, if some application requires the use of a new transport protocol, endpoints must be able to describe their configuration with respect to the parameters of that transport protocol. The mandatory and optional parameters that can be configured and negotiated when using the transport protocol will be specified in a definition document. Such a definition document is called a "profile".

A profile contains rules that specify how SDPng is used to describe conferences or end-system capabilities with respect to the parameters of the profile. The concrete properties of the profile definitions mechanism are still to be defined.

An example of such a profile would be the RTP profile that defines how to specify RTP parameters. Another example would be the audio codec profiles that defines how specify audio codec parameters.

SDPng documents can reference profiles and provide concrete definitions, for example the definition for the GSM audio codec. (This would be done in the "Definitions" section of an SDPng document.) An SDPng document that references a profile and provides concrete definitions of configurations can be validated against the profile definition.

3.4.2 Library Definitions

While profile definitions specify the allowed parameters for a given profile, SDPng "Definitions" sections refer to profile definitions and define concrete configurations based on a specific profile.

In order for such definitions to be imported into SDPng documents, "SDPng libraries" may be defined and referenced in SDPng documents. A library is a set of definitions that is conforming to one or more profile definitions.

The purpose of the library concept is to allow certain common definitions to be factored-out so that not every SDPng document has to include the basic definitions, for example the PCMU codec definition. SDP [2] uses a similar concept by relying on the well known static payload types (defined in RFC1890 [4]) that are also just referenced but never defined in SDP documents.

An SPDng document that references definitions from an external library has to declare the use of the external library. The external library, being a set of configuration definitions for a given profile, again needs to declare the use of the profile that it is conforming to. A library itself can make reference to other external libraries.

There are different possibilities of how profiles definitions and libraries can be used in SDPng documents:

- In an SPDng document, a profile definition can be referenced and all the configuration definitions are provided within the document itself. The SDPng document is self-contained with respect to the definitions it uses.

- In an SPDng document, the use of an external library can be declared. The library references a profile definition and the SDPng document references the library. There are two alternatives how external libraries can be referenced:

  by name: Referencing libraries by names implies the use of a registration authority where definitions and reference names can be registered with. It is conceivable that the most common SDPng definitions be registered that way and that there will be a baseline set of definitions that minimal implementations must understand. Secondly, a registration procedure will be defined, that allows vendors to register frequently used definitions with a registration authority (e.g., IANA) and to declare the use of registered definition packages in conforming SDPng documents. Of course, care should be taken not to make the external references too complex and thus require too much a priori knowledge in a protocol engine implementing SDPng. Relying on this mechanism in general is also problematic because it impedes the extensibility, as it requires implementors to provide support for new extensions in their products before they can inter-operate. Registration is not useful for spontaneous or experimental extensions that are defined in an SDPng library.

  by address: An alternative to referencing libraries by name is to declare the use of an external library by providing an address, i.e., an URL, that specifies where the library can be obtained. While this allows the use of arbitrary third-party libraries that can extend the basic SDPng set of configuration options in many ways, in introduces additional complexity that could result in in higher latency for the processing of a description document with references to external libraries. In addition, there are problems if the referenced libraries cannot be accessed by all communication partners.

- Because of these problematic properties of external libraries, the final SDPng specification will have to provide a set of recommendations under which circumstances the different mechanisms of referring to external definitions should be used.

3.5 Mappings

A mapping needs to be defined in particular to SDP that allows to translate final session descriptions (i.e. the result of capability negotiation processes) to SDP documents. In principle, this can be done in a rather schematic fashion for the basic definitions.

In addition, mappings to H.245 will be defined in order to support applications like SIP-H.323 gateways.

4. Capability Negotiation

SDPng is a description language for both potential configurations (i.e. capabilities) of participants in multimedia conferencers and for actual configurations (i.e. final specifications of parameters). Capability negotiation is the process of generating a usable set of potential configurations and finally an actual configuration from a set of potential configurations provided by each potential participant in a multimedia conference.

SDPng supports the specification of endpoint capabilities and defines a negotiation process: In a negotiation process, capability descriptions are exchanged between participants. These descriptions are processed in a "collapsing" step which results in a set of commonly supported potential configurations. In a second step, the final actual configuration is determined that is used for a conference. This section specifies the usage of SDPng for capability negotiation. It defines the collapsing algorithm and the procedures for exchanging SDPng documents in a negotiation phase.

The description language and the rules for the negotiation phase that are defined here are (in general) independent of the means by which descriptions are conveyed during a negotiation phase (a reliable transport service with causal ordering is assumed). There are however properties and requirements of call signalling protocols that have been considered to allow for a seamless integration of the negotiation into the call setup process. For example, in order to be usable with SIP, it must be possible to negotiate the conference configuration within the three-way-handshake of the call setup phase. In order to use SDPng instead of SDP according to the offer/answer model defined in [15] it must be able to determine an actual configuration in a single request/response cycle.

4.1 Outline of the Negotiation Process

Conceptually, the negotiation process comprises the following individual steps (considering two parties, A and B, where A tries to invite B to a conference). Please note that is describes the steps of the negotiation process conceptually -- it does not specify requirements for implementations. Specific procedures that MUST be followed by implementations are given below.

1. A determines its potential configurations for the components that should be used in the conference (e.g. "interactive audio" and "shared whiteboard") and sends a corresponding SDPng instance to B. This SDPng instances is denoted "CAP(A)".

2. B receives A's SDPng instance and analyzes the set of components

(sdpng:c elements) in the description. For each component that B wishes to support it generates a list of potential configurations corresponding to B's capabilities, denoted "CAP(B)".

3. B applies the collapsing function and obtains a list of potential configurations that both A and B can support, denoted "CAP(A)xCAP(B) = CAP(AB)".

4. B sends CAP(B) to A.

5. A also applies the collapsing function and obtains "CAP(AB)". At this step, both A and B know each other capabilities and the potential configurations that both can support.

6. In order to obtain an actual configuration from the potential configuration that have been obtained, both particpants have to pick a subset of the potential configurations should actually be used in the conference and generate the actual configuration. It should be noted that it depends on the specific application whether each component must be assigned exactly one actual configuration (one sdpng:alt element) or whether it is allowed to list multiple actual configurations. In this model we assume that A selects the actual configuration, denoted CFG(AB).

7. A augments CFG(AB) with the transport parameters it intends to use, e.g., on which endpoint addresses A wishes to receive data, obtaining CFG_T(A). A sends CFG_T(A) to A.

8. B receives CFG_T(A) and adds its own transport parameters, resulting in CFG_T(AB). CFG_T(AB) contains the selected actual configurations and the transport parameters of both A and B (plus any other SDPng data, e.g., meta-information on the conference). CFG_T(AB) is the complete conference description. Both A and B now have the following information:

   CAP(A) A's supported potential configurations

   CAP(B) B's supported potential configurations

   CAP(AB) The set of potential configurations supported by both A and B.

   CFG(AB) The set of actual configurations to be used.

   CFG_T(AB) The set of actual configurations to be used augmented with all required parameters.

In this model, the capability negotiation and configuration exchange

process leads to a description that represents a global view of the configuration that should be used. This means, it contains the complete configuration for all participants including per-participant information like transport parameters.

Note that the model presented here results in four SDPng exchanges. As an optimization, this procedure can be abbreviated to two exchanges by including the transport (and other) parameters into the potential configurations. A embeds its desired transport parameters into the list of potential configurations and B also sends all required parameters in the response together with B's potential configurations. Both A and B can then derive CFG_T(AB). Transport parameters are usually not negotiable, therefor they have to be distingiushed them from other configuration information.

Specific procedures for re-negotiation and multi-party negotiation will be defined in a future version of this document.

4.2 The Collapsing Algorithm

The following procedure MUST be used for the collapsing of two SDPng document instances into one:

CAP(A) and CAP(B) are the two SDPng description document instances. For each component (sdpng:c element) in CAP(A) there is a corresponding component in CAP(B). Components MAY be empty (containing no sdpng:alt elements) which means that there is no potential configuration and the component should not be used in the conference.

Let cfg_AB be the result configuration element, initialized to an empty sdpng:cfg element.

1. For each component (sdpng:c element) in CAP(A) named c_A

    * Let c_AB be the current result component, initialized to an empty sdpng:c element.

    * For each alternative (sdpng:alt element) in c_A named a_A

        + For each session element (name depends on the profile being used) in a_A named s_A

            - Resolve any reference to definition elements recursively and obtain s1_A, the standalone media session description. (Refer to Section 4.2.1 for a description of how to resolve references.)

- Locate the component element that matches c_A in CAP(B) named (c_B).

- Let a_AB be the current result alternative, initialized to an empty sdpng:alt element.

- For each alternative (sdpng:alt element) in c_B named a_B

  o For each session element (name depends on the profile being used) in a_B named s_B

    * Let s1_AB be the computed result media session configuration.

    * Resolve any reference to definition elements recursively and obtain s1_B, the standalone media session description.

    * Apply collapse(s1_A,s2_B) to compute s1_AB, the collapsed media session configuration.

    * If s1_AB is not empty, add s1_AB to a_AB, the set of sessions for the current result alternative.

- If a_AB is not empty, add a_AB to c_AB.

* If c_AB is not empty, add c_AB to cfg_AB.

The collapsing function for collapsing two elements is specified in Section 4.2.1.

4.2.1 Collapsing Two Configurations

Before two media session configuration element can be collapsed as described in Section 4.2 all references to definitions MUST be resolved. This MUST be performed recursively, i.e. references in definitions MUST also be resolved. For resolving references, the algorithm specified in Section 3.3 MUST be used.

By resolving all references two intermediate session configuration elements are obtained that can then be collapsed according to the algorithm specified in the following sections.

4.2.1.1 Collapsing of Attributes

In SDPng, capabilities are specified in attributs of XML elements. Three different types of capabilities with different collapsing rules

are defined. The type of a capability is encoded in the attribute value.

Set of symbols:

An attribute can specify a set of symbols. When two attributes are collapsed the result is the intersection of the two sets.

The following examples shows how two elements (with one attribute representing a set of symbols) originated from two capability descriptions from participants A and B are collapsed:

```
Element x in A's capability description:
<x a="[FOO, BAR, 3, 5, 8]"/>

Element x in B's capability description:
<x a="[3, 6, 8]"/>

Result:
<x a="[3, 8]"/>
```

If the intersection result in an empty set the collapsing process has failed and there is no common set of values. If the collapsing of one of an element's attributes with the type "set of symbols" has failed, the collapsing process of the element itself MUST be considered to have failed as well.

Numerical ranges:

An attribute can also specify a numercial range. When two attributes are collapsed the result is the range of values that represents the intersection of the set of values that is included in both ranges.

The following examples shows how two elements (with one attribute representing a numerical range) originated from two capability descriptions from participants A and B are collapsed:

```
Element x in A's capability description:
<x a="(2,8)"/>

Element x in B's capability description:
<x a="(5,10)"/>

Result:
<x a="(5,8)"/>
```

A numerical range is represented by a tuple of comma-separated

numbers in brackets. The first number represents the lower bound of the range and the second number represents the upper bound. Let MIN(a,b) be a function that returns the minimum of a and b and MAX(a,b) be a function that returns the maximum of a and b. Given two ranges (minA, maxA) and (minB, maxB), the collapsed new range MUST be calculated using this algorithm:

(MAX(minA, minB), MIN(maxA, maxB))

If this process results in a range with a smaller first value, the range is invalid and the collapsing has failed since there is no common range. If the collapsing of one of an element's attributes with the type "numerical range" has failed, the collapsing process of the element itself MUST be considered to have failed as well.

Optional parameters:

A failure of collapsing attributes of the types "set of symbols" and "numerical range" results in a failure of collapsing the corresponding element. There is a third type named "optional parameter" defined, that provides different collapsing rules. An optional parameter is an attribute with an arbitrary value. When collapsing two attributes of this type, their values MUST be tested for equality. If they are equal, the collapsing has been successful and the attribute MUST appear as is in the result description. If the attributes' values are different, the collapsing is considered to have failed and the attribute MUST not appear in the result description. However, a failure in collapsing an attribute of type "optional parameter" does not affect the collapsing of the containing element.

An example for a successful collapsing:

```
Element x in A's capability description:
<x a="foo"/>

Element x in B's capability description:
<x a="foo"/>

Result:
<x a="foo"/>
```

An example for an unsuccessful collapsing:

```
Element x in A's capability description:
<x a="foo"/>

Element x in B's capability description:
<x a="bar"/>

Result:
<x/>
```

4.2.1.2 Collapsing two Elements

In order to collapse two elements with multiple attributes, the following algorithm specified below MUST be applied. In general, the collapsing of two elements (if successful) yields a result element that contains the collapsed attributes. If the collapsing of two elements has failed, no result element is generated.

1. For each attribute, determine the type and collapse the attribute by applying the algorithm for the corresponding attribute type.

2. If an attribute with a different type than "optional parameter" does not occur in both elements, the collapsing for this element MUST be considered to have failed.

3. If the collapsing of any attribute with a different type than "optional parameter" has failed, the collapsing of the element itself MUST be considered to have failed.

4. If the collapsing has been successful, obtain the result element by using the same element name (GI) and the attributes with their collapsed values. Exclude any attribute of type "optional parameter" that has failed to collapse.

An example:

```
Element x in A's capability description:
<x a="[FOO, BAR, 3, 5, 8]" b="(2,8)" c="foo"/>

Element x in B's capability description:
<x a="[3, 6, 8]" b="(5,10)" c="bar"/>

Result:
<x a="[3, 8]" b="(5,8)"/>
```

4.2.1.3 Collapsing nested Elements

In order to collapse nested elements the following algorithm MUST be applied:

In analogy to attributes representing optional parameters there is also the possibility to mark elements as optional for the negotiation process. Elements MAY provide an attribute names "status" that contains a symbol or a comma-separated list of symbols as its value. If the value "opt" occurs in the list of a "status" attribute of both elements to be collapsed, the elements to be collapsed are treated as optional. This means, if the collapsing of the attributes has failed (according to the rules specified in Section 4.2.1.2), the collapsing process does not yield a result element but is still treated as "successful", i.e., further collapsing operation on other elements can continue. The semantics of optional elements are that they describe optional features that may be supported and selected during a negotiation phase but do not neccessarily have to be supported by all participants in order to achieve interoperability. The example below shows how to generate a result element in the presence of optional child elements that have failed to collapse.

The collapsing algorithm for nested elements:

1. Let x be an element that occurs in the capability description of two participants A and B and that should be collapsed.

2. Collapse the attributes of the element x using the algorithm specified in Section 4.2.1.2. If the collapsing has failed according to the rules of Section 4.2.1.2 and if the elements to be collapsed are not marked as optional, the collapsing of the element and all of its children MUST be considered to have failed. The collapsing MUST be stopped. If the collapsing has failed and both elements have been marked as optional, the child elements MUST NOT be processed. In this case, the collapsing process does not yield a result element but the collapsing of other elements (sibling or parent elements) MUST be continued.

3. If the collapsing has been successful according to the rules of Section 4.2.1.2, the child elements of A's and B's x element MUST be processed. If there are no child elements in both A's and B's content the collapsing has been successful and can be terminated. If either A's or B's x element provides child elements, apply the following algorithm to each child element named c of participant A's element x:

    1. Find a corresponding element (same GI) in the set of participant B's child elements. If no matching element has

been found, the collapsing of element x MUST be considered to have failed.

2. If a matching element has been found, apply the collapsing algorithm recursively. As long as the collapsing is successful, the result of collapsing each element is transferred to the result element, such that the resulting element tree is isomorphic to both A's and B's element tree.

If there are elements in B's x element that have not been processed (because there is no corresponding element in A's x element), the collapsing MUST be considered to have failed and MUST be stopped.

An example:

```
Element x in A's capability description:
<x a="[FOO, BAR, 3, 5, 8]" b="(2,8)" c="foo">
  <y b="[UVW, XYZ]"/>
</a>

Element x in B's capability description:
<x a="[3, 6, 8]" b="(5,10)" c="bar">
  <y b="[RST, XYZ]"/>
</a>

Result:
<x a="[3, 8]" b="(5,8)">
  <y b="[XYZ]"/>
</a>
```

An example for collapsing optional elements:

```
Element x in A's capability description:
<x a="[FOO, BAR, 3, 5, 8]" b="(2,8)" c="foo">
  <y status="opt" b="[UVW, XYZ]"/>
</a>

Element x in B's capability description:
<x a="[3, 6, 8]" b="(5,10)" c="bar">
  <y status="opt" b="[RST]"/>
</a>

Result:
<x a="[3, 8]" b="(5,8)"/>
```

4.2.2 Deriving an actual Configuration

The result of a capability negotiation process is a potential configuration, i.e., a description potentially containing multiple alternatives per component. The alternative themselves may provide elements that represent collapsed capabilities. In order to derive an actual configuration, the following problems must be addressed:

1. For each component (sdpng:c element) an appropriate alternative (sdpng:alt element) has to be selected. It is conceivable that the order of the alternatives in the description is used as a preference indicator. More details have to be specified in a future version of this document.

2. If the description of the selected alternatives contains attributes with numerical ranges or sets of symbols with more than one entry, those attributes either have to be transformed that they represent a single value or participants have to agree that an actual configuration may contain ranges and sets of symbols. The semantics of these variable actual configurations will have to specified in later versions of this document. For example, for certain applications it may be desireable to agree on ranges of values for certain attributes during a capability negotiating meaning that any of the values of the range are supported (and have to be supported).

The specific procedures to determine an actual configuration have to be defined in a later version on this document.

5. Formal Specification

This section defines the SDPng syntax and the use of XML mechanisms, such as XML Namespace and XML Schema. Section 5.1 defines the relation between SDPng and XML Schema, Section 5.2 specifies general requirements for documents and profile definitions that are conforming to the SDPng schema, Section 5.3 list requirements for profile definitions, Section 5.4 specifies specific requirements for conforming documents and Section 5.5 lists requirements for the definition of SDPng libraries.

Section 5.7 defines the SDPng base schema, Section 5.7.2 defines the profile for audio codec definitions and Section 5.7.3 defines the profile for RTP payload type definitions.

5.1 XML Schema as a Definition Mechanism

SDPng documents reference profile schema definitions and libraries. Profile schema definitions contain schema definitions of SDPng document elements. For example, the general structure is specified by a schema definition and extensions to SDPng for specific applications are specified as schema definitions as well.

The baseline SDPng specification consists of a profile (a schema definition) and a library of commonly used definitions.

SDPng uses XML-Schema [13][14] for defining the possible logical structures of SDPng documents for the following reasons:

Extensibility: XML-Schema provides mechanisms that allow to extend existing definitions allowing to uniquely identify element types (by relying on XML namespaces [11]).

Modularity: XML-Schema provide mechanisms that allow to organize schema definitions in multiple components.

Expressiveness: XML-Schema provides many data types, that can be refined by user-supplied definitions.

SDPng documents MUST be schema instances of the SDPng schema as defined in Section 5.7. The following example shows how a Schema definition can be referenced in a document instance.

Beginning of an SDPng-document:

```
<?xml version="1.0" ?>
<sdpng:desc xmlns:sdpng="http://www.iana.org/sdpng"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
         xsi:schemaLocation="http://www.iana.org/sdpng sdpng.xsd">
```

XML-Schema specifies that documents can assign a namespace when referencing a schema definition. A SDPng namespace is defined for this purpose. The name of this namespace is "http://www.iana.org/sdpng". A well-known namespace prefix is used for the SDPng schema definition, in order to allow for very simple implementations. The well-known SDPng namespace prefix is "sdpng". Conforming Documents, profile definition and libraries MUST use this namespace name and this namespace prefix.

For SDPng documents, this initial declaration can be added implicitly by applications, so that declarations like the one above do not have to be included in every description document. Details are to be defined in a later version of this document.

5.2 SDPng Schema

The basic SDPng schema definitions specifies the general document structures, e.g., one "Definitions" section followed by one "Configurations" sections, followed by one "Constraints" sections followed by a "Conference" section (for meta-information). Each document MUST provide the elements for definitions, configurations, constraints and conference information in exactly this order, whereby only the configurations section is MANDATORY. Refer to Section 5.7 for a formal definition of the SDPng base schema and the specific element types for definitions, configurations, constraints and conference information.

The SDPng base schema also specifies "abstract" base data types (by means of XML-Schema type definitions) for elements that MUST be used by documents in the corresponding sections. The base data types provide common required attributes, e.g. a "name" attribute for naming definition elements.

Example:
The following example shows the definition of the base type for definition elements:

```
<xsd:complexType name="Definition" abstract="true" mixed="false">
<xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

Profiles can then define specific types that augment the base type definitions. Common attributes or content models, that have been

defined by this base definition, do not have to be provided by those concrete type definitions. The type definitions can be identified as allowed element types for the content models that are specified in the base SDPng schema definition. This allows for automatic validation of profile definitions and facilitates the extension of SDPng.

5.3 Profiles

The baseline SDPng specification consists of a profile (a schema definition) and a library of commonly used definitions.

The library of commonly used definitions provides data types for IP (and other) addresses.

A profile definition MUST import (using the XML-Schema import mechanism) the base SDPng schema definition and MUST provide an extension definition, e.g., specializations of base element types. A profile definition MUST also provide a target namespace name for its definitions. For well-known (registered) profiles, the namespace name will be registered by IANA. Proprietary profiles will use other namespace names, for example, based on domain names, that are registered by vendors providing a profile.

Example:
The following example shows such a declaration at the beginning of a profile definition:

```
<xsd:schema targetNamespace="http://www.iana.org/sdpng/audio"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:sdpng="http://www.iana.org/sdpng"
         xmlns:audio="http://www.iana.org/sdpng/audio">

  <xsd:import namespace="http://www.iana.org/sdpng"
              schemaLocation="sdpng.xsd"/>
```

In this example, the namespace prefix "audio" is defined and later used in schema definitions. (The example profile provides definition mechanisms for audio codecs.)

The following example shows, how a derived type for "definition" elements can be specified with XML-Schema mechanisms. In this case, the abstract type "Definition" (fully qualified as "sdpng:Definition") is augmented by three attributes that are useful for defining audio codecs.

Example:

```
<xsd:complexType name="AudioCodec" mixed="false">
  <xsd:complexContent>
    <xsd:extension base="sdpng:Definition">
     <xsd:attribute name="encoding" type="xsd:string"/>
     <xsd:attribute name="sampling" type="xsd:positiveInteger"/>
     <xsd:attribute name="channels" type="xsd:positiveInteger"/>
    </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

This type definition is then used to define an XML element type called "codec".

Example:

```
         <xsd:element name="codec" type="AudioCodec"/>
```

When used by SDPng documents, the general identifier is qualified with a namespace prefix, for example as in: "audio:codec".

5.4 SDPng Documents

SDPng documents MUST reference the employed profiles and provide namespace prefixes for the namespace names of the profiles as shown in the following example.

Example:

```
<sdpng:desc xmlns:sdpng="http://www.iana.org/sdpng"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
         xsi:schemaLocation="http://www.iana.org/sdpng sdpng.xsd"
         xmlns:audio="http://www.iana.org/sdpng/audio"
         xmlns:rtp="http://www.iana.org/sdpng/rtp">
```

For well-known registered profiles, the namespace name AND the used namespace prefix SHOULD be registered to allow for simple basic implementations that can match identifiers by using fixed fully qualified names without having to interpret namespace declarations (see Section 5.6.3). There is one issue with declaring used XML-Schema definitions in documents (see Section 7 below).

The general structure of an SDPng documents MUST conform to the basic SDPng schema definition and MAY provide a "def" element for definitions; it MUST provide a "cfg" element for the configuration section; it MAY provide a "constraints" and a "conf" element.

Example:
The following example shows a sample definition section where the

element "codec" of the "audio codec profile" is used (plus the element type "pt" of an "RTP profile"):

```
    <def>
      <audio:codec name="dvi4" encoding="DVI4" channels="1"
                   sampling="8000"/>
      <audio:codec name="g722" encoding="G722" channels="1"
                   sampling="16000"/>
      <audio:codec name="g729" encoding="G729" channels="1"
                   sampling="8000"/>

      <rtp:pt name="rtp-avp-18" pt="18" format="g729"/>
    </def>
```

It can be seen how the attribute name (provided by the base type for definition elements) and the profile specific attributes "encoding", "channels" and "sampling" are used together.

The element "rtp:pt" is used to defined a payload type. "rtp:pt" would have been defined in another profile, again using a type derived from the base definition type. "rtp:pt" provides attribute for referencing other definitions, e.g., the definition of audio-codes as seen before.

5.5 Libraries

SDPng libraries are collections of definitions that are referenced by documents. Libraries are thus independent, valid SDPng documents.

For example, the definition of the different audio codecs as shown in the previous example could be provided by a library that can be referenced by documents without having to define such common codecs in every document.

The XML mechanism XInclude [12] is used for referencing libraries in SDPng documents. XInlcude works at the XML Information Set ("infoset") level, i.e. the mechanisms allows to have an integrating document reference fragment documents, while these fragments are well-formed (and, if applicable, valid) documents themselves. By resolving XInclude directives in integrating documents the documents' infosets are "merged" together, enabling applications to operate on the resulting infosets as if it had been generated by parsing a single, monolithic document.

Inclusion at the XML infoset level has the advantage that documents are standalone -- they can be validated independently. Another advantage is that is relatively easy to generate a "merged" infoset for applications that are not able to resolve references to libraries

themselves.

An alternative for XInclude would be to use references that are resolved by applications. For XML, this would probably mean to use an XLink-based approach. This solution would require the definition of an SDPng link element type and require applications to support XLink (or at least the SDPng-relevant subset thereof). The inclusion at the application level is however problematic, because it does not result in a common integrated XML document infoset but would require applications to handle multiple infosets, i.e. multiple documents.

5.6 Details on the use of specific XML Mechanisms

This section specifies the use of specific XML mechanisms for SDPng. In order to allow for efficient parsing and processing, not all features of XML Schema are allowed. Some variable information is set to fixed values to allow the development of simplistic servers.

5.6.1 Default Namespace

SDPng document instances MUST use the SDPng namespace "http://www.iana.org/sdpng". That means, the general SDPng identifiers can be used without namespace prefixes.

5.6.2 Qualified Locals

XML Schema allows to specify qualification of elements and attributes. It is possible to use non-qualified element and attribute names in Schema definitions and document instances for so-called "local definitions" (this is the default setting). "Local Definitions" are contained within "global definitions" in an XML schema definition. In order to simplify parsing and processing of SDPng document instances, all elements MUST be fully qualified. Attribute names MUST NOT be fully qualified, they are considered to have the same namespace as their corresponding elements.

This means, the SDPng Schema definition contains the following attributes for the "schema" element, that MUST also be used by SDPng profiles:

- elementFormDefault="qualified"
  This means that "locally defined" elements that are used within the scope of fully-qualified elements MUST always be fully qualified as well.

- attributeFormDefault="unqualified"
  This means that attribute names do not have to be fully qualified. Implementations MUST infer the namespace for attributes from the

namespace of the element that the attribute belongs to. Note that the specification of XML Namespaces [11] defines that default namespaces do not apply to attributes. In profile definitions, all attributes MUST be defined locally. The same holds for the base SDPng schema.

These rules make SDPng document instances process-able by non-Schema-aware XML parsers by requiring all element names to be fully qualified (no "local elements").

5.6.3 Fixed Namespace Prefixes

In order to facilitate the development of basic implementations, a few commonly used namespaces names are associated with fixed prefixes, i.e. document instances and libraries MUST always use these prefixes. These prefixes MUST NOT be used for namespaces names than the ones that are assigned to them. In order to ensure the uniqueness of namespace prefixes, namespace prefixes will be have to registered together with the corresponding namespace names.

The namespace prefix for the SDPng namespace is "sdpng".

5.7 SDPng Schema Definitions

This section provides the definition of the base SDPng XML Schema.

1. Section 5.7.1 contains the base definition that provides the general element types for SDPng.

2. Section 5.7.2 contains a profile for audio codecs.

3. Section 5.7.3 contains a profile for RTP payload type definitions.

5.7.1 SDPng Base Definition

This schema definition defines the general structure of SDPng document instances. It defines the top-level element type "desc" that can have a sequence of "def", "cfg", "constraints" and "conf" elements as element content.

In addition, "extensions hooks" are provided that can be used by extension profiles providing definitions for specific applications. In general, these extension are implemented by deriving profile definitions from SDPng base definitions. The deployed XML Schema mechanisms are "deriving by extension" and "substitution groups". The SDPng base definition provides different base types (as

complexType definitions) for elements that are to be used in "def", "cfg" and "conf" sections. In addition, it also defines specific element types as "head elements" with assigned types that are used for defining the content model of, e.g., the "def" element type.

Profiles that provide new element types for specific applications will define types that are derived from the base types and provide the additional attributes and element content definitions that are required for the application. The XML element types that are defined in a profile are declared as valid substitutes for the base elements ("head elements") by setting the "substitutionGroup" attribute to the name of the "head element" type.

For an extension-profile that provides new definition element types, e.g. for codec definitions, a new complexType would be defined that extends sdpng:Definition (see below). An element type definition that assigns that new type must then be declared to be in the substitutionGroup "sdpng:d".

This mechanism allows common rules for attributes and content models to be defined in base element definition and re-used by extension profiles and it also allows validating parsers to identify the correct type of elements that have been defined by profile definitions.

The SDPng Base Definition:

```
<xsd:schema targetNamespace="http://www.iana.org/sdpng"
         xmlns:sdpng="http://www.iana.org/sdpng"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         elementFormDefault="qualified"
         attributeFormDefault="unqualified">

  <xsd:annotation>
    <xsd:documentation>
      This schema definition defines the general structure of SDPng
      document instances. It provides base type and base element
      definition for elements to occur in the different sections (def,
      cfg, constraints, conf) to be derived from in extension-profile
      definitions.

      For an extension-profile that provides new definition element
      types, e.g. for codec definitions, a new complexType would be
      defined that extends sdpng:Definition (see below). An element
      type definition that assigns that new type must then be declared
      to be in the substitutionGroup "sdpng:d".
    </xsd:documentation>
  </xsd:annotation>
```

```
    <xsd:element name="desc">
      <xsd:annotation>
        <xsd:documentation>
       The top-level element of an SDPng document. It defines the
       overall structure of an SPDng document.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType mixed="false">
        <xsd:sequence>
       <xsd:element ref="sdpng:def" minOccurs="0"/>
       <xsd:element ref="sdpng:cfg"/>
       <xsd:element ref="sdpng:constraints" minOccurs="0"/>
       <xsd:element ref="sdpng:conf" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:element name="def">
      <xsd:annotation>
        <xsd:documentation>The definitions section</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType mixed="false">
        <xsd:sequence>
       <xsd:element ref="sdpng:d" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:element name="cfg">
      <xsd:annotation>
        <xsd:documentation>The configurations section</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType mixed="false">
        <xsd:sequence>
       <xsd:element ref="sdpng:c" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:element name="constraints">
      <xsd:annotation>
        <xsd:documentation>The constraints section</xsd:documentation>
```

```
      </xsd:annotation>
      <xsd:complexType mixed="false">
        <xsd:sequence>
       <xsd:element ref="sdpng:cn" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:element name="conf" type="sdpng:Conference">
      <xsd:annotation>
        <xsd:documentation>The conference section</xsd:documentation>
      </xsd:annotation>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:element name="d" type="sdpng:Definition">
      <xsd:annotation>
        <xsd:documentation>
       Placeholder base element for a definition element in the
       definitions section. To be derived from by specific definition
       element type definitions.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:element name="c" type="sdpng:Component">
      <xsd:annotation>
        <xsd:documentation>
       Placeholder base element for a configuration element in the
       configurations section. To be derived from by specific
       configuration element type definitions.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:element name="cn" type="sdpng:Constraint">
      <xsd:annotation>
        <xsd:documentation>
       Placeholder base element for a contraint element in the
       contraints section. To be derived from by specific constraint
       element type definitions.
```

```
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:complexType name="Definition" abstract="true" mixed="false">
      <xsd:annotation>
        <xsd:documentation>
       The base type for definition. Defines a attribute "name" for
       naming definitions.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="name" type="xsd:string"/>
    </xsd:complexType>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:complexType name="Component" mixed="false">
      <xsd:annotation>
        <xsd:documentation>
       The specification of a component consists of a sequence of
       alternatives.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element ref="sdpng:alt" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:attribute name="media" type="xsd:string"/>
    </xsd:complexType>

    <xsd:element name="alt">
      <xsd:annotation>
        <xsd:documentation>
       Each alternative consists of a "sc" (session configuration)
       element. The "sc" element is a base element of base type
       "sdpng:Session" that is used to derive specific session types
       in extension profiles.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType mixed="false">
        <xsd:sequence>
       <xsd:element ref="sdpng:sc" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
```

```
    <xsd:element name="sc" type="sdpng:SessionConfig"/>


    <xsd:complexType name="SessionConfig" abstract="true" mixed="false">
      <xsd:annotation>
        <xsd:documentation>
       The (abstract) base type for session elements. To be derived
       from in extension profiles.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:complexType>

  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <xsd:complexType name="Constraint" mixed="false">
      <xsd:annotation>
        <xsd:documentation>
       The current content model for constraints is a sequence of
       "sdpng:par" elements. In each "par" element a sequence of
       "use-alt" elements may be used to specific the definitions
       that may used in parallel. Each "use-alt" element can define
       the number of minimum and maximum instantiations.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element ref="sdpng:par"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="par">
      <xsd:complexType mixed="false">
        <xsd:sequence>
       <xsd:element ref="sdpng:use-alt">
       </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>


    <xsd:element name="use-alt">
      <xsd:complexType mixed="false">
        <xsd:attribute name="ref" type="xsd:string"/>
        <xsd:attribute name="min" type="xsd:positiveInteger"
                       use="optional"/>
        <xsd:attribute name="max" type="xsd:positiveInteger"
                       use="optional"/>
      </xsd:complexType>
    </xsd:element>
```

```
   <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

     <xsd:complexType name="Conference" mixed="false">
       <xsd:sequence>
         <xsd:element name="meta" type="sdpng:ConfItem"/>
       </xsd:sequence>
       <!-- TBD -->
     </xsd:complexType>

     <xsd:complexType name="ConfItem" abstract="true" mixed="false">
       <xsd:annotation>
         <xsd:documentation>
        The base type for conference meta inforformation
        element. Currently, there is no common content model defined.
         </xsd:documentation>
       </xsd:annotation>
     </xsd:complexType>

     <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

     <xsd:element name="owner">
       <xsd:complexType mixed="false">
         <xsd:complexContent mixed="false">
        <xsd:extension base="sdpng:ConfItem">
          <xsd:attribute name="user" type="xsd:string"/>
          <xsd:attribute name="session-id" type="xsd:string"/>
          <xsd:attribute name="version" type="xsd:string"/>
          <xsd:attribute name="nettype" type="xsd:string"/>
          <xsd:attribute name="addrtype" type="xsd:string"/>
          <xsd:attribute name="addr" type="xsd:string">
            <!-- FIXME: re-use common address type! -->
          </xsd:attribute>
        </xsd:extension>
         </xsd:complexContent>
       </xsd:complexType>
     </xsd:element>

   <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

     <xsd:complexType name="SimpleLink" mixed="false">
       <xsd:attribute name="xlink:type" type="xsd:string" fixed="simple"/>
       <xsd:attribute name="xlink:role" type="xsd:string"/>
       <xsd:attribute name="xlink:arcrole" type="xsd:string"/>
       <xsd:attribute name="xlink:title" type="xsd:string"/>
       <xsd:attribute name="xlink:show" type="xsd:string" fixed="none"/>
       <xsd:attribute name="xlink:actuate" type="xsd:string" fixed="none"/>
       <xsd:attribute name="xlink:href" type="xsd:string"/>
     </xsd:complexType>
```

```
    <xsd:element name="session">
      <xsd:complexType mixed="false">
        <xsd:complexContent mixed="false">
       <xsd:extension base="sdpng:ConfItem">
         <xsd:sequence>
           <xsd:element name="description" type="xsd:string"/>
           <xsd:element name="info" type="sdpng:SimpleLink"/>
           <xsd:sequence minOccurs="0">
             <xsd:element name="contact" type="sdpng:SimpleLink"/>
           </xsd:sequence>
         </xsd:sequence>
       </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```

5.7.2 Audio Codec Profile

The following profile defines an element type that can be used for specifying audio codec characteristics. The element "audio:codec" is of type "audio:AudioCodec" which is derived from the SDPng base type "sdpng:Definition". The element "audio:codec" is declared to have the substitution group "sdpng:d" (the "head element" of the SDPng base definition).

This means, "audio:codec" element can be used as child elements in "sdpng:def" elements. In addition to the attributes specified here "audio:codec" elements will also have to provide a "name" attribute as defined by "sdpng:Definition".

```
<xsd:schema targetNamespace="http://www.iana.org/sdpng/audio"
         xmlns:audio="http://www.iana.org/sdpng/audio"
         xmlns:sdpng="http://www.iana.org/sdpng"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         elementFormDefault="qualified"
         attributeFormDefault="unqualified">

  <xsd:import namespace="http://www.iana.org/sdpng"
           schemaLocation="sdpng.xsd"/>

  <!-- AudioCodecs extends the abstract type "Definition" -->
  <!-- The data types for the attributes could be more restrictive... -->
  <xsd:complexType name="AudioCodec" mixed="false">
    <xsd:complexContent mixed="false">
      <xsd:extension base="sdpng:Definition">
     <xsd:attribute name="encoding" type="xsd:string"/>
     <xsd:attribute name="sampling" type="xsd:positiveInteger"/>
     <xsd:attribute name="channels" type="xsd:positiveInteger"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>


<xsd:element name="codec" substitutionGroup="sdpng:d">
    <xsd:complexType>
      <xsd:complexContent>
     <xsd:extension base="audio:AudioCodec"/>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

5.7.3 RTP profile

The following profile defines element types that can be used for specifying RTP payload types and RTP session configurations. The element "rtp:pt" is of type "rtp:PayloadType" which is derived from the SDPng base type "sdpng:Definition". The element "rtp:pt" is declared to have the substitution group "sdpng:d" (the "head element" of the SDPng base definition).

The element "rtp:session" is of type "rtp:Session" which is derived from the SDPng base type "sdpng:SessionConfig". The element "rtp:session" is declared to have the substitution group "sdpng:sc" (the "head element" of the SDPng base definition).

The RTP profile in turn defines base types for the specification of transport parameters that are to be derived from by profiles that define rules for elements that can be used to specify parameters for specific transport mechanisms.

```
<xsd:schema targetNamespace="http://www.iana.org/sdpng/rtp"
         xmlns:rtp="http://www.iana.org/sdpng/rtp"
         xmlns:sdpng="http://www.iana.org/sdpng"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         elementFormDefault="qualified"
         attributeFormDefault="unqualified">

  <xsd:import namespace="http://www.iana.org/sdpng"
           schemaLocation="sdpng.xsd"/>

  <xsd:complexType name="PayloadType" mixed="false">
    <xsd:annotation>
      <xsd:documentation>
     PayloadType, the element for payload type definitions is
     derived from "sdpng:Definition". Inside an element of this
     type, more definitions may be given (derived from
     sdpng:Definition themselves). If no definition is given in the
     content, a definition may be referenced using the "format
     attribute".
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent mixed="false">
      <xsd:extension base="sdpng:Definition">
     <xsd:sequence>
       <xsd:element ref="sdpng:d" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
     <xsd:attribute name="pt" type="xsd:unsignedByte"/>
     <xsd:attribute name="format" type="xsd:string">
       <!-- IDREF? Issue: unique names for definitions!-->
     </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="pt" type="rtp:PayloadType" substitutionGroup="sdpng:d"/
>

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

  <xsd:element name="session" type="rtp:Session"
substitutionGroup="sdpng:sc"/>

  <xsd:complexType name="Session" mixed="false">
```

```
        <xsd:complexContent mixed="false">
          <xsd:extension base="sdpng:SessionConfig">
```

```
        <xsd:sequence>
          <xsd:element name="transport" type="rtp:Transport"/>
        </xsd:sequence>
        <xsd:attribute name="format" type="xsd:string"/>
         </xsd:extension>
       </xsd:complexContent>
     </xsd:complexType>

     <xsd:complexType name="Transport" abstract="true" mixed="false">
       <xsd:complexContent>
         <xsd:extension base="sdpng:Definition">
        <xsd:attribute name="role" type="xsd:string"/>
        <xsd:attribute name="endpoint" type="xsd:string"/>
        <xsd:attribute name="rtp-port" type="xsd:unsignedShort" use="optional"/
>
        <xsd:attribute name="rtcp-port" type="xsd:unsignedShort"
use="optional"/>
         </xsd:extension>
       </xsd:complexContent>
     </xsd:complexType>

     <xsd:simpleType name="IPAddr">
       <xsd:restriction base="xsd:string"/>
     </xsd:simpleType>

     <xsd:simpleType name="IP4Addr">
       <xsd:restriction base="rtp:IPAddr"/>
     </xsd:simpleType>

     <xsd:simpleType name="IP6Addr">
       <xsd:restriction base="rtp:IPAddr"/>
     </xsd:simpleType>

     <xsd:complexType name="UDP" mixed="false">
       <xsd:complexContent mixed="false">
         <xsd:extension base="rtp:Transport">
        <xsd:choice>
          <xsd:element name="option">
            <!-- define options -->
          </xsd:element>
        </xsd:choice>
        <xsd:attribute name="addr" type="rtp:IP4Addr"/>
         </xsd:extension>
       </xsd:complexContent>
     </xsd:complexType>

     <xsd:element name="udp" type="rtp:UDP"/>
```

```
</xsd:schema>
```

5.8 Issues

- Libraries provide partially specified definitions, i.e. without transport parameters. How can SDPng documents reference the definitions and augment them with specific transport parameters?

- Referencing extension profiles: XML-Schema does not support the declaration of multiple schemas via the schemaLocation attribute. Conceivable solution: When extension profiles are used, the SDPng description is a "multi-part" object, that consists of an integrating schema definition (that references all necessary profiles and the base definition) and the actual description document that is a schema instance of the integrating schema.

- Uniqueness of attribute values: When libraries are used they will contain definition elements with "name" attributes for later referencing. How to avoid name clashes for those identifiers? When an SDPng document uses libraries from different sources they could be incompatible because of name collisions. Possible solution: Prefix such IDs with a namespace name (either explicitly or implicitly by interpreting applications). The explicit prefixes have the advantage that no special knowledge would be required to resolve links at the cost of very long ID values.

6. Use of SDPng in conjunction with other IETF Signaling Protocols

The SDPng model provides the notion of Components to indicate the intended types of collaboration between the users in e.g. a teleconferencing scenario.

Three different abstractions are defined that are used for describing the properties of a specific Component:

- a Capability refers to the fact that one of the involved parties supports one particular way of exchanging media -- defined in terms of transport, codec, and other parameters -- as part of the media session.

- a Potential Configuration denotes a set of matching Capabilities from all those involved parties required to successfully realize one particular Component.

- an Actual Configuration indicates the Potential Configuration which was chosen by the involved parties to realize a certain Component at one particular point in time.

As mentioned before, this abstract notion of the interactions between a number of communicating systems needs to be mapped to the application scenarios of SDPng in conjunction with the various IETF signaling protocols: SAP, SIP, RTSP, and MEGACO.

In general, this section provides recommendations and possible scenarios for the use of SDPng within specific protocols and applications. Is does not specify normative requirements.

6.1 The Session Announcement Protocol (SAP)

SAP is used to disseminate a previously created (and typically fixed) session description to a potentially large audience. An interested member of the audience will use the SDPng description contained in SAP to join the announced media sessions.

This means that a SAP announcement contains the Actual Configurations of all Components that are part of the overall teleconference or broadcast.

A SAP announcement may contain multiple Actual Configurations for the same Component. In this case, the "same" (i.e. semantically equivalent) media data from one configuration must be available from each of the Actual Configurations. In practice, this limits the use of multiple Actual Configurations to single-source multicast or broadcast scenarios.

Each receiver of a SAP announcement with SDPng compares its locally stored Capabilities to realize a certain Component against the Actual Configurations contained in the announcement. If the intersection yields one or more Potential Configurations for the receiver, it chooses the one it sees fit best. If the intersection is empty, the receiver cannot participate in the announced session.

SAP may be substituted by HTTP (in the general case, at least), SMTP, NNTP, or other IETF protocols suitable for conveying a media description from one entity to one or more other without the intend for further negotiation of the session parameters.

Example from the SAP spec. to be provided.

6.2 Session Initiation Protocol (SIP)

SIP is used to establish and modify multimedia sessions, and SDPng may be carried at least in SIP INVITE and ACK messages as well as in a number of responses. From dealing with legacy SDP (and its essential non-suitability for capability negotiation), a particular use and interpretation of SDP has been defined for SIP.

One of the important flexibilities introduced by SIP's usage of SDP is that a sender can change dynamically between all codecs that a receiver has indicated support (and has provided an address) for. Codec changes are not signaled out-of-band but only indicated by the payload type within the media stream. From this arises one important consequence to the conceptual view of a Component within SDPng.

There is no clear distinction between Potential and Actual Configurations. There need not be a single Actual Configuration be chosen at setup time within the SIP signaling. Instead, a number of Potential Configurations is signaled in SIP (with all transport parameters required for carrying media streams) and the Actual Configuration is only identified by the payload type which is actually being transmitted at any point in time.

Note that since SDPng does not explicitly distinguish between Potential and Actual Configurations, this has no implications on the SDPng signaling itself.

SIP relies on an "offer/answer" model for the exchange of capability and configuration information. Either the caller or the callee sends an initial session description that is processed by the other side and returned. For capability negotiation, this means that the negotiation follows a two-stage-process: The "offerer" sends its capability description to the receiver. The receiver processes the offerers capabilities and his own capabilities and generates a result

capability description that is sent back to the offerer. Both sides now know the commonly supported configurations and can initiate the media sessions.

Because of this strict "offer/answer" model, the offerer must already send complete configurations (i.e. include transport addresses) along with the capability descriptions. The answer must also contain complete configuration parameters. The following figure shows, how SDPng content can be used in an INVITE request with a correspong 200 OK message.

Simple description document with only one alternative:

```
   F1 INVITE A -> B

   INVITE sip:B@example.com SIP/2.0
   Via: SIP/2.0/UDP hostA.example.com:5060
   From: A <sip:A@example.com>
   To: B <sip:B@example.com>
   Call-ID: 1234@hostA.example.com
   CSeq: 1 INVITE
   Contact: <sip:UserA@192.168.1.1>
   Content-Type: application/sdpng
   Content-Length: 685

<def>
 <audio:codec name="audio-basic" encoding="PCMU"
              sampling="8000" channels="1"/>

 <rtp:pt name="rtp-avp-0" pt="0" format="audio-basic"/>
</def>

<cfg>
  <component name="interactive-audio" media="audio">
    <alt name="AVP-audio-0">
      <rtp:session format="rtp-avp-0">
       <rtp:udp role="receive" endpoint="A" addr="192.168.1.1"
             rtp-port="7800"/>
      </rtp:session>
    </alt>
   </component>
</cfg>

<conf>
 <owner user="A@example.com" id="98765432" version="1" nettype="IN"
                      addrtype="IP4" addr="192.168.1.1"/>
 <session name="SDPng questions">
 </session>
```

```
    <info name="interactive-audio" function="voice">
     Telephony media stream
    <info>
   </conf>

   ================================================

      F2 (100 Trying) B -> A

      SIP/2.0 100 Trying
      Via: SIP/2.0/UDP hostA.example.com:5060
      From: A <sip:A@example.com>
      To: B <sip:B@example.com>
      Call-ID: 1234@hostA.example.com
      CSeq: 1 INVITE
      Content-Length: 0

   ================================================

      F3 180 Ringing B -> A

      SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP hostA.example.com:5060
      From: A <sip:A@example.com>
      To: B <sip:B@example.com>;tag=987654
      Call-ID: 1234@hostA.example.com
      CSeq: 1 INVITE
      Content-Length: 0

   ================================================

      F4 200 OK B -> A

      SIP/2.0 200 OK
      Via: SIP/2.0/UDP hostA.example.com:5060
      From: A <sip:A@example.com>
      To: B <sip:B@example.com>;tag=987654
      Call-ID: 1234@hostA.example.com
      CSeq: 1 INVITE
      Contact: <sip:B@192.168.1.2>
      Content-Type: application/sdpng
      Content-Length: 479

   <def>
    <audio:codec name="audio-basic" encoding="PCMU"
                 sampling="8000" channels="1"/>

    <rtp:pt name="rtp-avp-0" pt="0" format="audio-basic"/>
```

```
   </def>

   <cfg>
     <component name="interactive-audio" media="audio">
       <alt name="AVP-audio-0">
         <rtp:session format="rtp-avp-0">
          <rtp:udp role="receive" endpoint="A" addr="192.168.1.1"
                rtp-port="7800"/>
          <rtp:udp role="receive" endpoint="B" addr="192.168.1.2"
                rtp-port="9410"/>
         </rtp:session>
       </alt>
      </component>
   </cfg>
   ================================================

   ACK from A to B omitted
```

In the INVITE message, A sends B a description document, that specifies exactly one component with one alternative (the PCMU audio stream). All required transport parameters all already contained in the description. The rtp:udp element provides an attribute "role" with a value of "receive", indicating that the specified endpoint address is used by the endpoint to receive media data. The element also provides the attribute "endpoint" with a value of "A", denominating the endpoint that can receive data on the specified address. This means, the semantics of specified transport addresses in configuration descriptions are the same as for SDP (when used with SIP): An endpoint specifies where it wants to receive data.

In the 200 OK message, B sends an updated description document to A. For the sake of conciseness, the conf element (containing meta information about the conference) has been omitted. B supports the payload format that A has offered and adds his own transport parameters to the configuration information, specifying the endpoint address where B wants to receive media data. In order to disambiguate its transport configurations from A's, B sets the attribute "endpoint" to the value "B". The specific value of the "endpoint" attribute is not important, the only requirements are that a party that contributes to the session description, must use a unique name for the endpoint attribute and that a contributing party must use the same value for the endpoint attributes of all elements

it adds to the session description.

The following example shows a capability description that provides two alternatives for the audio component.

Description document with two alternatives:

F1 INVITE A -> B

```
INVITE sip:B@example.com SIP/2.0
Via: SIP/2.0/UDP hostA.example.com:5060
From: A <sip:A@example.com>
To: B <sip:B@example.com>
Call-ID: 1234@hostA.example.com
CSeq: 1 INVITE
Contact: <sip:UserA@192.168.1.1>
Content-Type: application/sdpng
Content-Length: 935

<def>
 <audio:codec name="audio-basic" encoding="PCMU"
              sampling="8000" channels="1"/>

 <audio:codec name="g729" encoding="G729" channels="1" sampling="8000"/>

 <rtp:pt name="rtp-avp-0" pt="0" format="audio-basic"/>
 <rtp:pt name="rtp-avp-18" pt="18" format="g729"/>

 <rtp:udp name="A-rcv" role="receive" endpoint="A" addr="192.168.1.1"
       rtp-port="7800"/>
</def>

<cfg>
  <component name="interactive-audio" media="audio">
    <alt name="AVP-audio-0">
      <rtp:session format="rtp-avp-0" transport="A-rcv""/>
    </alt>
    <alt name="AVP-audio-18">
      <rtp:session format="rtp-avp-18" transport="A-rcv"/>
    </alt>
   </component>
</cfg>

<conf>
 <owner user="A@example.com" id="98765432" version="1" nettype="IN"
                      addrtype="IP4" addr="192.168.1.1"/>
 <session name="SDPng questions">
 </session>
```

```
    <info name="interactive-audio" function="voice">
     Telephony media stream
    <info>
   </conf>

   ================================================

      F2 (100 Trying) B -> A

      SIP/2.0 100 Trying
      Via: SIP/2.0/UDP hostA.example.com:5060
      From: A <sip:A@example.com>
      To: B <sip:B@example.com>
      Call-ID: 1234@hostA.example.com
      CSeq: 1 INVITE
      Content-Length: 0

   ================================================

      F3 180 Ringing B -> A

      SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP hostA.example.com:5060
      From: A <sip:A@example.com>
      To: B <sip:B@example.com>;tag=987654
      Call-ID: 1234@hostA.example.com
      CSeq: 1 INVITE
      Content-Length: 0

   ================================================

      F4 200 OK B -> A

      SIP/2.0 200 OK
      Via: SIP/2.0/UDP hostA.example.com:5060
      From: A <sip:A@example.com>
      To: B <sip:B@example.com>;tag=987654
      Call-ID: 1234@hostA.example.com
      CSeq: 1 INVITE
      Contact: <sip:B@192.168.1.2>
      Content-Type: application/sdpng
      Content-Length: 479

   <def>
    <audio:codec name="audio-basic" encoding="PCMU"
                 sampling="8000" channels="1"/>

    <audio:codec name="g729" encoding="G729" channels="1" sampling="8000"/>
```

```
    <rtp:pt name="rtp-avp-0" pt="0" format="audio-basic"/>
    <rtp:pt name="rtp-avp-18" pt="18" format="g729"/>

    <rtp:udp name="A-rcv" role="receive" endpoint="A" addr="192.168.1.1"
          rtp-port="7800"/>

    <rtp:udp name="B-rcv" role="receive" endpoint="B" addr="192.168.1.2"
          rtp-port="9410"/>
   </def>

   <cfg>
     <component name="interactive-audio" media="audio">
       <alt name="AVP-audio-0">
         <rtp:session format="rtp-avp-0" transport="A-rcv B-rcv"/>
       </alt>
      </component>
   </cfg>
   ================================================

   ACK from A to B omitted
```

In the INVITE message, A sends B a description document, that specifies one component with two alternatives for the audio stream (PCMU and G.729). Since A wants to use the same transport address for receiving media data regardless of the payload format, A provides the transport specification in the def element and references this definition in the rtp:session elements for both alternatives by using the attribute "transport".

In the 200 OK message, B sends an updated description document to A. B does only support PCMU, so it removes the alternative for G.729 from the description. B also defines its transport address in the def element and references this definition by adding "B-rcv" to the attribute "transport" of the rtp:session element. (B could also have used the rtp:udp element inside the rtp:session element, but this example intends to demonstrate how to reference multiple transport definitions by using the attribute "transport").

6.3 Real-Time Streaming Protocol (RTSP)

In contrast to SIP, RTSP has, from its intended usage, a clear distinction between offering Potential Configurations (typically by

the server) and choosing one out of these (by the client), and, in some cases; some parameters (such as multicast addresses) may be dictated by the server. Hence with RTSP, there is a clear distinguish between Potential Configurations during the negotiation phase and a finally chosen Actual Configuration according to which streaming will take place.

Example from the RTSP spec to be provided.

6.4 Media Gateway Control Protocol (MEGACOP)

The MEGACO architecture also follows the SDPng model of a clear separation between Potential and Actual Configurations. Upon startup, a Media Gateway (MG) will "register" with its Media Gateway Controller (MGC) and the latter will audit the MG for its Capabilities. Those will be provided as Potential Configurations, possibly with extensive Constraints specifications. Whenever a media path needs to be set up by the MGC between two MGs or an MG needs to be reconfigured internally, the MGC will use (updated) Actual Configurations.

Details and examples to be defined.

7. Open Issues

Definition of baseline libraries

A registry (reuse of SDP mechanisms and names etc.) needs to be set up.

Negotiation mechanisms for multiparty conferencing need to be formalized.

Mapping to other media description formats (SDP, H.245, ...) should be provided. For H.245, this is probably a different document (belonging to the SIP-H.323 inter-working group).

Implicit declaration of SDPng schema and default profiles

References

[1] Kutscher, D., Ott, J., Bormann, C. and I. Curcio, "Requirements for Session Description and Capability Negotiation", Internet Draft draft-ietf-mmusic-sdpng-req-01.txt, April 2001.

[2] Handley, M. and V. Jacobsen, "SDP: Session Description Protocol", RFC 2327, April 1998.

[3] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobsen, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[4] Schulzrinne, H., "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, January 1996.

[5] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", draft-ietf-avt-profile-new-10.txt (work in progress), March 2001.

[6] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J., Vega-Garcia, A. and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, September 1997.

[7] Rosenberg, J. and H. Schulzrinne, "An RTP Payload Format for Generic Forward Error Correction", RFC 2733, December 1999.

[8] Perkins, C. and O. Hodson, "Options for Repair of Streaming Media", RFC 2354, June 1998.

[9] Handley, M., Perkins, C. and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[10] World Wide Web Consortium (W3C), "Extensible Markup Language (XML) 1.0 (Second Edition)", Status W3C Recommendation, Version http://www.w3.org/TR/2000/REC-xml-20001006, October 2000.

[11] World Wide Web Consortium (W3C), "Namespaces in XML", Status W3C Recommendation, Version http://www.w3.org/TR/1999/REC-xml-names-19990114, January 1999.

[12] World Wide Web Consortium (W3C), "XML Inclusions (XInclude) Version 1.0", Status W3C Working Draft, Version http://www.w3.org/TR/2001/WD-xinclude-20010516, May 2001.

[13] World Wide Web Consortium (W3C), "XML Schema Part 1: Structures", Version http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/, Status W3C Recommendation, May 2001.

[14] World Wide Web Consortium (W3C), "XML Schema Part 2: Datatypes", Version http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/, Status W3C Recommendation, May 2001.

[15] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with SDP", draft-ietf-mmusic-sdp-offer-answer-01.txt (work in progress), February 2002.

Authors' Addresses

Dirk Kutscher
TZI, Universitaet Bremen
Bibliothekstr. 1
Bremen 28359
Germany

Phone: +49.421.218-7595, sip:dku@tzi.org
Fax: +49.421.218-7000
EMail: dku@tzi.uni-bremen.de

Joerg Ott
TZI, Universitaet Bremen
Bibliothekstr. 1
Bremen 28359
Germany

Phone: +49.421.201-7028, sip:jo@tzi.org
Fax: +49.421.218-7000
EMail: jo@tzi.uni-bremen.de

Carsten Bormann
TZI, Universitaet Bremen
Bibliothekstr. 1
Bremen 28359
Germany

Phone: +49.421.218-7024, sip:cabo@tzi.org
Fax: +49.421.218-7000
EMail: cabo@tzi.org

Appendix A. Base SDPng Specifications for Audio Codec Descriptions

[5] specifies a number of audio codecs including short name to be used as reference by session description protocols such as SDP and SDPng. Those codec names, as listed in the first column of the above table, are used to identify codecs in SDPng.

The following sections indicate the default values that are assumed if nothing else than the codec reference is specified.

The following audio:codec attributes are defined for audio codecs:

name: the identifier to be later used for referencing the codec spec

encoding: the RTP/AVP profile identifier as registered with IANA

mime: the MIME type; may alternatively be specified instead of "encoding"

channels: the number of independent media channels

pattern: the media channel pattern for mapping channels to payload

sampling: the sample rate for the codec (which in most cases equals the RTP clock)

Furthermore, options may be defined of the following format:

<option id="name">value</option>

if a value is associated with the option (note that arbitrary complex values are allowed), or alternatively:

<option id="name"/>

if the option is just a boolean indicator.

Attributes for the "option" tag are the following:

id: the identifier for the option (variable name)

collaps: the collapsing rules for this optional element, defined as follows:

min: for numeric values only

max: for numeric values only

x: intersection of enumerated values, value lists

A.1 DVI4

```
<audio:codec name="dvi4" encoding="DVI4" channels="1" sampling="8000">

<rtp:pt name="rtp-avp-5" pt="5" format="dvi4"/>
<rtp:pt name="rtp-avp-6" pt="6">
    <audio:codec encoding="DVI4" channels="1" sampling="16000">
</rtp:pt>
```

Note that there is no default sampling rate specified for DVI4 and hence a sampling rate MUST be specified.

A.2 G.722

```
<audio:codec name="g722" encoding="G722" channels="1" sampling="16000"/>
<rtp:pt name="rtp-avp-9" pt="9" format="g722"/>
```

Note as per [5] that the RTP clock rate is 8000Hz rather than 16000 Hz.

A.3 G.726

```
<audio:codec name="g726-40" encoding="G726-40" channels="1" sampling="8000"/
>
<audio:codec name="g726-32" encoding="G726-32" channels="1" sampling="8000"/
>
<audio:codec name="g726-24" encoding="G726-24" channels="1" sampling="8000"/
>
<audio:codec name="g726-16" encoding="G726-16" channels="1" sampling="8000"/
>

<rtp:pt name="rtp-avp-5" pt="5" format="g726-32"/>
```

A.4 G.728

```
<audio:codec name="g728" encoding="G728" channels="1" sampling="8000"/>
<rtp:pt name="rtp-avp-15" pt="15" format="g728"/>
```

A.5 G.729

```
G.729 Annex A: reduced complexity of G.729
G.729 Annex B: comfort noise

<audio:codec name="g729" encoding="G729" channels="1" sampling="8000"/>
<rtp:pt name="rtp-avp-18" pt="18" format="g729"/>
```

For further codec description, the following options (which carry no

values associated with them) MAY be included:

```
<option id="annexA"/>
<!-- to indicate the use of Annex A reduced complexity -->

<option id="annexB"/>
<!-- to indicate the use of Annex B comfort noise -->
```

As stated in [5], the use of these options can be detected within the media stream.

A.6 G.729 Annex D and E

```
<audio:codec name="g729d" encoding="G729D" channels="1" sampling="8000"/>
<audio:codec name="g729e" encoding="G729E" channels="1" sampling="8000"/>
```

The following option MAY be used with both Annexes D and E:

```
<option id="annexB"/>
<!-- to indicate the use of Annex B comfort noise -->
```

A.7 GSM

A.7.1 GSM Full Rate

The GSM Full Rate codec is indicated as follows:

```
<audio:codec name="gsm" encoding="GSM" channels="1" sampling="8000"/>
<rtp:pt name="rtp-avp-3" pt="3" format="gsm"/>
```

A.7.2 GSM Half Rate

The GSM Half Rate codec is indicated as follows:

```
<audio:codec name="gsm-hr" encoding="GSM-HR" channels="1" sampling="8000"/>
```

A.7.3 GSM Enhanced Full Rate

The GSM Enhanced Full Rate codec is indicated as follows:

```
<audio:codec name="gsm-efr" encoding="GSM-EFR" channels="1" sampling="8000"/
>
```

A.8 L8

```
<audio:codec name="l8" encoding="L8" channels="1" sampling="8000"/>
```

A.9 L16

```
<audio:codec name="l16" encoding="L16" channels="1" sampling="8000"/>

<rtp:pt name="rtp-avp-11" pt="11" format="gsm"/>
<rtp:pt name="rtp-avp-10" pt="11" format="gsm">
  <audio:codec encoding="L16" channels="2" sampling="8000"/>
</rtp:pt>
```

A.10 LPC

```
<audio:codec name="lpc" encoding="LPC" channels="1" sampling="8000"/>
```

A.11 MPA

```
<audio:codec name="mpa" encoding="MPA" channels="1" sampling="8000"/>
<rtp:pt name="rtp-avp-14" pt="14" format="mpa"/>
```

A.12 PCMA and PCMU

```
<audio:codec name="pcmu" encoding="PCMU" channels="1" sampling="8000"/>
<audio:codec name="pcma" encoding="PCMA" channels="1" sampling="8000"/>

<rtp:pt name="rtp-avp-0" pt="0" format="pcmu"/>
<rtp:pt name="rtp-avp-8" pt="8" format="pcma"/>
```

A.13 QCELP

```
<audio:codec name="qcelp" encoding="QCELP" channels="1" sampling="8000"/>
<rtp:pt name="rtp-avp-12" pt="12" format="qcelp"/>
```

A.14 VDVI

```
<audio:codec name="vdvi" encoding="VDVI" channels="1" sampling="8000"/>
```

Appendix B. SDPng Library for Audio Codec Definitions

This section contains an SDPng library with the audio codec definitions from Appendix A.

```
<def xmlns="http://www.iana.org/sdpng"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.iana.org/sdpng/int integrated.xsd"
      xmlns:audio="http://www.iana.org/sdpng/audio"
      xmlns:rtp="http://www.iana.org/sdpng/rtp">

    <audio:codec name="dvi4" encoding="DVI4" channels="1" sampling="8000"/>
    <audio:codec name="g722" encoding="G722" channels="1" sampling="16000"/>
    <audio:codec name="g726-40" encoding="G726-40" channels="1"
sampling="8000"/>
    <audio:codec name="g726-32" encoding="G726-32" channels="1"
sampling="8000"/>
    <audio:codec name="g726-24" encoding="G726-24" channels="1"
sampling="8000"/>
    <audio:codec name="g726-16" encoding="G726-16" channels="1"
sampling="8000"/>
    <audio:codec name="g728" encoding="G728" channels="1" sampling="8000"/>
    <audio:codec name="g729" encoding="G729" channels="1" sampling="8000"/>
    <audio:codec name="g729d" encoding="G729D" channels="1" sampling="8000"/
>
    <audio:codec name="g729e" encoding="G729E" channels="1" sampling="8000"/
>
    <audio:codec name="gsm" encoding="GSM" channels="1" sampling="8000"/>
    <audio:codec name="gsm-hr" encoding="GSM-HR" channels="1"
sampling="8000"/>
    <audio:codec name="gsm-efr" encoding="GSM-EFR" channels="1"
sampling="8000"/>
    <audio:codec name="l8" encoding="L8" channels="1" sampling="8000"/>
    <audio:codec name="l16" encoding="L16" channels="1" sampling="8000"/>
    <audio:codec name="lpc" encoding="LPC" channels="1" sampling="8000"/>
    <audio:codec name="mpa" encoding="MPA" channels="1" sampling="8000"/>
    <audio:codec name="pcmu" encoding="PCMU" channels="1" sampling="8000"/>
    <audio:codec name="pcma" encoding="PCMA" channels="1" sampling="8000"/>
    <audio:codec name="qcelp" encoding="QCELP" channels="1" sampling="8000"/
>
    <audio:codec name="vdvi" encoding="VDVI" channels="1" sampling="8000"/>

</def>
```

Appendix C. SDPng Library for RTP Payload Format Definitions

This section contains an SDPng library with the RTP payload format definitions from Appendix A.

```
<def xmlns="http://www.iana.org/sdpng"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.iana.org/sdpng/int integrated.xsd"
      xmlns:audio="http://www.iana.org/sdpng/audio"
      xmlns:rtp="http://www.iana.org/sdpng/rtp"
      xmlns:xi="http://www.w3.org/2001/XInclude">

<!-- import audio codec definitions here: -->

    <xi:xinclude href="http://www.iana.org/sdpng/audio/audio.sdpng"
parse="xml"/>

    <rtp:pt name="rtp-avp-5" pt="5" format="dvi4"/>

    <rtp:pt name="rtp-avp-6" pt="6">
      <audio:codec encoding="DVI4" channels="1" sampling="16000">
    </rtp:pt>

    <rtp:pt name="rtp-avp-9" pt="9" format="g722"/>

    <rtp:pt name="rtp-avp-5" pt="5" format="g726-32"/>

    <rtp:pt name="rtp-avp-15" pt="15" format="g728"/>

    <rtp:pt name="rtp-avp-18" pt="18" format="g729"/>

    <rtp:pt name="rtp-avp-3" pt="3" format="gsm"/>

    <rtp:pt name="rtp-avp-11" pt="11" format="gsm"/>

    <rtp:pt name="rtp-avp-10" pt="11" format="gsm">
      <audio:codec encoding="L16" channels="2" sampling="8000"/>
    </rtp:pt>

    <rtp:pt name="rtp-avp-14" pt="14" format="mpa"/>

    <rtp:pt name="rtp-avp-0" pt="0" format="pcmu"/>

    <rtp:pt name="rtp-avp-8" pt="8" format="pcma"/>

    <rtp:pt name="rtp-avp-12" pt="12" format="qcelp"/>

</def>
```

Appendix D. Change History

draft-ietf-mmusic-sdpng-04.txt

* New section on capability negotiation (Section 4).
* New section on referencing definitions (Section 3.3).
* New section on properties (Section 3.3.2).
* New section on definition groups (Section 3.3.3).

draft-ietf-mmusic-sdpng-03.txt

* Extension of the SDPng schema (use of Xlinks etc.)
* Clarification in the text
* Fixed examples
* Added example libraries as appendices
* More details on usage with SIP, including examples.

draft-ietf-mmusic-sdpng-02.txt

* Added a section on formal specification mechanisms (Section 5).

draft-ietf-mmusic-sdpng-01.txt

* renamed section "Syntax Proposal" to "Syntax Definition Mechanisms". More text on DTD vs. schema. Edited the example description.
* updated example definitions in section "Definitions" and "Components & Configurations"
* section "Session Attributes" replaces section "Session"
* new appendix on audio codec definitions

Full Copyright Statement

Acknowledgement

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