**PROPOSED changes**

<table>
<thead>
<tr>
<th>Change</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>Add Message Sequence Chart (MSC) and/or example code snippet of the method usage: SipClientConnection.setCredentials(). page 23</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>Should say clearly that the SipClientConnection returned from the SipDialog.getNewClientConnection() is in <strong>Initialized</strong> state. page 35</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>Correct SipDialog SUBSCRIBE code example. The opened SipClientConnection is not in shared mode. Should set both From and Contact header for the initial SUBSCRIBE request. Also correct the line scc.setHeader(&quot;Accept&quot;, &quot;application/xpidf+xml&quot;); to be scc.setHeader(&quot;Accept&quot;, &quot;application/pidf+xml&quot;); page 34</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>The SipDialog state diagram indicates that the SipDialog instance is already created when INVITE/SUBSCRIBE is sent. That is not the case; it is not necessary (for API implementation) to create the SipDialog until the provisional 101-199 response is received. The state diagram is also wrong for the SUBSCRIBE case because there you do not get provisional response at all. It is directly 2xx or NOTIFY, that creates the dialog. Proposed to add a new state <em>Initialized</em> which in practise is never visible to the user, but acts as a common starting point for both INVITE/SUBSCRIBE cases. The SipDialog can be fetched earliest in the <em>Early</em> state. In order to help reading draw separate SipDialog state diagrams for both client and server sides. page 33</td>
</tr>
</tbody>
</table>
SipClientConnection.receive() cannot be called after sending ACK. This prevents receiving 200 OKs from multiple end-points (forking case) and multiple re-sent 200 OKs from the same end-point.

Correct state diagram and the rules for receiving multiple 2xx responses. Enable calling receive() in Completed state.

Furthermore, it should be clarified how responses from multiple endpoints are treated in the SipClientConnection. Essentially SipClientConnection associates with latest response fetched with receive() method. Also the dialog will be always associated to the latest response received.

page 16 & 22

Unclear how the multiple 200 OK responses are handled with the API.

Error response to INVITE:

When error response for INVITE is received the client transaction sends automatically ACK. Now on the server side this ACK is handled in the same server transaction that received original INVITE, but the ACK is not given up to the TU (as a new SipServerConnection).

200 OK response to INVITE:

Should be able to resend 200 OK if the ACK has not been received. Correct SipServerConnection state diagram to enable calling send() for 2xx responses in Completed state.

Generally if sending response fails IOException is thrown.

These should be clarified in the SipServerConnection section.

page 24

Clarity why ACK is (not) visible to the application in error response case.

In the case where 200 OK has been sent for INVITE, but the ACK has not been received it should be possible to resend the 200 OK.
### 7. Rewrite server connection initialization rules.

*SipConnectionNotifier* can be opened with the following URI:

```
sip:[nnnn][;type="application/x-game"]
```

- **sip:** or **sips:** - protocol scheme without address to indicate server mode
- **nnnn** - listening port number (optional)
- **type** - URI parameter specifying application identifier (optional)

- Port number **nnnn** specifies the listening port. The **IOException** is thrown if the port number is already in use or it cannot be opened for other reason. If the port number is given it always indicates dedicated port number for the application (not shared SIP identity mode).
- If only "**sip:**" scheme is given the system is selecting the port number, otherwise the case is similar as if the number was given by the application.
- If application identifier is given with parameter 'type' the system listening port and the SIP identity is shared, with other applications (shared SIP identity mode).

---

### 8. More specific definition how incoming request messages are dispatched to applications when using shared mode server connection.

- **page 9 & 10**

### 9. Correct interface

*SipServerConnectionListener,* the parameter **ssc** in **notifyRequest()** should be named **scn**.

- **page 32**

### 10. Should state clearly that the **SipConnection.send()** method is asynchronous. Any kind of immediate failure should throw **IOException**.

- **The InterruptedIOException should be removed from SipConnection.send().**

- **page 12**

### 11. Create PRACK from the **SipDialog**. At the moment the **SipDialog** does not allow that in the state diagram specification. It should be generally possible to call **getNewClientConnection()** in *Early state*

- **page 33**

---

**Clarify** *SipConnectionNotifier* opening method to have more exact rules. It is assumed that applications using JSR180 would mostly use the shared identity case (where port number is omitted).

**Clarify how the SIP dispatching should be done in the JSR180 implementations.**

This is just to avoid confusion. The type of the **notifyRequest()** parameter is **SipConnectionNotifier** not **SipServerConnection**.

Now for example if the network is not available calling **send()** could throw **IOException** immediately.

The description of **InterruptedIOException** is wrong for **send()**, since it does not timeout. Furthermore, this exception does not carry any extra information to the **IOException**. Also this exception is used with **InputStream/OutputStream** based classes.

Support for RFC3262, Reliability of Provisional Responses in the Session Initiation Protocol (SIP)
<table>
<thead>
<tr>
<th>No.</th>
<th>Original Text</th>
<th>Clarification</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Remove the <code>InterruptedIOException</code> from <code>SipConnectionNotifier.acceptAndOpen()</code></td>
<td>The <code>InterruptedIOException</code> is used with the <code>InputStream</code> and <code>OutputStream</code> based classes and does not give any additional information here. If the <code>SipConnectionNotifier</code> is closed for some reason the <code>IOException</code> should be thrown.</td>
</tr>
<tr>
<td>13</td>
<td>Giving <code>SipConnectionNotifier</code> as a parameter for <code>SipClientConnection.initRequest(…)</code></td>
<td>Clarify the rules for <code>SipClientConnection.initRequest(…)</code>, when <code>SipConnectionNotifier</code> is given as a parameter. The <code>SipConnectionNotifier</code> can be in either shared or not shared SIP identity mode. See also the item 7.</td>
</tr>
</tbody>
</table>

**ACCEPTED changes**

None

**DEFERRED changes**

None