Subject: Re: SEMS DTMF
From: Stefan Sayer <stefan.sayer@googlemail.com>
Date: Tue, 24 Aug 2010 01:54:37 +0200
To: "David J." <david@styleflare.com>
CC: sems mailing list <sems@lists.iptel.org>

Hi,

ok, time to continue here a little bit.

Stefan Sayer wrote:
> Hi,
>
> it's raining, so seems like its good time for a second part of this small DSM tutorial.
>
> Stefan Sayer wrote:
> > David J. wrote:
> > > Hi Stefan,
> > >
> > > I am trying to accomplish a script that parallel forks to many callers and then the one who enters the right DTMF code gets the call.
> > >
> > > In order to do this, I am guessing, I need b2b_mode which sends RINGING back to the Caller, does the forking plays a WAV/MP3 announcement to the Callee's and some sort of DTMF detection that captures the dialed digits and verify's them via script, database or whatever.
> >
> > actually, in order to be able to prompt the callees and collect the DTMF from them, you will have to establish separate calls to them - so that b2b mode is not really suitable. Once the right callee leg is identified, though, you can connect audio from the caller and the callee leg by joining the same conference room. You can interact between the two legs (e.g. when one hangs up) by sending events back and forth, the only thing you need is to know in both legs is the id of the other leg (local-tag).
> >
> > But, to start from the beginning. We will need two DSM scripts, one for the caller, one for the callee leg. We call that application quizconnect, and we tell dsm to load DSM application configurations by setting in dsm.conf:
> > ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> > conf_dir=*/usr/local/etc/sems/dsm/*
> > ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> >
> > Then we can create /usr/local/etc/sems/dsm/quizconnect.conf which loads the scripts and sets the settings for the quizconnect application:
> > /usr/local/etc/sems/dsm/quizconnect.conf:
> > ----------------------------------
> > diag_path=*/usr/local/lib/sems/dsm/quizconnect/*
> > load_diags=quizconnect_caller,quizconnect_callee
> > register_apps=quizconnect_caller,quizconnect_callee
> > mod_path=*/usr/local/lib/sems/dsm/*
> > preload_mods=mod_mysql
> > run_invite_event=yes
> > set_param_variables=yes
> > #run_system_dsms=
> >
> > # some configuration parameters
> > # - can be used with e.g. $config.prompt_path
> > prompt_path=*/usr/local/lib/sems/dsm/quizconnect/prompts/*
> > ----------------------------------
> >
> > We have preloaded the mysql module, which needs this to be initialized and read its configuration (which contains the DB connection, for example). We also set run_invite_event=yes in the dsm config, that way we get an 'invite' event into the DSM scripts. Now we create two scripts, /usr/local/lib/sems/dsm/quizconnect/quizconnect_caller.dsm and /usr/local/lib/sems/dsm/quizconnect/quizconnect_callee.dsm .
> >
> > When that invite event comes, we tell dsm to not connect the session (i.e. reply with 200 and connect audio), but to reply 183 (early media) and play a file:
> >
> > /usr/local/lib/sems/dsm/quizconnect/quizconnect_caller.dsm:
> > --
> > import(mod_dlg);
> > initial state START;
> > transition "got INVITE in caller leg" START - invite -> RUN_INVITE;
>
> apparently no-one tried this, because here we obviously have a c&p typo:
>
> - > initial state RUN_INVITE enter {
> + > state RUN_INVITE enter {
>
> we should only have one initial state ('START').
>
> >
> > log(2, "got invite!");
> > set($connect_session=0);
> > -- reply with 183 and parse SDP
> > dlg.acceptInvite(183, Session Progress);
> > -- set input and output of the session (we have $connect_session=0)
> > setInOutPlaylist();
> > -- play some welcome message
> > sets($prompt_name=$(config.prompt_path)/welcome_caller.wav)
> > playFile($prompt_name);
> > };
> > --
> >
> > To run this, in sems.conf set application=quizconnect_caller .
> >
> > so, that's the first part. in the next part, we will see how we can read callee numbers from mysql DB, create some callee legs, and interact between caller and callee legs.
>
> now, first we want to handle the error that the file does not exist or can not be opened. For this, we create a special

transition, an "exception transition". Once an exception is thrown (by some internal function or a module function), the current sequence of statements is interrupted, and only exception transitions are executed; all other transitions are ignored.

quizconnect_caller.dsm:
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "error opening file" RUN_INVITE - exception / {
  dlg.reply(500, Server Internal Error);
  stop(false);
} -> END;

state END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

using 'stop', we stop execution of this session. stop() or stop(true) sends a BYE, which we don't want here, and stop(false) just ends the session after the current event is processed.

Now, a little improvement in the development environment is helpful: when we update the DSM script, we don't want to have to restart SEMS every time. So, what we do is we load the xmlrpc2di module

sems.conf:
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
load_plugins=wav;session_timer;uac_auth;dsm;monitoring;xmlrpc2di
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

and we tell xmlrpc2di to export the functions from dsm module directly

xmlrpc2di.conf:
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
export_di=yes
direct_export=dsm;monitoring
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

then we can in another shell write that few python lines to reload the quizconnect config:
```
$ python
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
 >>> from xmlrpclib import *
 >>> s = ServerProxy('http://localhost:8090')
 >>> s.calls()
0
 >>> s.loadConfig('/usr/local/etc/sems/dsm/quizconnect.conf', 'quizconnect')
[200, 'OK']
```

this way we can replace existing applications (the new set of scripts are executed for new calls only), and also load other new applications into the running server.

So, every time we change the DSM scripts, we can simply run s.loadConfig(...).

The next task should be to get the list of possible destinations (callees) from the database; we will use a mysql DB. For this, we create a table in mysql

```
CREATE TABLE callees (
  id int(10) unsigned NOT NULL auto_increment,
  caller varchar(128) NOT NULL,
  callee varchar(128) NOT NULL,
  pin varchar(32) NOT NULL,
  PRIMARY KEY  (id)
);
```

and insert some rows:
```
insert into callees (caller,callee,pin) values ("35","john","12345");
insert into callees (caller,callee,pin) values ("35","anna","54321");
```

so, if the number '35' will be called, john and anna will be connected, and john should enter 12345, while anna should enter 54321.

In the script we use the mod_mysql module:

quizconnect_caller.dsm:
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
import(mod_mysql);
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

we can set the DB connection in our quizconnect.conf (or pass it to mysql.connect() action):

quizconnect.conf:
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
db_url=mysql://user:pwd@localhost/quizconnect
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

btw, as we have seen above with prompt_path, all configuration keys from our quizconnect.conf are accessible with $config.key, so we could also write mysql.connect($config.db_url).

With the beginning of processing the call, we connect to the database.
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state RUN_INVITE enter {
  log(2, "got invite!");
  set($connect_session=0);
  myslq.connect();
  -- reply with 183 and parse SDP
  dlg.acceptInvite(183, Session Progress);
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

mysql.connect doesn't throw an exception if the connection fails ('Access denied', 'server has gone away'), instead it sets an error code ($errno) which is the old style of reporting errors. To throw an exception in that case, we can use throwOnError() and also handle that exception:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state RUN_INVITE enter {
  log(2, "got invite!");
```

```
  set($connect_session=0);
  mysql.connect();
  throwOnError()
  -- reply with 183 and parse SDP
  dlg.acceptInvite(183, Session Progress);
  -- set input and output of the session (we have $connect_session=0)
  setInOutPlaylist();
  -- play some welcome message
  sets($prompt_name=$(config.prompt_path)/welcome_caller.wav)
  playFile($prompt_name);
};

transition "error opening file" RUN_INVITE - exception; test(#type==file) / {
  log(0, "error opening file!");
  dlg.reply(500, Server Internal Error);
  stop(false);
} -> END;

transition "DB error" RUN_INVITE - exception; test(#type==connection) / {
  log(0, "error connecting to DB!");
  logParams(0);
  dlg.reply(500, Server Internal Error);
  stop(false);
} -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

When an exception is processed, the parameters (#paramname) are those of the exception - thus if we do logParams(0), we can see the actual error from DB in the log.

Now we can select the callees from the database:

```
  set($query_key=@user);
  mysql.query(select callee, pin from callees where caller=$query_key);
```

which should give us $errno and $db.rows.

We will now apply a small trick: We want to process the results of the DB query, and make some transitions depending on whether that worked or not. By doing a "repost()", the current event is evaluated once more. so we can do:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
...
  set($query_key=@user);
  mysql.query(select callee, pin from callees where caller=$query_key);
  repost();
};

transition "query failed" RUN_INVITE - test($errno!="") / {
  log(1, "query failed!");
  logParams(0);
  dlg.reply(500, Server Internal Error);
  stop(false);
} -> END;

transition "no results" RUN_INVITE - test($db.rows==0) / {
  log(3, "no results");
  dlg.reply(404, Not found);
  stop(false);
} -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

to handle query error and empty destination set. If we have some results, we go to a new state CREATE_CALLEE_LEGS:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "we have results" RUN_INVITE - test($db.rows!=0) / set($callee_counter=0) -> CREATE_CALLEE_LEGS;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

We will actually loop a few times into that state, for every row that we get from the database - thus we do repost() every time we enter the state, to make sure we don't stay there (its a 'transitional state'):

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state CREATE_CALLEE_LEGS
 enter {
   repost();
 };

transition "create one more" CREATE_CALLEE_LEGS - test($callee_counter<$db.rows) / {
   -- this will fill $callee, $pin from current row
   mysql.getResult($callee_counter);

   set(b_leg_caller=quizconnect);
   set(b_leg_callee=$callee);
   set(b_leg_domain=sip.domain.net);
   set(b_leg_app=quizconnect_callee);

   -- pass $pin to other leg
   set(b_leg_var.pin=$pin);

   dlg.dialout(b_leg);

   -- if that worked, we have the ID of the other leg in $b_leg_ltag
   log(3, $b_leg_ltag);
```

First, here a maybe not so obvious fix (but one gets it by reading the ERROR logs carefully):
if we are looping several times through this, $b_leg_ltag is still set, so dlg.dialout will try to create another call with the same local tag, which fails (error message is something like it can not be added to session container). so we need to add here:

```
   -- reset for new call
   clear($b_leg_ltag);
```

```
   inc(callee_counter);
} -> CREATE_CALLEE_LEGS;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

We use the dlg.dialout function to create an entirely new call, which will execute the quizconnect_callee application, and
will be from quizconnect to callee@sip.domain.net. We also pass the pin to the other leg, this variable can be accessed as
$pin in the other script.

When all callee legs are created, we go to a new state. Here we also handle the BYE in the caller leg, at least we stop
our own call:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "done creating callee legs" CREATE_CALLEE_LEGS - test($callee_counter==$db.rows) -> WAIT_CALLEE;

state WAIT_CALLEE;

transition "BYE received" WAIT_CALLEE - hangup / stop -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

So, that's for the second part of that tutorial. In the third part, we are hopefully finally going to see how to interact
between the call legs, and how to connect the legs into the same conference. Some hints:
- postEvent() can post events with variables between DSM call legs
- mod_conference is used to join audio of two calls to a conference
```

But now, if we look at in which state where the caller session actually is, we will see that after all that it actually is in the START state. Strange, why does this happen? The reason is that there's one event run for processing the INVITE message, the 'invite' event, and then, for the sessionStart event (which is executed because we do dlg.acceptInvite()) the DSM starts again at the initial state. So we need to add another transition to go to the right place:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "got session start in caller leg" START - sessionStart -> WAIT_CALLEE;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

If we try canceling the call, we see that there's another bugfix needed: "stop(true)" does not work here, because we have not yet really accepted the call (so bye() won't work) - to the CANCEL we should reply "487 Request Terminated":

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "CANCEL received" WAIT_CALLEE - hangup /  {
  dlg.reply(487, Request Terminated);
  stop(false);
} -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

But, when the caller hangs up, we need to tell the callee legs to tear down. So, first we need to save the ltags of the callee legs, (using some variable names trickery) into $b_ltags[0] .. $b_ltags[n]:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   -- if that worked, we have the ID of the other leg in $b_leg_ltag
   log(3, $b_leg_ltag);

   -- save it
   sets($var_name=b_ltags[$(callee_counter)]);
   setVar($var_name=$b_leg_ltag);

   -- reset for new call
   clear($b_leg_ltag);
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

In the case that the A leg is canceled, we send an event to the B legs, looping the same way as we did with creating the calls:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state WAIT_CALLEE;

transition "CANCEL received" WAIT_CALLEE - hangup /  {
  dlg.reply(487, Request Terminated);
  set($callee_counter=0);
} -> CANCEL_CALLEES;

state CANCEL_CALLEES
 enter {
   repost();
 };

transition "one more to cancel" CANCEL_CALLEES - test($callee_counter<$db.rows) / {
  sets($var_name=b_ltags[$(callee_counter)]);
  var(cancel_ltag=$var_name);
  set(a_status=CANCEL);
  postEvent($cancel_ltag, a_status);
  inc(callee_counter);
} -> CANCEL_CALLEES;

transition "canceled all" CANCEL_CALLEES - test($callee_counter==$db.rows) / stop(false) -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Now it's time to create the script for the callee legs. For the beginning, this will only play a prompt, and end the call if an event with a_status==CANCEL is received:

```
quizconnect_callee.dsm:
------------------------------------------------------
import(mod_dlg);

initial state START;
transition "got INVITE in callee leg" START - invite -> RUN_INVITE;
transition "session starts in callee leg" START - sessionStart / {
  sets($prompt_name=$(config.prompt_path)/welcome_callee.wav)
  playFile($prompt_name);
} -> ENTER_PIN;

state RUN_INVITE
 enter {
```

```
   logAll(3);
 };

state ENTER_PIN;

transition "got cancel from A leg" ENTER_PIN - event(#a_status==CANCEL) / stop(true) -> END;

state END;
-----------------------------------------------------------

Time to try that out! I am actually replacing the hard-coded outbound domain with a config variable:

quizconnect_caller.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   set(b_leg_callee=$callee);
   set(b_leg_domain=$config.outbound_domain);
   set(b_leg_app=quizconnect_callee);
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
so that I can set the outbound domain in quizconnect.conf:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
db_url=mysql://user:pwd@127.0.0.1/quizconnect
outbound_domain=192.168.5.106:5080
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

and for testing, for the moment I'll start a sipp uas responder:
$ sipp -sn uas -i 192.168.5.106 -mi 192.168.5.106 -p 5080

If I cancel the test call, I can see the two calls in sipp, which were in after ACK, receive BYEs. Nice!

So, now it would be nice to check what's actually going on with the calls, in which state they are etc, without always ahving
to scroll through countless log lines. Fortunately, the 'monitoring' module helps, because if it is loaded, the core saves
information about all running calls to the monitoring in-memory database, and if we enable two features in

dsm.conf:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
monitoring_full_stategraph=yes
monitoring_full_transitions=yes
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

DSM also saves the states that were visited and the transitions to monitoring. Lets check it out from the python console over
xmlrpc (that's why we have set direct_export=dsm;monitoring in xmlrpc2di.conf):
-----------------------------------------------------------
>>> s.list()
['24ECAEFD-4C72F33D000438F1-B6843B70', '0FA6BCB2-4C72F33D0004E577-B6843B70', '59FA4B10-4C72F33D0003C671-B6944B70']
>>> s.get(s.list()[0])
[{'from': '<sip:quizconnect@192.168.5.106:5080>', 'ruri': 'sip:john@192.168.5.106:5080', 'app': 'quizconnect_callee', 'to':
'<sip:john@192.168.5.106:5080>', 'dsm_state': 'ENTER_PIN', 'dsm_stategraph': [0, 'quizconnect_callee/START', '> got INVITE in
callee leg >', 'quizconnect_callee/RUN_INVITE', 'quizconnect_callee/START', '> session starts in callee leg >',
'quizconnect_callee/ENTER_PIN'], 'dsm_diag': 'quizconnect_callee', 'dir': 'out'}]
>>> s.get(s.list()[1])
[{'from': '<sip:quizconnect@192.168.5.106:5080>', 'ruri': 'sip:anna@192.168.5.106:5080', 'app': 'quizconnect_callee', 'to':
'<sip:anna@192.168.5.106:5080>', 'dsm_state': 'ENTER_PIN', 'dsm_stategraph': [0, 'quizconnect_callee/START', '> got INVITE in
callee leg >', 'quizconnect_callee/RUN_INVITE', 'quizconnect_callee/START', '> session starts in callee leg >',
'quizconnect_callee/ENTER_PIN'], 'dsm_diag': 'quizconnect_callee', 'dir': 'out'}]
>>> s.get(s.list()[2])
[{'from': '"bee" <sip:5@192.168.5.106>', 'ruri': 'sip:35@192.168.5.106', 'app': 'quizconnect_caller', 'to':
'<sip:35@192.168.5.106>', 'dsm_state': 'WAIT_CALLEE', 'dsm_stategraph': [0, 'quizconnect_caller/START', '> got INVITE in
caller leg >', 'quizconnect_caller/RUN_INVITE', '> we have results >', 'quizconnect_caller/CREATE_CALLEE_LEGS', '> create one
more >', 'quizconnect_caller/CREATE_CALLEE_LEGS', '> create one more >', 'quizconnect_caller/CREATE_CALLEE_LEGS', '> done
creating callee legs >', 'quizconnect_caller/WAIT_CALLEE', 'quizconnect_caller/START', '> got INVITE in caller leg >',
'quizconnect_caller/WAIT_CALLEE'], 'dsm_diag': 'quizconnect_caller', 'dir': 'in'}]
>>>
-----------------------------------------------------------

so we see two calls running the quizconnect_callee diagram in ENTER_PIN state, and one running the quizconnect_caller in
WAIT_CALLEE.

two simple scripts are also useful:
$ ./sems-sessions-states.py
calls: 3
call id                                 state
6ABC60BB-4C72F4BF0005C7FD-B6A45B70     WAIT_CALLEE
19D97994-4C72F4BF0006A862-B65D5B70     ENTER_PIN
3EC2A856-4C72F4BF00062B82-B65D5B70     ENTER_PIN
$ ./sems-get-session.py 6ABC60BB-4C72F4BF0005C7FD-B6A45B70
calls: 3
attrib                                  val
from                "bee" <sip:5@192.168.5.106>
ruri                sip:35@192.168.5.106
app                 quizconnect_caller
to                  <sip:35@192.168.5.106>
dsm_state           WAIT_CALLEE
dsm_stategraph      [0, 'quizconnect_caller/START', '> got INVITE in caller leg >', 'quizconnect_caller/RUN_INVITE', '> we
have results >', 'quizconnect_caller/CREATE_CALLEE_LEGS', '> create one more >', 'quizconnect_caller/CREATE_CALLEE_LEGS', '>
create one more >', 'quizconnect_caller/CREATE_CALLEE_LEGS', '> done creating callee legs >',
'quizconnect_caller/WAIT_CALLEE', 'quizconnect_caller/START', '> got INVITE in caller leg >',
'quizconnect_caller/WAIT_CALLEE']
dsm_diag            quizconnect_caller
dir                 in


-----------------------------------------------------------
$ cat sems-sessions-states.py
#!/usr/bin/python
from xmlrpclib import *
s = ServerProxy("http://127.0.0.1:8090")
print "calls: %d" % s.calls()
calls_states =  s.getAttributeActive("dsm_state")
print "call id                                 state"
for state in calls_states:
        print state[0] + "     " + state[1]
```

```
-------------------------------------------------------


-------------------------------------------------------
$ cat sems-get-session.py
#!/usr/bin/python
import sys
from xmlrpclib import *

if len(sys.argv) != 2:
        print "usage: %s <call_id>" % sys.argv[0]
        sys.exit(1)

s = ServerProxy("http://127.0.0.1:8090")
print "calls: %d" % s.calls()
calls_states =  s.get(sys.argv[1])
if len(calls_states)==0:
        print "call not found"
        sys.exit(1)
print "attrib                                  val"
for state in calls_states[0].items():
        print str(state[0]).ljust(20) + str(state[1])
-------------------------------------------------------

so, what is next? entering the PIN number, of course. that should be simple:

quizconnect_callee.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state ENTER_PIN;

transition "got cancel from A leg" ENTER_PIN - event(#a_status==CANCEL) / stop(true) -> END;

transition "pressed a number" ENTER_PIN - key(#key<10) / append($entered_pin, #key) -> TEST_PIN;
transition "pressed hash or start" ENTER_PIN - key -> TEST_PIN_FINAL;

state TEST_PIN
 enter {
   repost();
 };
transition "pin matches" TEST_PIN - test($pin==$entered_pin) -> MATCHING_PIN;
transition "pin doesn't match" TEST_PIN - test($pin!=$entered_pin) -> ENTER_PIN;

state TEST_PIN_FINAL
 enter {
   repost();
 };
transition "pin matches" TEST_PIN_FINAL - test($pin==$entered_pin) -> MATCHING_PIN;
transition "pin doesn't match" TEST_PIN_FINAL - test($pin!=$entered_pin) / {
  clear($entered_pin);
  sets($prompt_name=$(config.prompt_path)/sorry_pin_wrong.wav)
  playFile($prompt_name);
}  -> ENTER_PIN;

state MATCHING_PIN;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

That works, but we want to break the prompt when the user enters a key, so we add closePlaylist(false), which stops playback
of currently playing items in the playlist, but doesn't generate an event:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "pressed a number" ENTER_PIN - key(#key<10) / {
  closePlaylist(false);
  append($entered_pin, #key);
} -> TEST_PIN;
transition "pressed hash or start" ENTER_PIN - key / closePlaylist(false) -> TEST_PIN_FINAL;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Now I realize we need to pass the id of the caller leg to the callee leg as well, so we can post back events:

quizconnect_caller.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   -- pass $pin to other leg
   set(b_leg_var.pin=$pin);

   -- our ltag
   set(b_leg_var.a_ltag=@local_tag);

   dlg.dialout(b_leg);
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

in the MATCHING_PIN state, we let the A leg know that the callee found the solution, and we connect to the conference room
named with the ltag of the caller leg:

quizconnect_callee.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state MATCHING_PIN
 enter {
   set($b_status=MATCHED);
   postEvent($a_ltag, b_status);
   conference.join($a_ltag);
   repost();
 };
transition "ok, connected" MATCHING_PIN --> CONNECTED;

state CONNECTED;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

in the caller leg, we have to do three things if one of the callees knows the right pin:
1. cancel all the other calls
2. reply to caller with 200 OK
3. join the conference room named with the @local_tag
```

```
quizconnect_caller.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "callee got it" WAIT_CALLEE - test(#b_status==MATCHED) / set($callee_counter=0) -> CANCEL_OTHER_CALLEES;

state CANCEL_OTHER_CALLEES
 enter {
   repost();
 };

transition "one more to cancel" CANCEL_OTHER_CALLEES - test($callee_counter<$db.rows) / {
  sets($var_name=b_ltags[$(callee_counter)]);
  var(cancel_ltag=$var_name);
  set(a_status=CANCEL);
  postEvent($cancel_ltag, a_status);
  inc(callee_counter);
} -> CANCEL_OTHER_CALLEES;

transition "canceled all" CANCEL_OTHER_CALLEES - test($callee_counter==$db.rows) / {
  closePlaylist(false);
  dlg.acceptInvite(200, OK);
  conference.join(@local_tag);
} -> CONNECTED;

state CONNECTED;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

what's left now is two things only: handling BYE in CONNECTED state for both sides, and handling CANCEL when the callee leg
rings. For the first one, we need the ltag of the right callee leg, so we'll do this:

quizconnect_callee.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state MATCHING_PIN
 enter {
   set($b_status=MATCHED);
   set($b_ltag=@local_tag);
   postEvent($a_ltag, b_status;b_ltag);
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

and save it:

quizconnect_caller.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "callee got it" WAIT_CALLEE - test(#b_status==MATCHED) / {
  set($b_ltag=#b_ltag);
  set($callee_counter=0);
} -> CANCEL_OTHER_CALLEES;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

so we can use it:
quizconnect_caller.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state CONNECTED;

transition "BYE received" CONNECTED - hangup / {
  set($a_status=BYE);
  postEvent($b_ltag, a_status);
  stop(false);
} -> END;

transition "BYE in other leg" CONNECTED - event(#b_status==BYE) / {
  stop(true);
} -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

and vice versa:

quizconnect_callee.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
state CONNECTED;

transition "BYE received" CONNECTED - hangup / {
  set($b_status=BYE);
  postEvent($a_ltag, b_status);
  stop(false);
} -> END;

transition "BYE in other leg" CONNECTED - event(#a_status==BYE) / {
  stop(true);
} -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


For the second issue, we will simply add the START and RUN_INVITE states to the "got cancel from A leg" transition:

quizconnect_callee.dsm:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
transition "got cancel from A leg" (START, RUN_INVITE, ENTER_PIN) - event(#a_status==CANCEL) / stop(true) -> END;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Attached you find the two scripts, which will (with this mail) be added soon to git in doc/dsm/tutorials.

Stefan
```

> Attached is the full script that we have so far. I hope you have some fun trying it out, and I would be happy about some feedback.
>
> Best Regards
> Stefan

--
Stefan Sayer
VoIP Services Consulting and Development

Warschauer Str. 24
10243 Berlin

tel:+491621366449
sip:sayer@iptel.org
email/xmpp:stefan.sayer@gmail.com

quizconnect_callee.dsm	**Content-Type:** text/plain **Content-Encoding:** base64

---

quizconnect_caller.dsm	**Content-Type:** text/plain **Content-Encoding:** 7bit