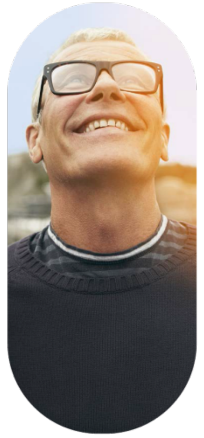
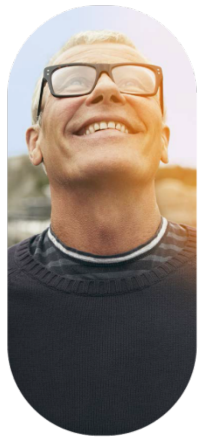
CVP Advanced Topics



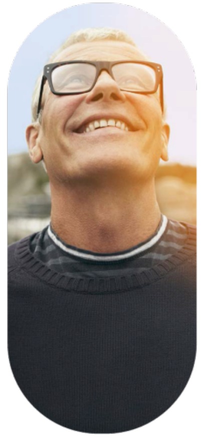
Paul Tindall
@tindallpaul

Topics We'll Examine

All of the topics covered result from actual solution requirements and work on real-world use cases where CVP is the chosen call handling platform and deployed in either Standalone or Comprehensive model

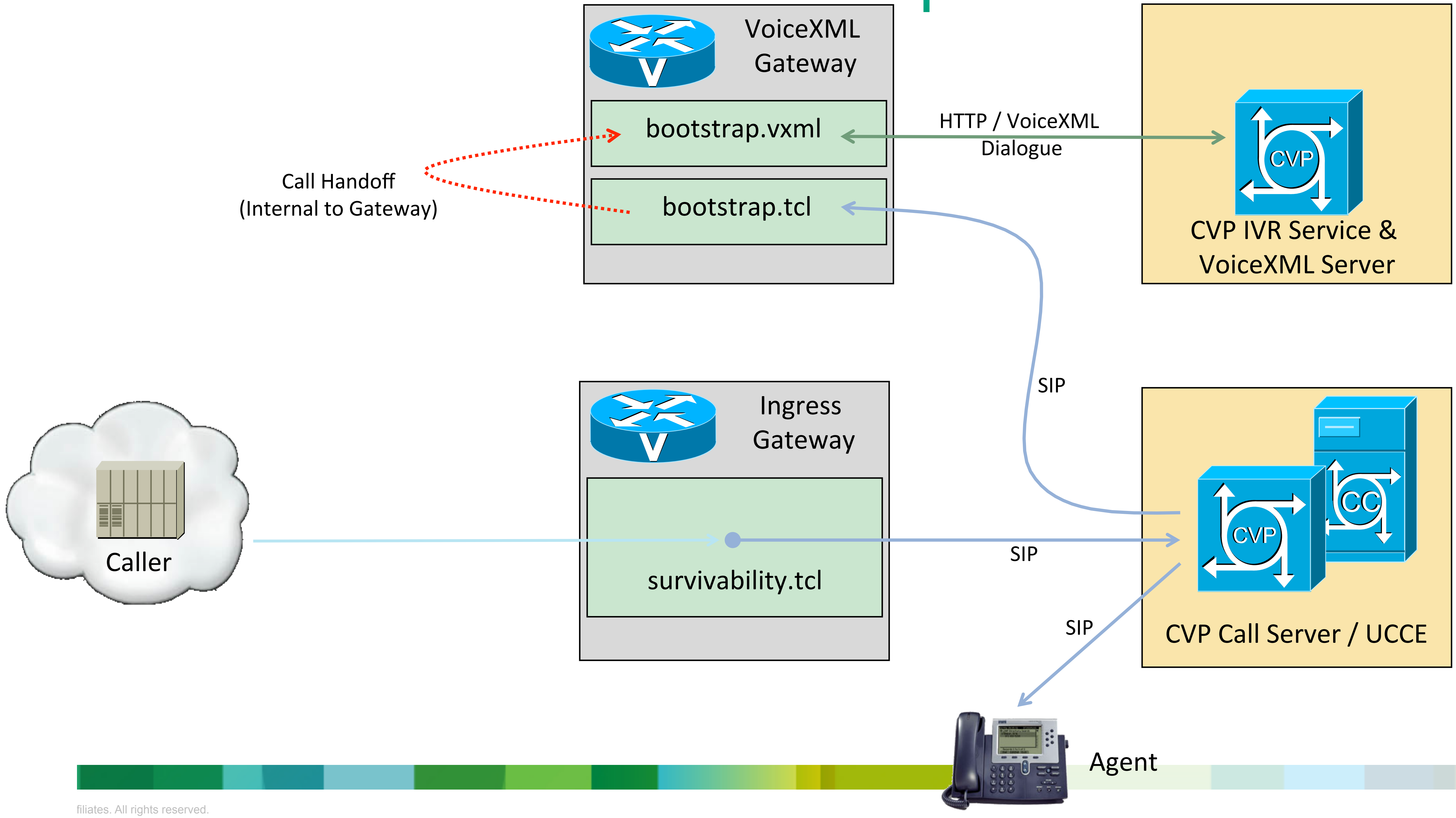


Architecture



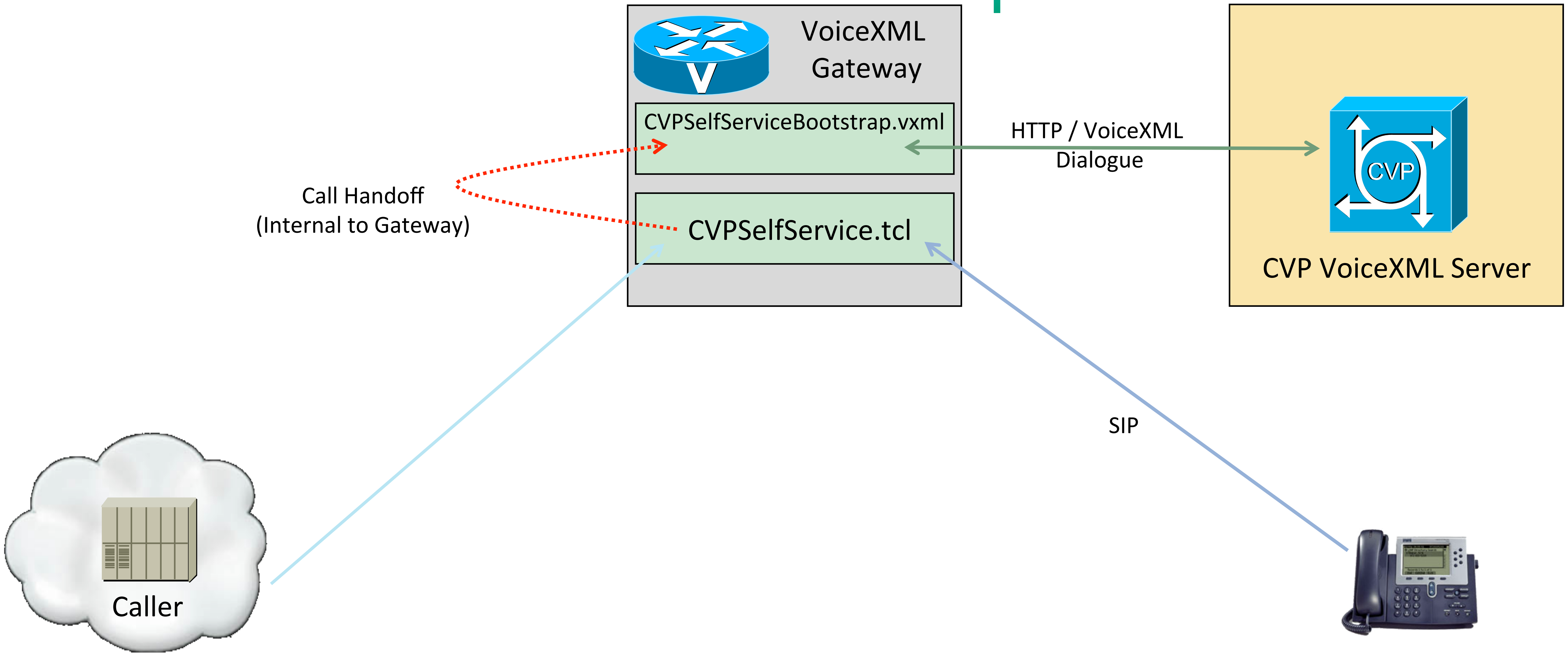
CVP TCL / VoiceXML Components

Comprehensive Model



CVP TCL / VoiceXML Components

Standalone Model



Getting/Setting SIP Headers

Use Case / Challenge

Getting SIP Headers

- Use of custom SIP headers increasingly requested as a mechanism for passing data with calls to/from CVP
- Pass user-data/context from third-party SIP PBX/ACD to CVP
- Forward user-data with transfers from CVP to third-party platforms
- Access additional signaling information such as Remote-Party-ID, physical trunk information, privacy settings

```
INVITE sip:90179017@10.52.200.50:5060 SIP/2.0
Via: SIP/2.0/UDP 10.58.16.170:5060;x-ds0num="Basic Rate Interface 0/1/0 1:DS0";branch=z9hG4bK45751B21
Remote-Party-ID: <sip:396298@10.58.16.170>;party=calling;screen=yes;privacy=off
```

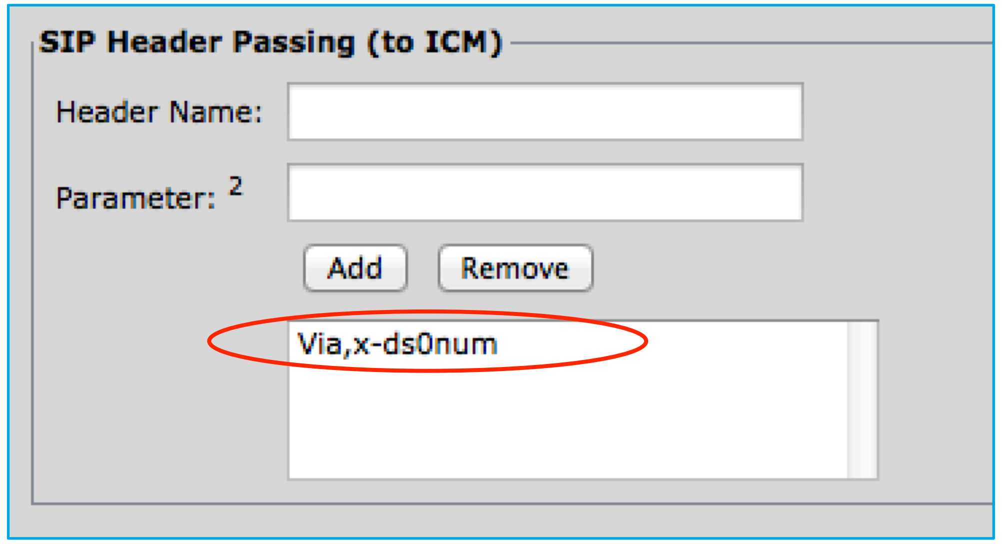
Problem: CVP Comprehensive allows SIP header content to be retrieved and added/modified but how is it done in CVP Standalone?

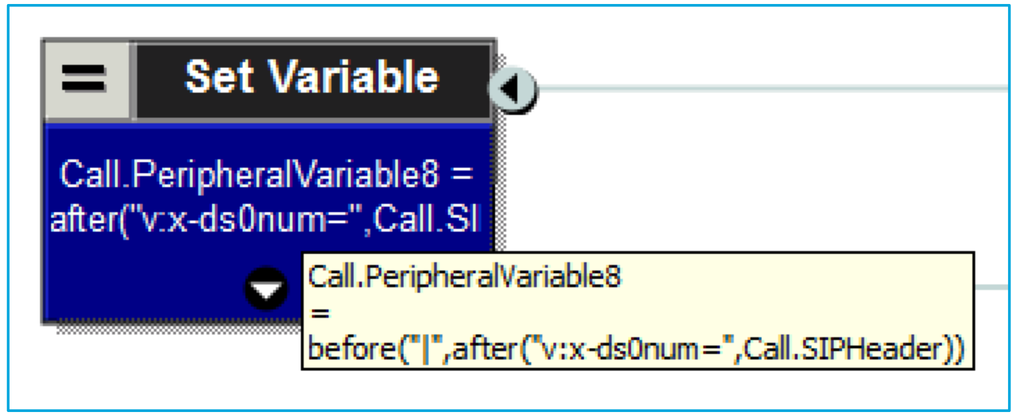
ICM Script / CVP Comprehensive Model

Getting SIP Headers

Documented in the CVP Configuration and Administration Guide

Configure CVP Call Server SIP settings via OAMP console to specify which headers and parameters should be extracted and passed up to ICM





Call.SIPHeader contains the headers configured to be passed to ICM. In this example, it contains:

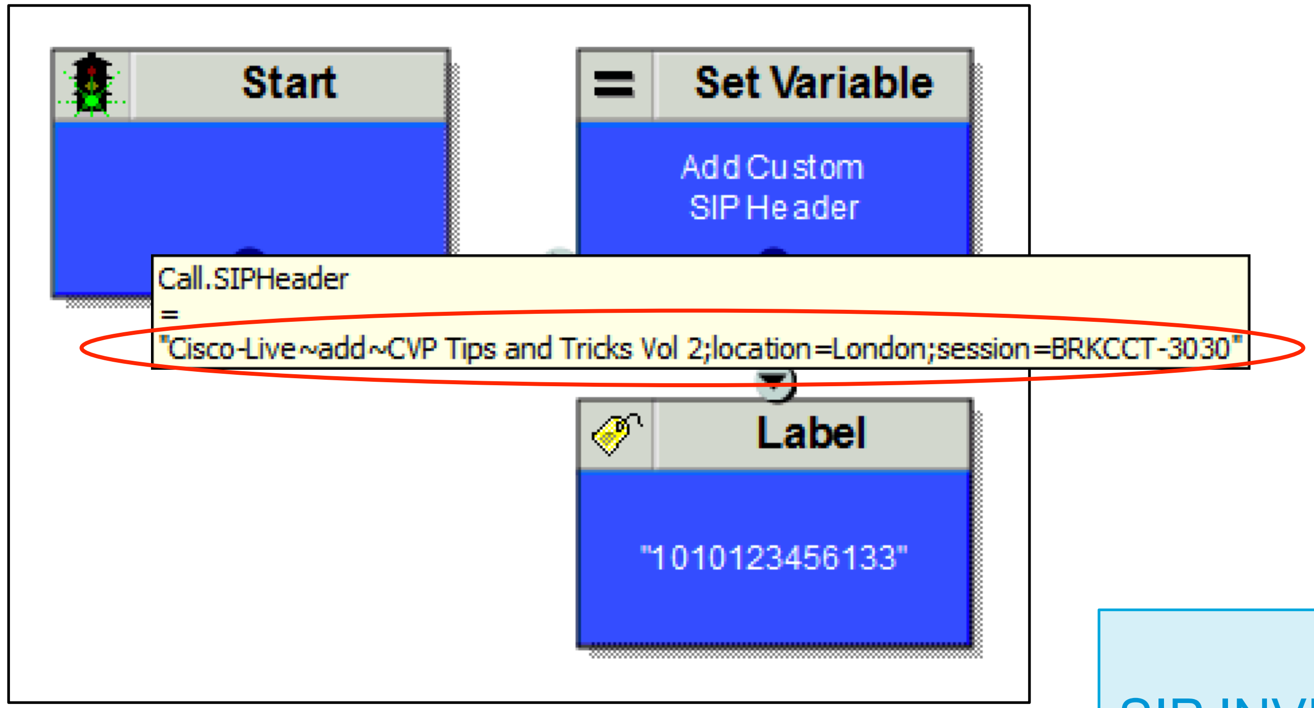
v:x-ds0num="Basic Rate Interface 0/1/0 1:DS0"

Call.SIPHeader contents are parsed to extract the right-hand-side into a call variable.

(Vertical bar "|" character is the separator if multiple items extracted)

ICM Script / CVP Comprehensive Model

Setting SIP Headers



Set variable Call.SIPHeader with the required header modifications

- Add, Modify, Remove, Substitute operations
- Vertical bar used as separator between multiple headers

Example here adds Cisco-Live custom header before CVP transfers call

SIP INVITE on CVP transferred call leg showing the Cisco-Live custom header added by the ICM script

```
INVITE sip:1010123456133@10.58.16.170;transport=tcp SIP/2.0
Via: SIP/2.0/TCP 10.58.16.180:5060;branch=z9hG4bKV0kC39vJ3IT8H+sGkW2wcg~~809
To: <sip:1010123456133@10.58.16.170;transport=tcp>
From: 396298 <sip:396298@10.58.16.180:5060>;tag=ds1f6be9
Call-ID: 76918BCF560111E2811E001B0CFAA768-1357343908541127@10.58.16.180
User-Agent: CVP 9.0 (1) Build-670
Call-Info: <sip:10.58.16.170:5060>;purpose=x-cisco-origIP
Date: Fri, 04 Jan 2013 23:58:25 GMT
Cisco-Live: CVP Tips and Tricks Vol 2;location=London;session=BRKCCT-3030
```

The custom header has been inserted

How can a CVP Call Studio application read it?

CVP Standalone Model

Getting SIP Headers

- SIP headers can be retrieved using the Cisco VoiceXML session variable session.com.cisco.proto_headers
- Invoke simple external VoiceXML using a Subdialog Invoke element
- Return the header(s) from the subdialog

  but may still need more parsing if header has multiple parameters and …

  need to avoid problems with “=“ characters in the header parameters

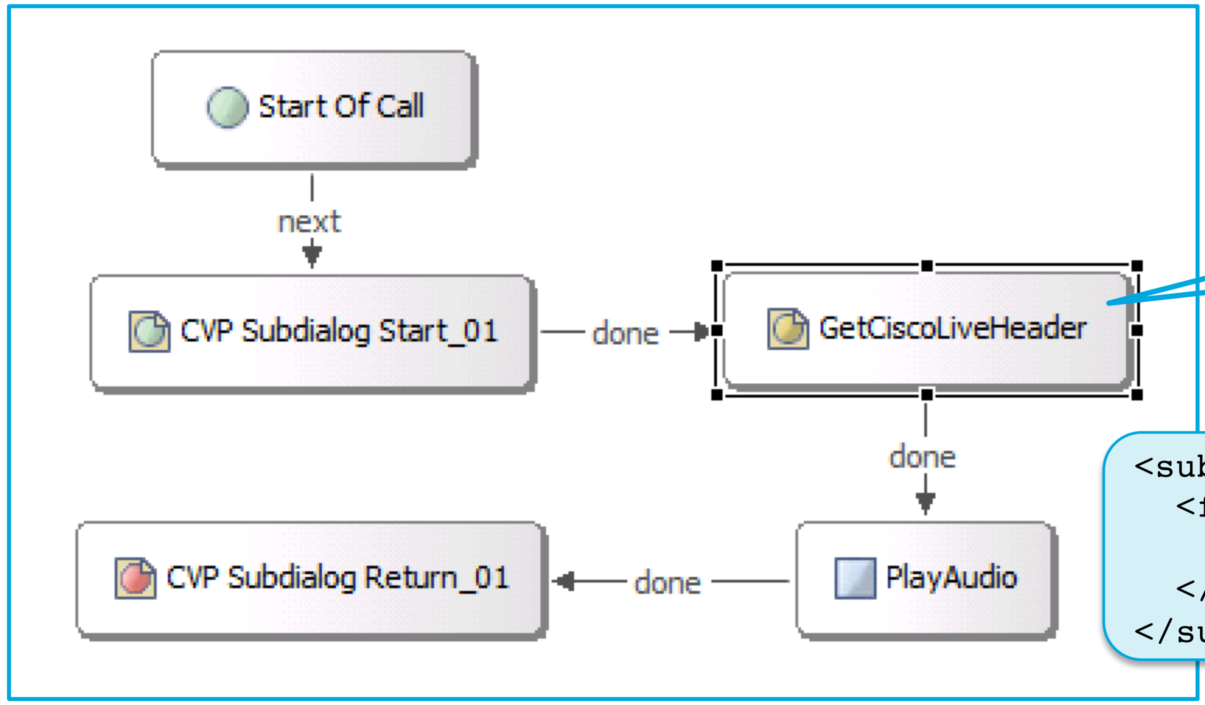
getCiscoLiveHeader.vxml

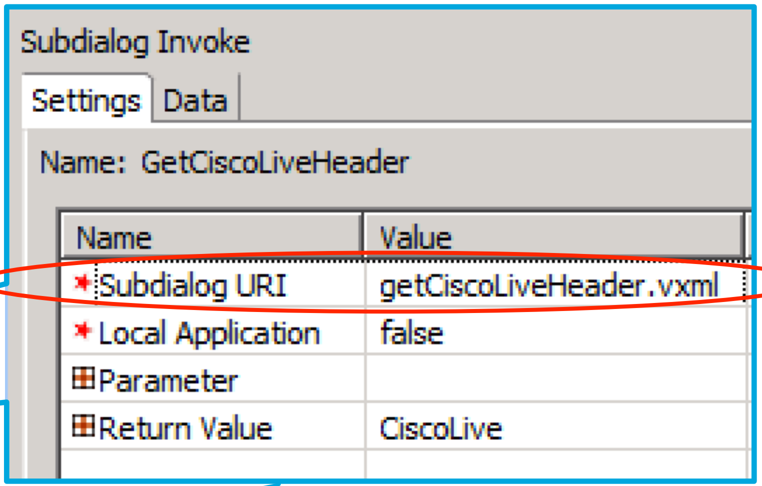
```
<?xml version="1.0"?>
<vxml version="2.0”>

  <form id="getheaders”>
    <var name="CiscoLive" expr="session.com.cisco.proto_headers['Cisco-Live'].replace(new RegExp('=', 'g' ),'::')"/>
    <block>
      <return namelist="CiscoLive"/>
    </block>
  </form>
</vxml>
```

Get Cisco-Live SIP header and return it as CiscoLive parameter from the subdialog

Using Subdialog Invoke

CVP Call Studio Application






```
<subdialog name="subdialog" src="getCiscoLiveHeader.vxml”>
  <filled>
    <submit next="/CVP/Server" method="post" namelist="subdialog.CiscoLive audium_vxmlLog subdialog”/>
  </filled>
</subdialog>
```

```
INVITE sip:1010123456133@10.58.16.170;transport=tcp SIP/2.0
Via: SIP/2.0/TCP 10.58.16.180:5060;branch=z9hG4bKV0kC39vJ3IT8H+sGkW2wcg~~809
To: <sip:1010123456133@10.58.16.170;transport=tcp>
From: 396298 <sip:396298@10.58.16.180:5060>;tag=ds1f6be9
Call-ID: 76918BCF560111E2811E001B0CFAA768-135734390854112 7@10.58.16.180
User-Agent: CVP 9.0 (1) Build-670
Call-Info: <sip:10.58.16.170:5060>;purpose=x-cisco-origIP
Date: Fri, 04 Jan 2013 23:58:25 GMT
Cisco-Live: CVP Tips and Tricks Vol 2;location=London;session=BRKCCT-3030
```

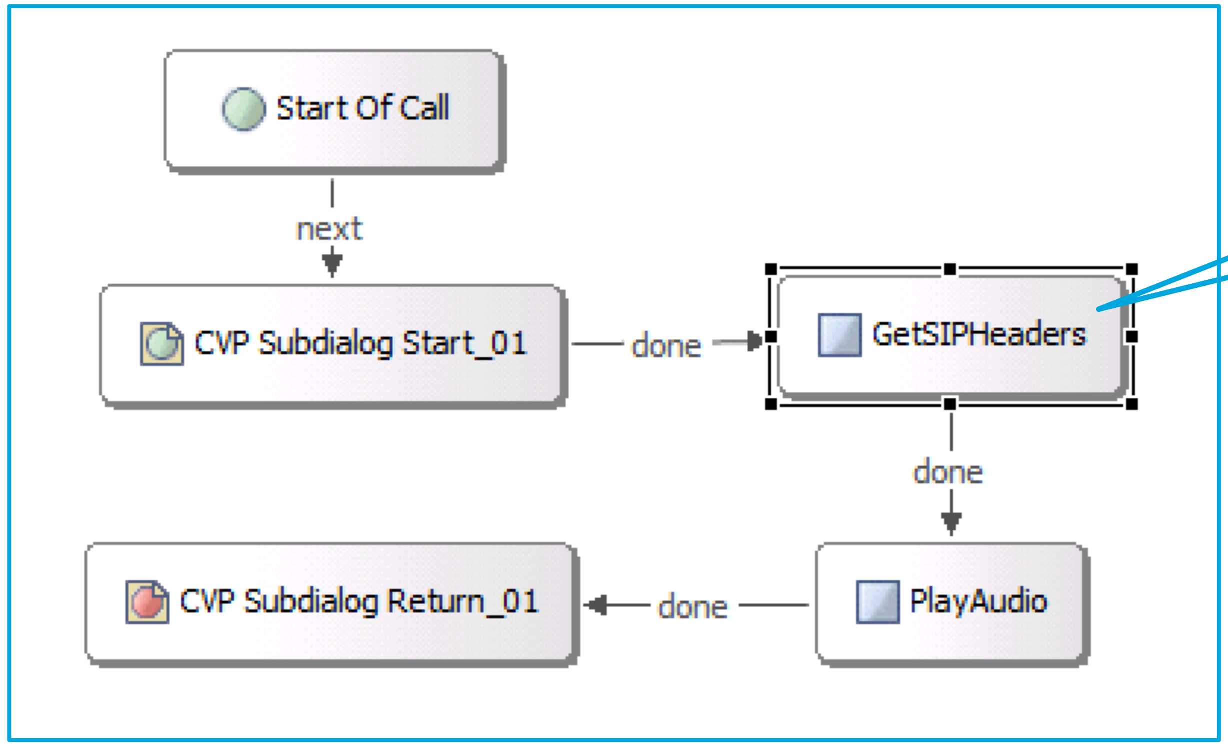
Good candidate for a custom element:

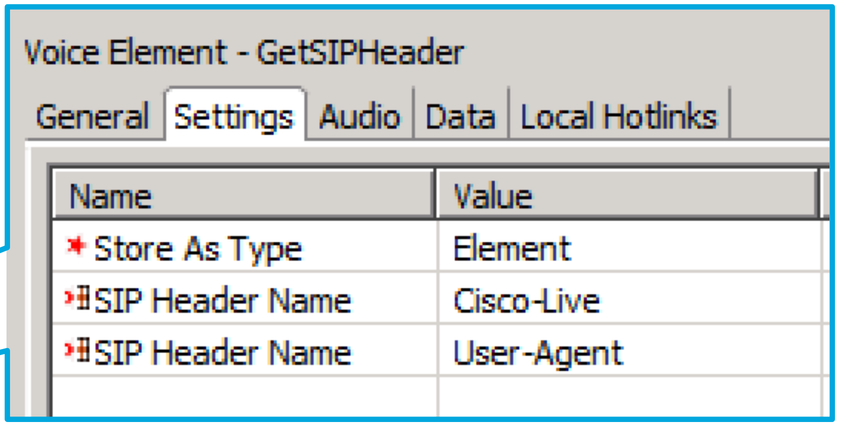
- Add settings flexibility
- Parse results into element/session data
- Avoid using external VoiceXML files

```
CL13_SIPHeader,01/06/2013 02:39:15.580,CVP Subdialog Start_01,exit,done
CL13_SIPHeader,01/06/2013 02:39:15.580,GetCiscoLiveHeader,enter,
CL13_SIPHeader,01/06/2013 02:39:15.658,GetCiscoLiveHeader,data,CiscoLive,CVP Tips and Tricks Vol 2;location::London;session::BRKCCT-3030
CL13_SIPHeader,01/06/2013 02:39:15.658,GetCiscoLiveHeader,exit,done
CL13_SIPHeader,01/06/2013 02:39:15.658,PlayAudio,enter,
```

Using Custom Element

CVP Call Studio Application





Name	Value
Store As Type	Element
SIP Header Name	Cisco-Live
SIP Header Name	User-Agent

Settings flexibility

- Specify whether header info written to element or session data
- Allow any number of headers to be retrieved

More elegant results parsing and storage

- Separate data items for each parameter
- Create element data with naming “headername.parameter”

Activity Log

```
CL13_SIPHeader,01/06/2013 00:27:02.317,GetSIPHeaders,enter,
CL13_SIPHeader,01/06/2013 00:27:02.348,GetSIPHeaders,data,Cisco-Live,CVP Tips and Tricks Vol 2
CL13_SIPHeader,01/06/2013 00:27:02.348,GetSIPHeaders,data,Cisco-Live.location,London
CL13_SIPHeader,01/06/2013 00:27:02.348,GetSIPHeaders,data,Cisco-Live.session,BRKCCT-3030
CL13_SIPHeader,01/06/2013 00:27:02.348,GetSIPHeaders,data,User-Agent,CVP 9.0 (1) Build-670
CL13_SIPHeader,01/06/2013 00:27:02.348,GetSIPHeaders,exit,done
```

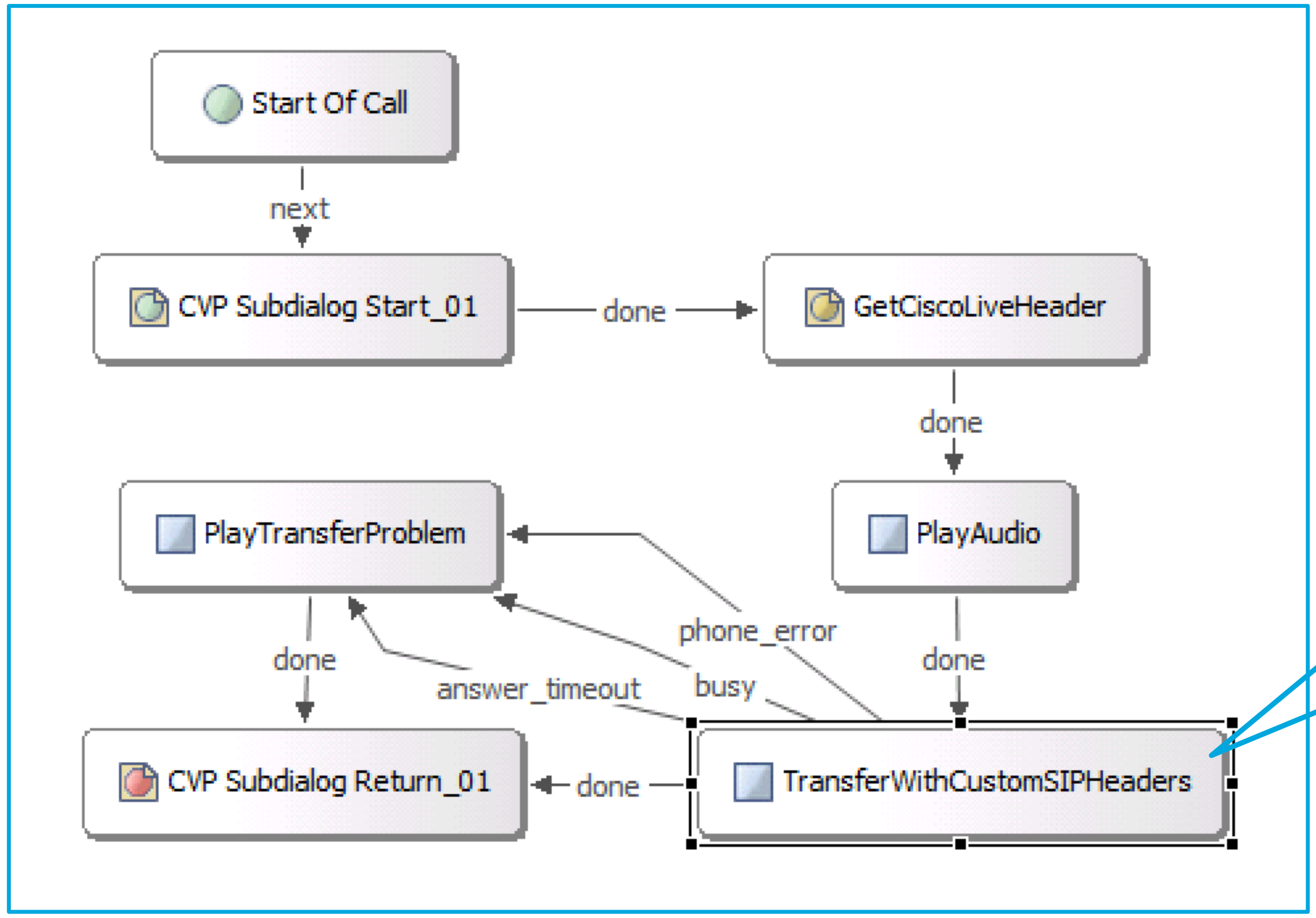
Setting SIP Headers

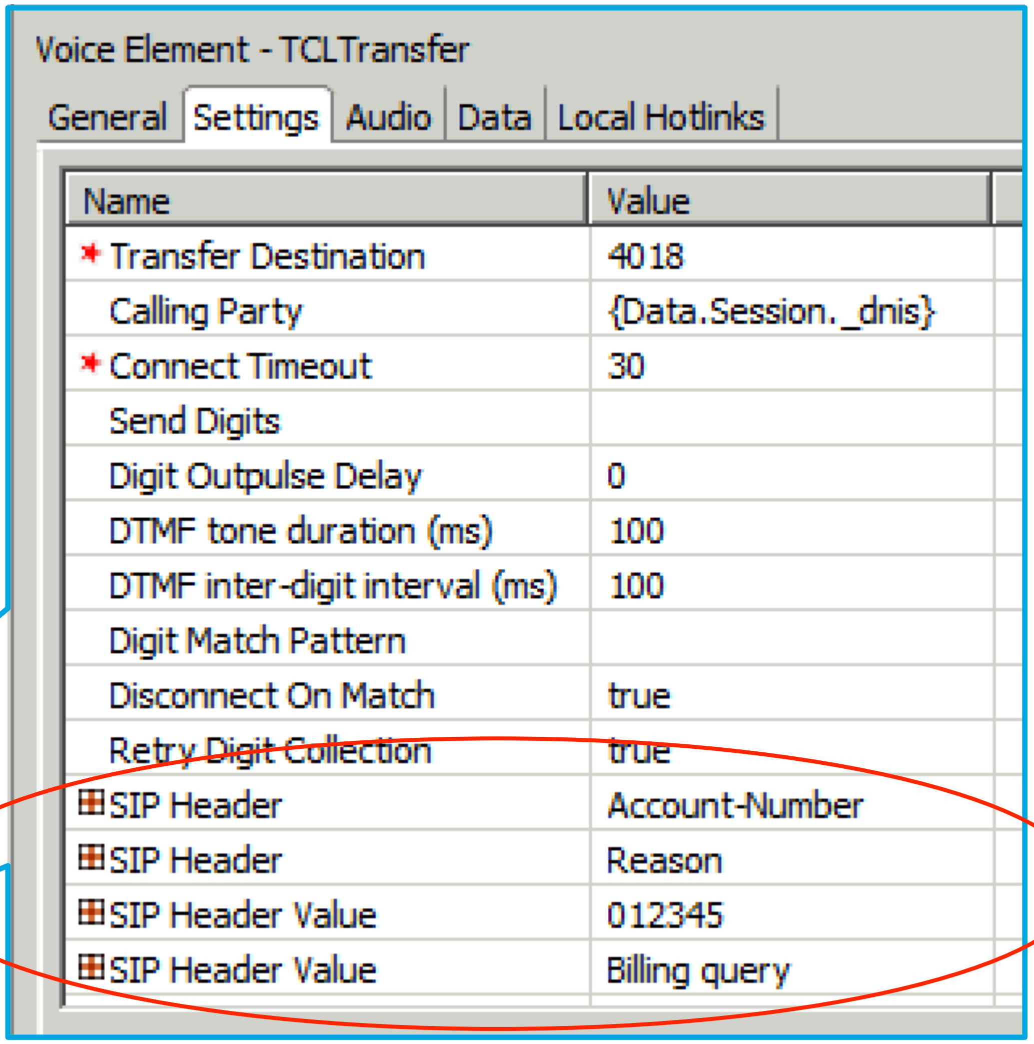
CVP Standalone Model

- Unfortunately no easy way to set SIP headers on VoiceXML transfers
- Necessary to handoff the call to TCL from VoiceXML
- An approach that enables a whole range of additional functionality, not just adding SIP headers
- Technique already used in several other places
  - CVP Standalone Outbound: sends SIP INFO messages to the voice gateway to make the call and return the outcome
  - Courtesy Callback: to initiate the callback on the ingress gateway
  - VideoConnect Element: transfers the caller to the video media server and listens for caller-side DTMF while video is playing
- Especially useful for adding custom transfer functionality as in the VideoConnect case

Using Custom Transfers

CVP Call Studio Application





- Custom transfer generates VoiceXML with <object> element to perform call handoff to TCL application cvp_tclxfer
- Data from element settings is passed to the TCL application via the arg-string param
- The TCL application retains control of the call during the transfer while the VoiceXML session is temporarily suspended

Extract from VoiceXML generated by custom transfer element

```
<object name="tclxfer" classid="builtin://com.cisco.callhandoff">
  <param name="return" expr="true" valuetype="data" />
  <param name="app-uri" expr="'builtin://cvp_tclxfer'" valuetype="data" />
  <param name="arg-string" expr="'dest=4018 rna=30 cli=651963499 pause=0 tonedur=100 tonegap=100 disc=true
reco=true siphdr=(Account-Number:012345&amp;Reason:Billing query)'" valuetype="data" />
</object>
```

CVP Call Studio Application

Using Custom Transfers

TCL transfer application is passed parameters from Call Studio script

```
CVP_TCLXFER, assumed control of call with argument: <dest=4018 rna=30 cli=651963499 pause=0 tonedur=100 tonegap=100 disc=true reco=true
siphdr=(Account-Number:012345&Reason:Billing query)>
```

Transfer leg is set up and SIP INVITE sent including custom SIP headers

```
INVITE sip:4018@10.58.16.175:5060 SIP/2.0
Via: SIP/2.0/UDP 10.58.16.170:5060;branch=z9hG4bK461E64D
From: <sip:651963499@10.58.16.170>;tag=C5AB8A80-F76
To: <sip:4018@10.58.16.175>
Date: Sun, 06 Jan 2013 18:05:10 GMT
Call-ID: 721DC9AF-576211E2-BD42CA6D-8EF6E38D@10.58.16.170
User-Agent: Cisco-SIPGateway/IOS-12.x
Reason: Billing query
Account-Number: 012345
Call-Info: <sip:10.58.16.170:5060>;purpose=X-cisco-forkingcapable
App-Info: <10.58.16.180:8000:8443>
Cisco-Live: CVP Tips and Tricks Vol 2;location=London;session=BRKCCT-3030
```

```
CVP_TCLXFER, event ev_proceeding on call leg 25268 received in state DIALING
CVP_TCLXFER, event ev_alert on call leg 25268 received in state DIALING
CVP_TCLXFER, event ev_connected on call leg 25268 received in state DIALING
CVP_TCLXFER, event ev_setup_done on call leg 25263 25268 received in state DIALING
CVP_TCLXFER, caller (leg 25263) connected to 4018 (leg 25268)
```

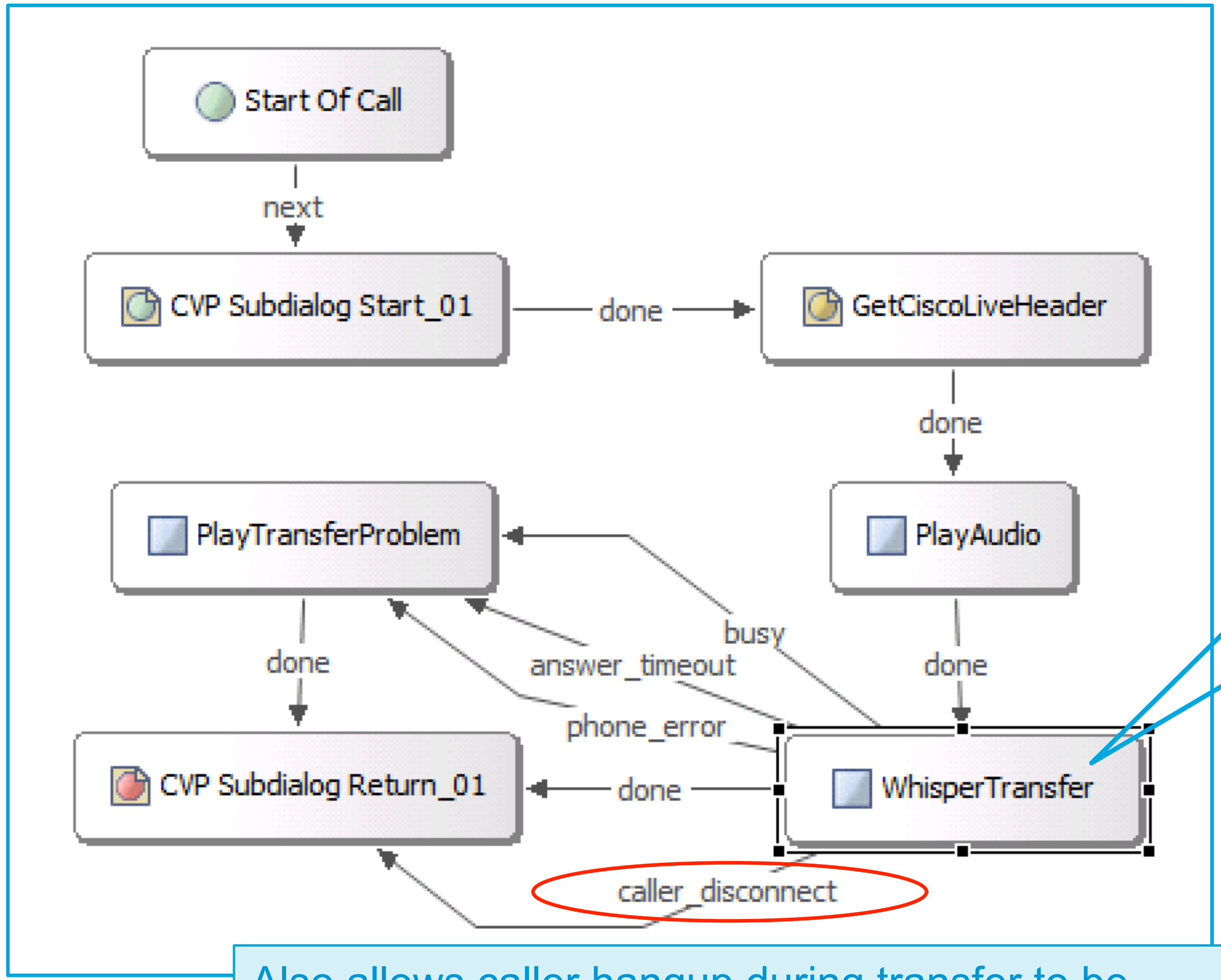
Called party answers and transfer is connected

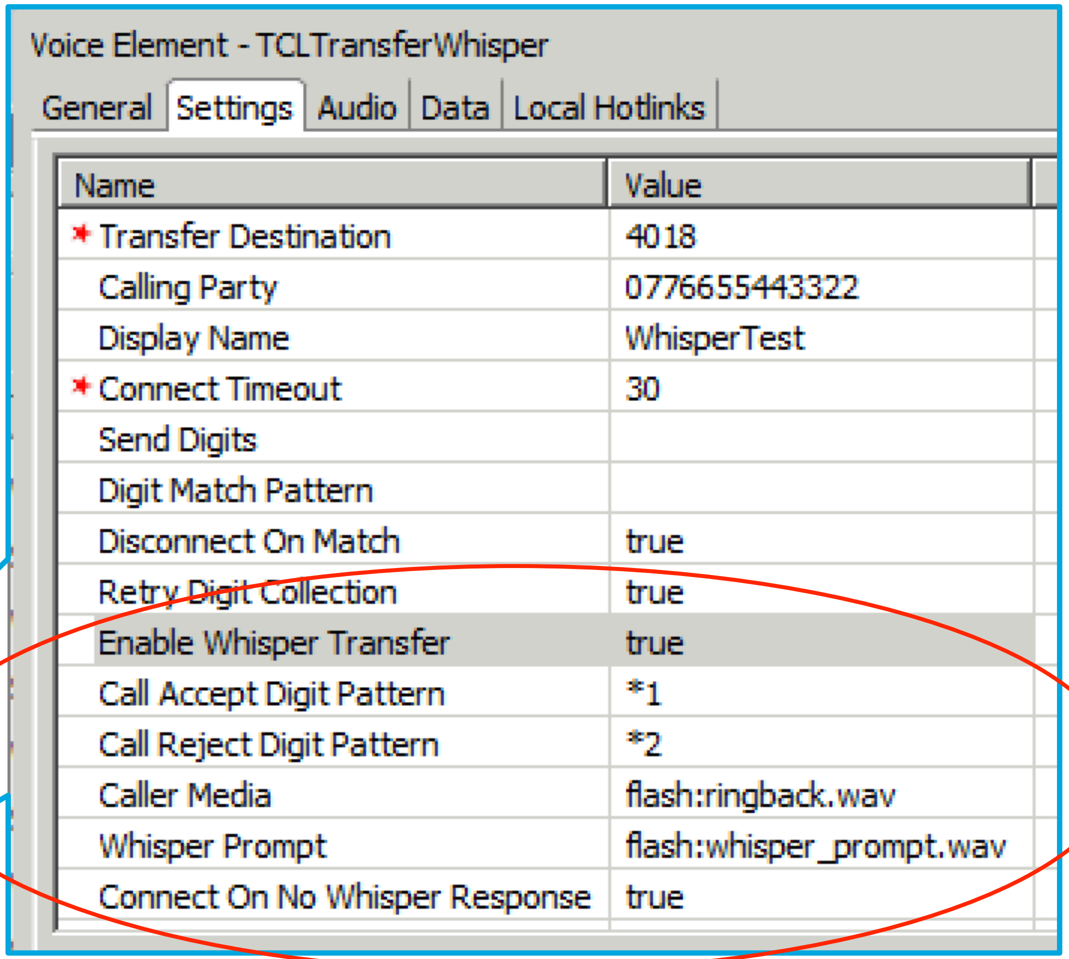
```
CVP_TCLXFER, event ev_disconnected on call leg 25268 received in state SETUP_DONE
CVP_TCLXFER, far-end disconnected, returning caller to VoiceXML
CVP_TCLXFER, event ev_destroy_done received in state FAR_END_DISC
CVP_TCLXFER, handoff return with argstring <far_end_disconnect>
CVP_TCLXFER, event ev_disconnect_done on call leg 25268 received in state FAR_END_DISC
CVP_TCLXFER, exiting
```

Called party hangs-up and call control is returned to VoiceXML

Other Custom Transfer Capability

CVP Call Studio Application





Name	Value
* Transfer Destination	4018
Calling Party	0776655443322
Display Name	WhisperTest
* Connect Timeout	30
Send Digits	
Digit Match Pattern	
Disconnect On Match	true
Retry Digit Collection	true
Enable Whisper Transfer	true
Call Accept Digit Pattern	*1
Call Reject Digit Pattern	*2
Caller Media	flash:ringback.wav
Whisper Prompt	flash:whisper_prompt.wav
Connect On No Whisper Response	true

Also allows caller hangup during transfer to be caught and presented as an exit state to the application script

- Whisper transfer with explicit called party accept/reject
- Send DTMF to called party on answer
- Receive DTMF from called party
- End transfer on DTMF pattern match
- Configurable calling party number
- Configurable display name / remote party ID
- Call parking while other party connects-in (effectively reverse direction transfer)

Thank you

REFER Transfers With Data

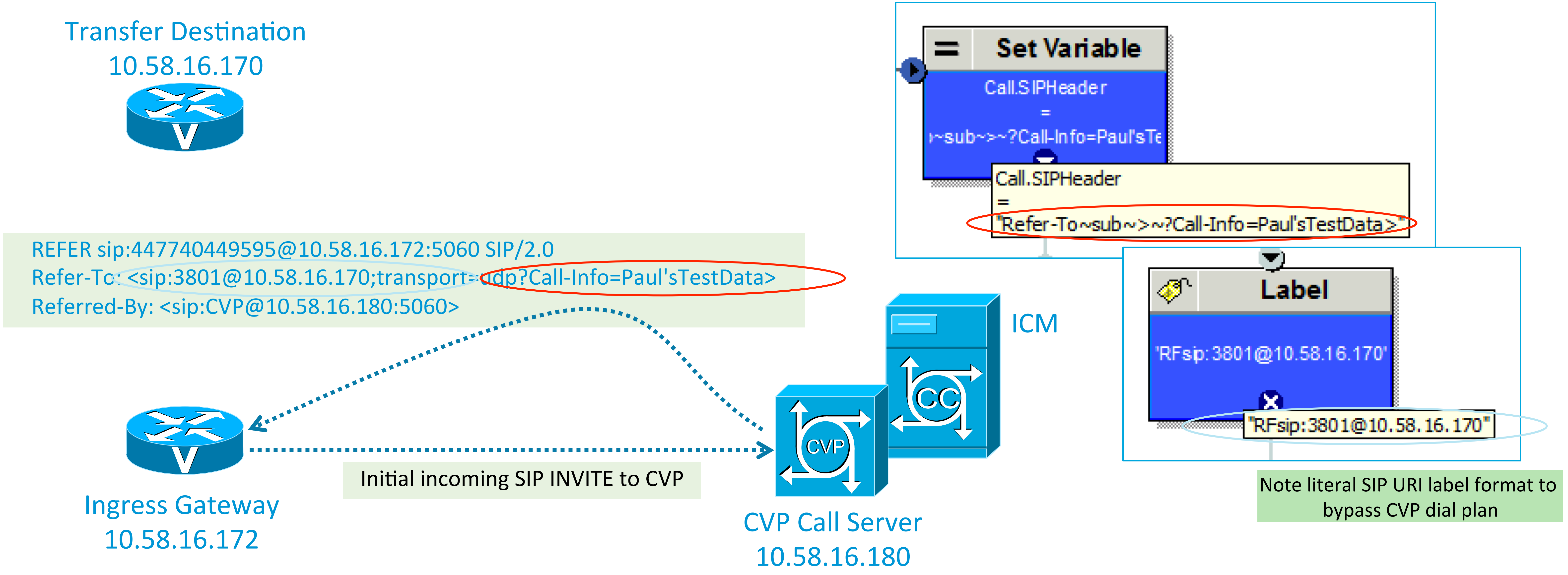
Use Case / Challenge

CVP REFERs With Data Passing

- SIP REFERs are used to instruct the calling (or called) user agent to transfer the call
- Useful for dropping CVP/ICM out of the call signalling path
  - Perform the transfer at the ingress gateway or even further back along SIP trunk
  - Typical use case is for transfers to third-party ACD/PBXs (non-CCE integrated)
  - Avoids tromboning call signalling through CVP Call Server B2BUA
  - Can also avoid tromboning media through ingress gateway depending on where the REFER is consumed

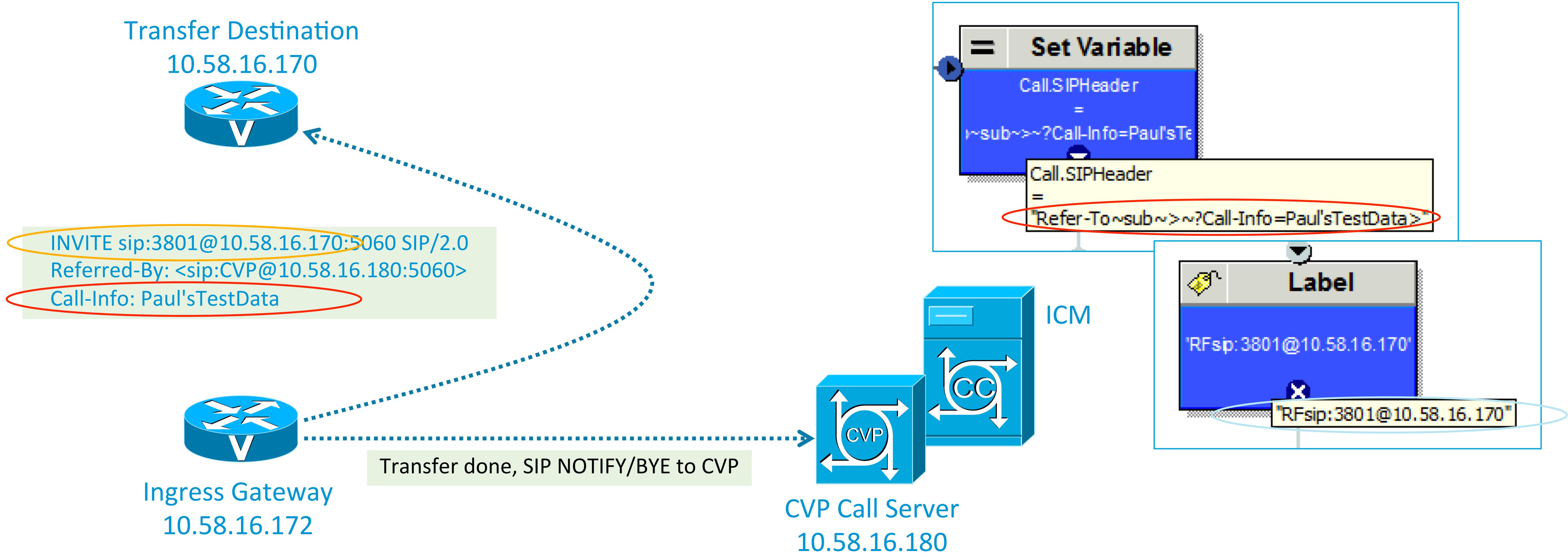
Problem: How do we pass context data with the transfer using REFER?

CVP REFERs With Data Passing



CVP REFERs With Data Passing

Ingress Gateway Consumes REFER



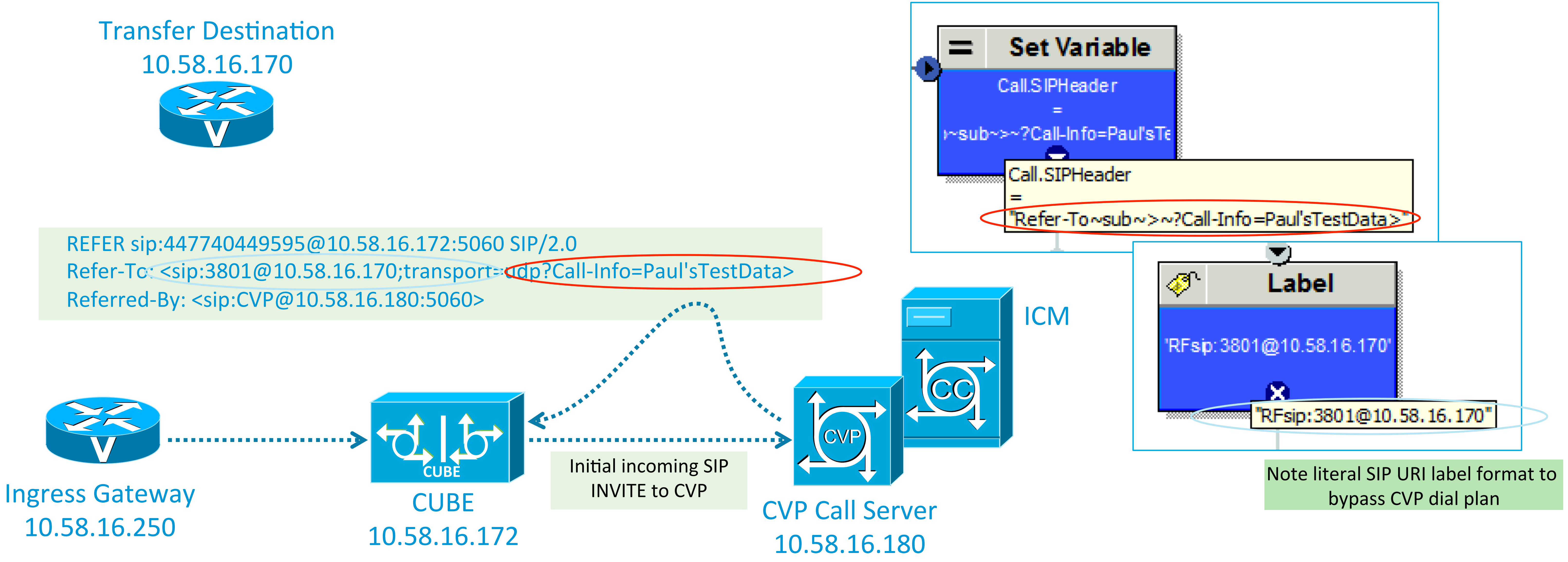
CVP REFERs With Data Passing

Not Quite Finished

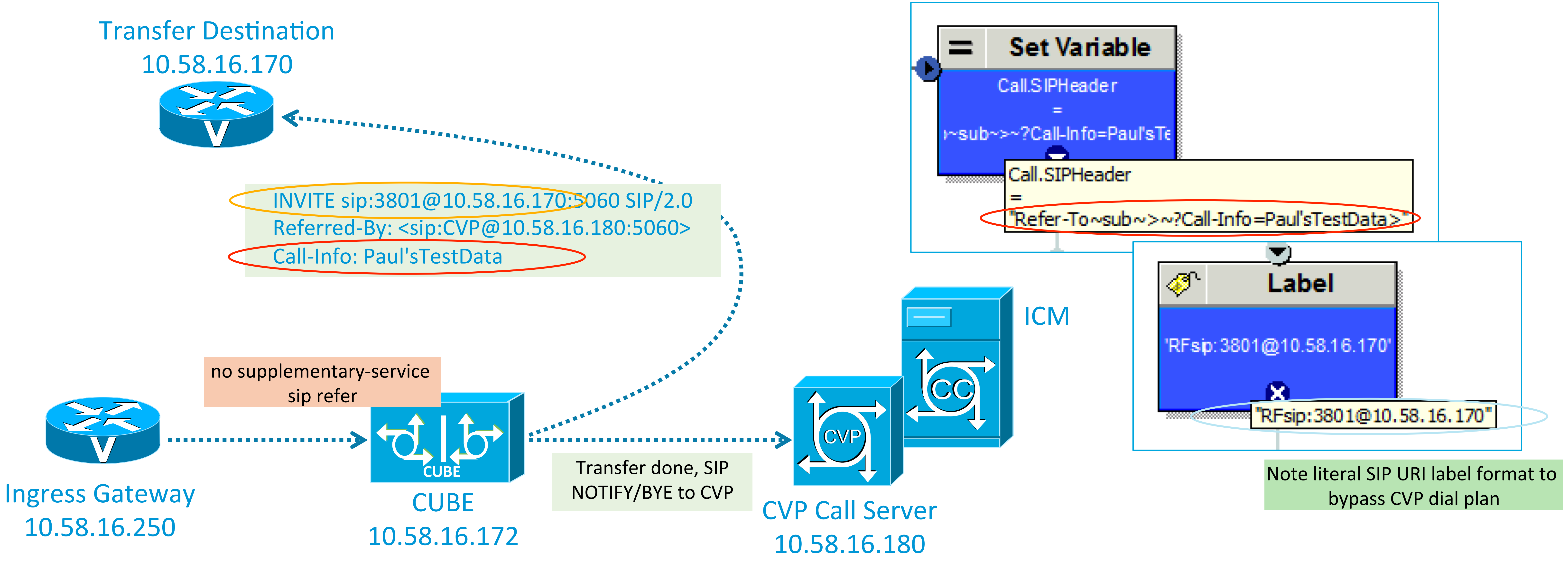
- Unfortunately not quite that straightforward to consume the REFER
- CVP Survivability intercepts the **ev_transfer_request** event when the REFER is received
- Survivability manually performs the transfer but ...
- It doesn't populate the SIP header using the Refer-To Call-Info URI parameter
  - Either don't use CVP Survivability, or ...
  - Modify it to let the gateway default handling perform the REFER
  - Comment-out the **ev_transfer_request** entry in the state machine so it's ignored

CVP REFERs With Data Passing

What if CUBE is Present?

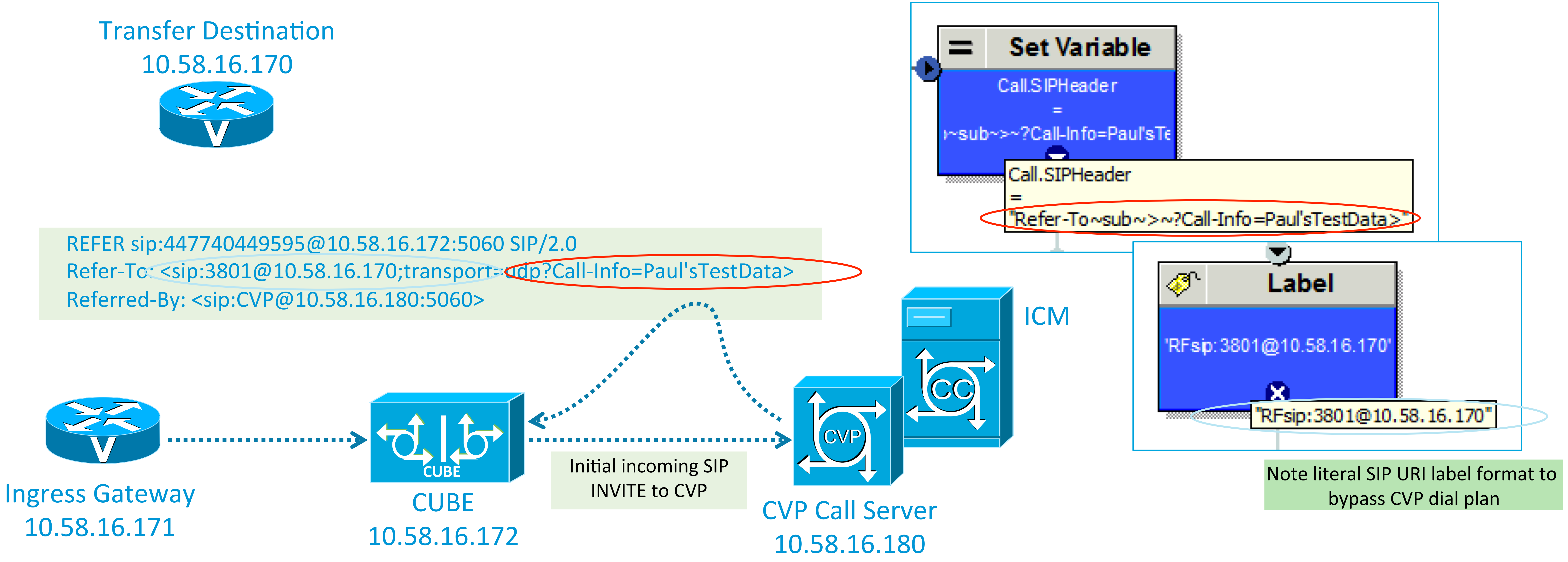


CUBE Consumes REFER
CVP REFERs With Data Passing



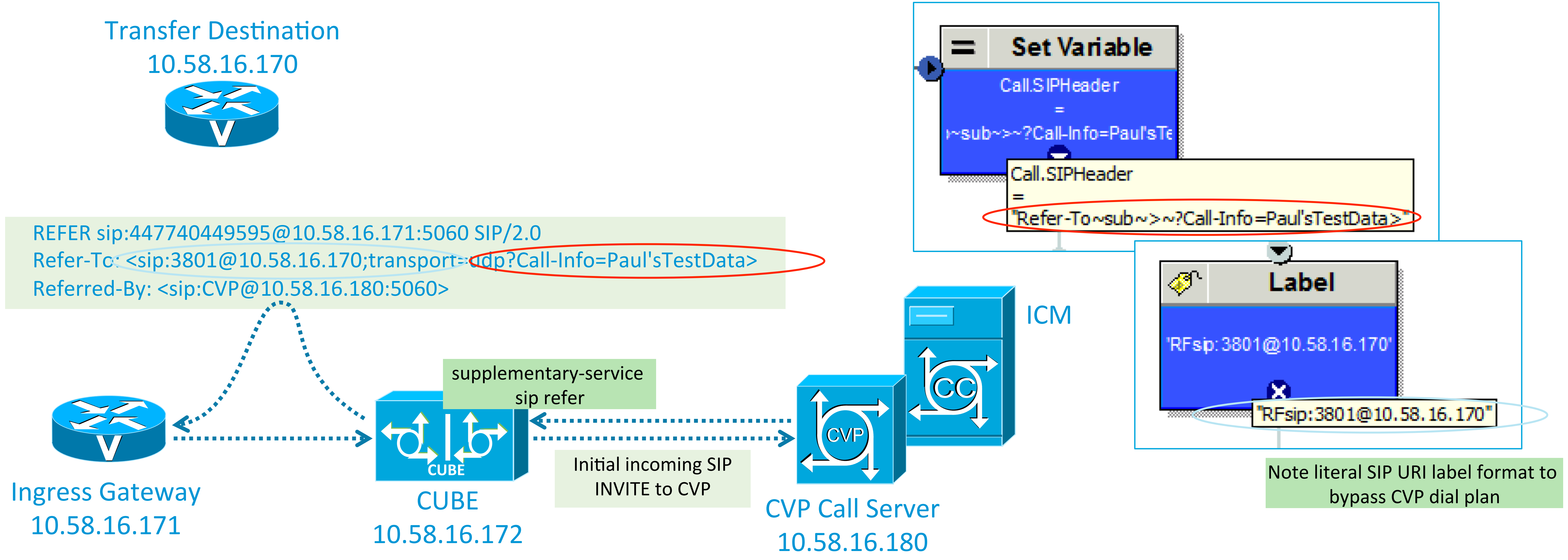
CVP REFERs With Data Passing

CUBE REFER Passthru



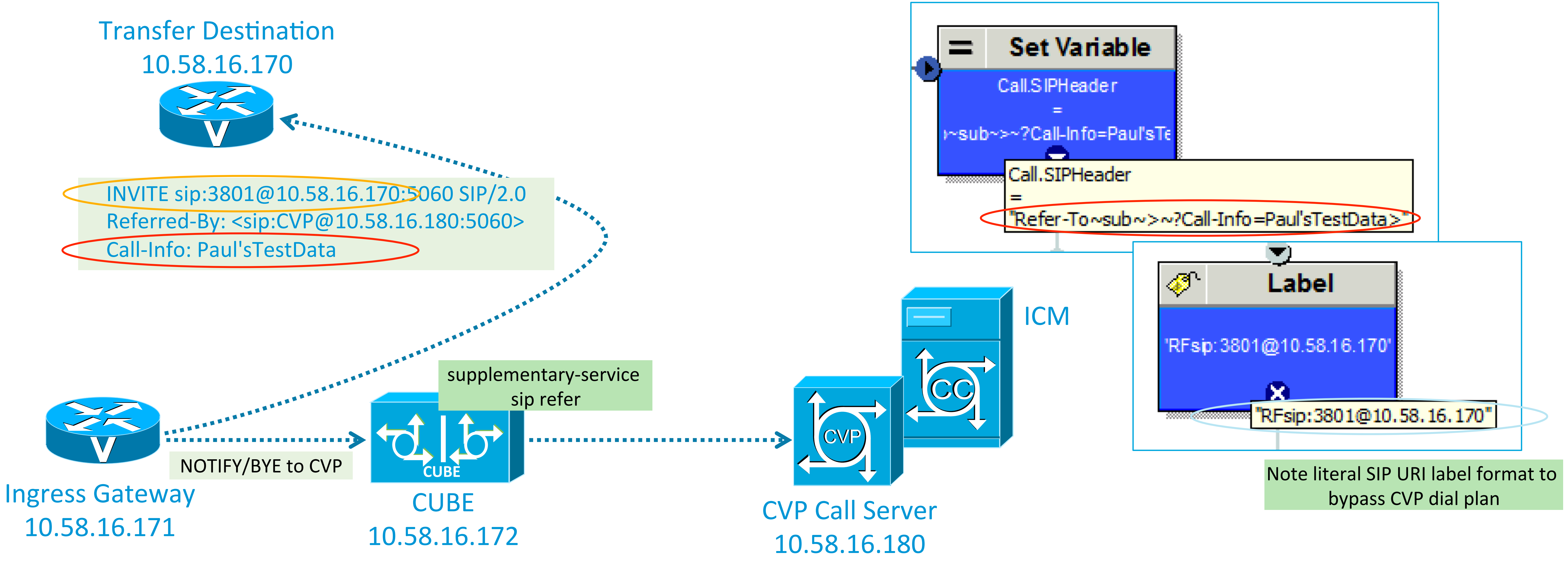
CVP REFERs With Data Passing

CUBE REFER Passthru



CVP REFERs With Data Passing

Ingress Gateway Consumes REFER



Thank you

Intelligent Call Rejection

Use Case / Challenge

Intelligent Call Rejection

- Not always desirable to answer all calls
- Sometimes want to reject calls without answering them, based on:
  - Particular call signaling content
  - Caller blacklist
  - System load
  - Number of calls active already on particular services
  - Reduce call costs for toll-free numbers
- Reject calls with specific cause code:
  - Most commonly require Busy or Unassigned Number

Problem: (a) Calls sent to IVR are answered immediately
(b) Need a way to disconnect with the required cause code

Intelligent Call Rejection

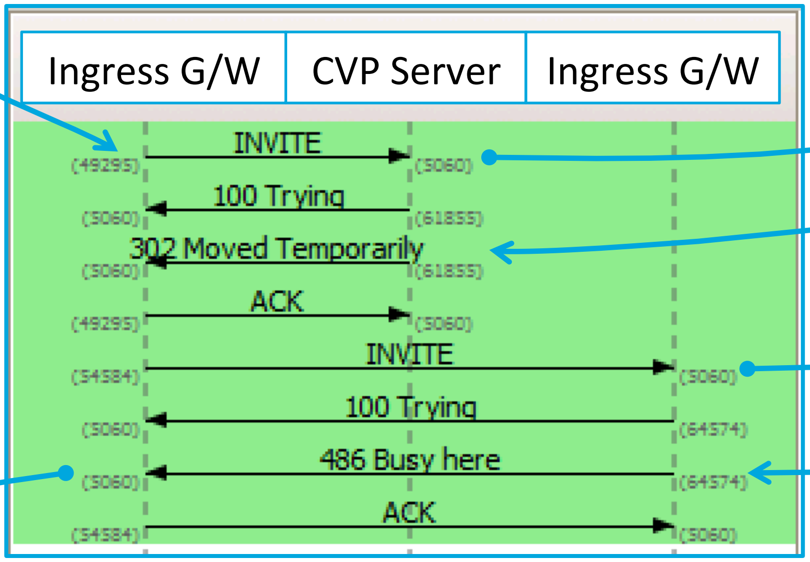
ICM Script / CVP Comprehensive Model

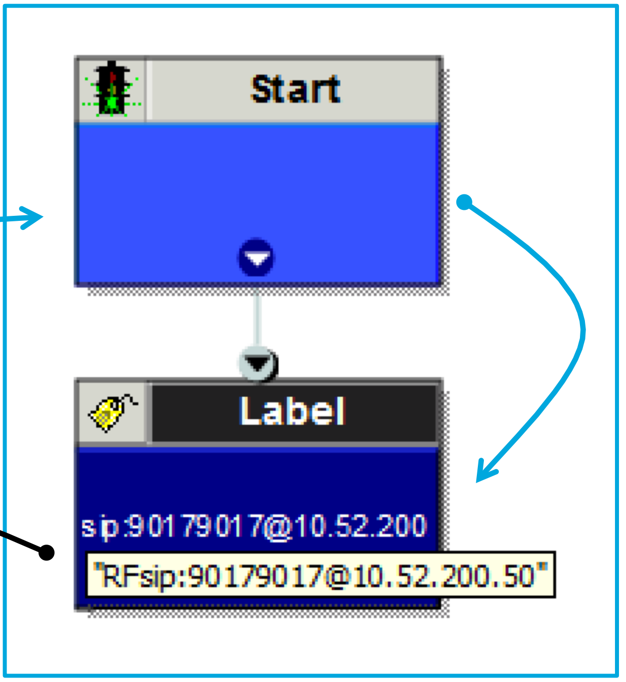
- Use REFER mode transfer with CVP immediately, before performing Send To VRU
  - REFER is only performed when CVP has already transferred/answered the call
  - Does a redirect when the call hasn't been answered
  - 302 Temporarily Moved response is sent to the calling user agent
  - Includes redirect destination in the Contact header
  - Gateway sends new SIP INVITE to the redirect destination (typically on the same gateway)
  - That destination is configured in IOS to reject the call
  - Or, could invoke a TCL script that simply does a leg disconnect with the required cause code

ICM Script / CVP Comprehensive Model

Intelligent Call Rejection

```
Jan  3 15:38:37.489: ISDN BR0/1/0 Q931: RX <- SETUP pd = 8  callref = 0x44
Jan  3 15:38:37.525: ISDN BR0/1/0 Q931: TX -> CALL_PROC pd = 8  callref = 0xC4
```






```
SIP/2.0 302 Moved Temporarily
Contact: <sip:90179017@10.52.200.50>
```

```
INVITE sip:90179017@10.52.200.50:5060 SIP/2.0
```

```
SIP/2.0 486 Busy here
Reason: Q.850;cause=17
```

```
dial-peer voice 17 voip
 call-block translation-profile incoming busy
 call-block disconnect-cause incoming user-busy
 incoming called-number 90179017
 dtmf-relay rtp-nte
 codec g711ulaw
!
voice translation-profile busy
 translate called 17
!
voice translation-rule 17
 rule 1 reject /90179017/
```

```
Jan  3 15:38:37.597: ISDN BR0/1/0 Q931: TX -> DISCONNECT pd = 8  callref = 0xC4
            Cause i = 0x8091 - User busy
Jan  3 15:38:37.689: ISDN BR0/1/0 Q931: RX <- RELEASE pd = 8  callref = 0x44
Jan  3 15:38:37.689: ISDN BR0/1/0 Q931: TX -> RELEASE_COMP pd = 8  callref = 0xC4
```

Intelligent Call Rejection

ICM Script / CVP Comprehensive Model

- REFER label can be a numeric label or SIP URI
  - SIP URI is sent as-is
  - Numeric label is resolved using CVP static dial plan
  - Use RF prefix or set ECC user.sip.refertransfer to “y”
- Remember that survivability.tcl traps abnormal disconnects (cause value not 16)
  - Either don’t use it, or …
  - Customise TCL to suppress call recovery on required failure causes, and …
  - Comment out “leg setupack leg_incoming” to prevent spurious immediate answer

Intelligent Call Rejection

ICM Script / CVP Comprehensive Model

- Survivability.tcl must disconnect rather than perform recovery on initial call setup failures as required
- Necessitates modifying CVPTransferDone procedure

```
"ls_004" {
     set msg  "******* Call setup failed: CVP number is invalid $displayStr *******"
      LogMsg "DEBUG" $msg
      set recovery 1
      RecoveryActivities
}


"ls_007" {
      set msg  "******* Call setup failed: CVP number is busy – disconnecting caller as custom action *******"
      LogMsg "DEBUG" $msg
      fsm setstate CALLER_DISCONNECTED
      leg disconnect $incoming –c 17
}
```

Intelligent Call Rejection

CVP Standalone Model

- Should be as simple as disconnecting via VoiceXML

  <disconnect cisco-disc_cause="17"/>

- But this will leave the CVP session hanging until its timeout expires
- Alternative is to let CVPSelfServiceBootstrap.vxml do the disconnect
- Normal call flow:

  Call Studio application ends via CVP Subdialog Return

  CVPSelfServiceBootstrap.vxml processes “Caller Input” return parameter

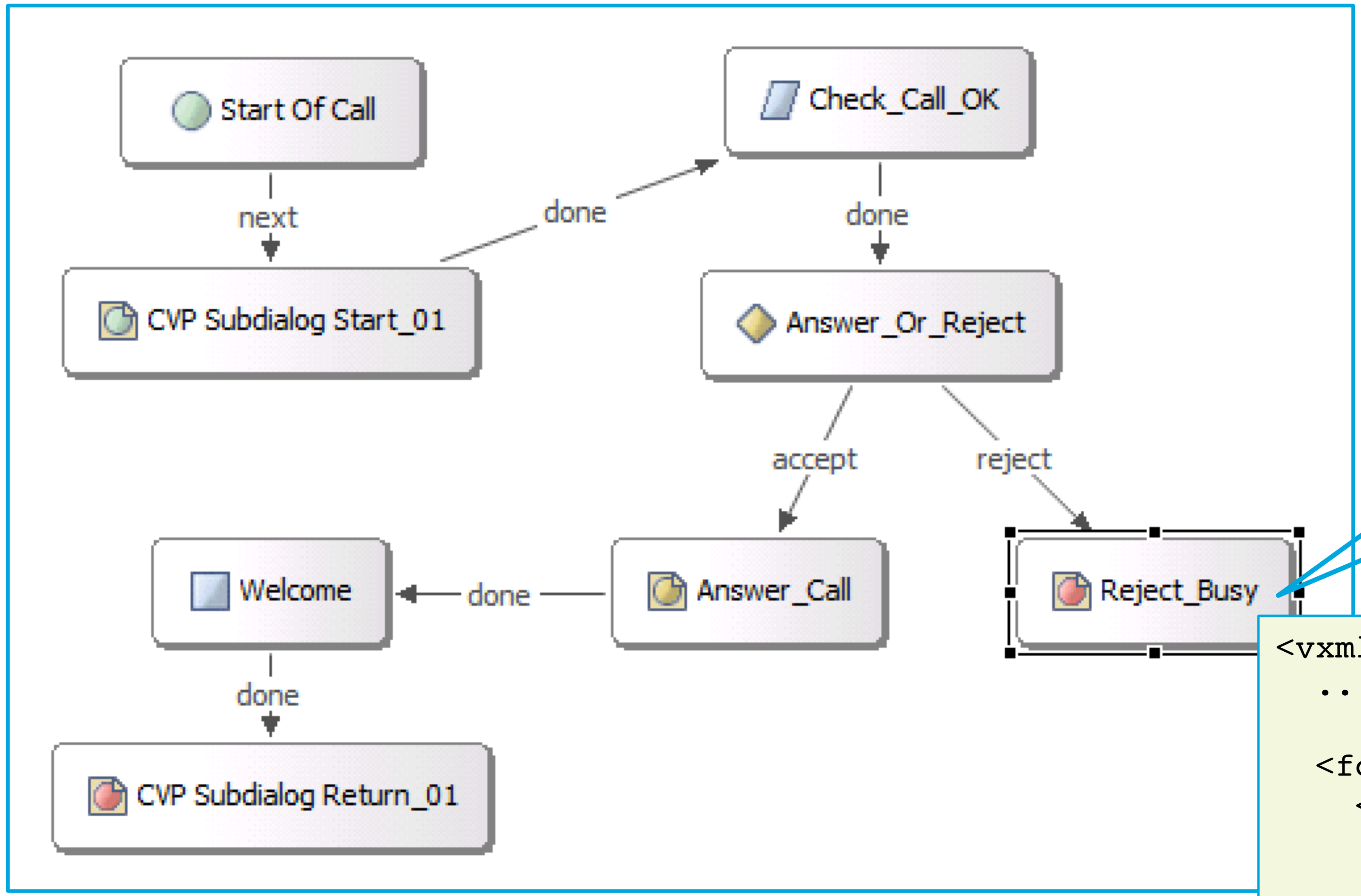
  By default, will result in Normal Clearing (16)

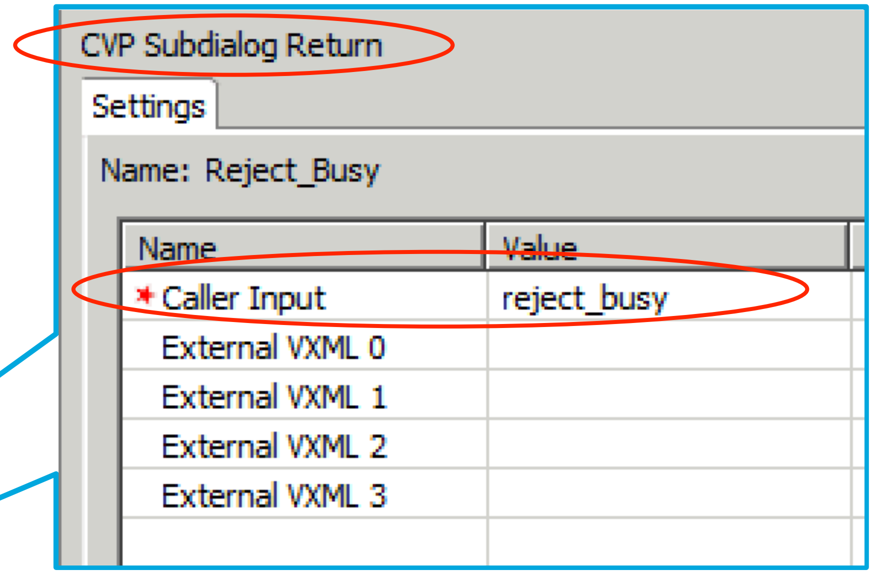
  Certain predefined values returned to indicate failures or invoke non-VoiceXML transfers

Intelligent Call Rejection

CVP Standalone Model

- Return additional custom defined values on the CVP Subdialog Return






```
<vxml version="2.0">
  ...

  <form>
    <block>
      <var name="caller_input" expr="'reject_busy'" />
      <return namelist="caller_input" />
    </block>
  </form>
</vxml>
```

Return control to CVPSelfServiceBootstrap.vxml

Intelligent Call Rejection

CVP Standalone Model

- Modify CVPSelfServiceBootstrap.vxml to process those values

  reject_busy

  reject_unassigned

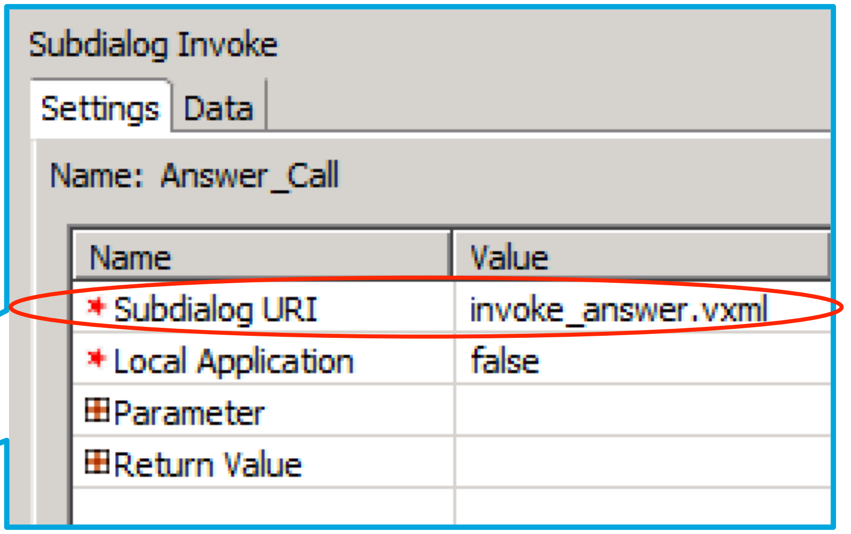
```
<subdialog name="RunCVPServer” ... >
    <filled>
        ...

        <elseif cond="RunCVPServer.caller_input == 'reject_busy'"/>
            <disconnect cisco-disc_cause="17"/>

        <elseif cond="RunCVPServer.caller_input == 'reject_unassigned'"/>
            <disconnect cisco-disc_cause="1"/>

        <else/>
            <disconnect cisco-disc_cause="16"/>
        </if>
    </filled>
</subdialog>
```

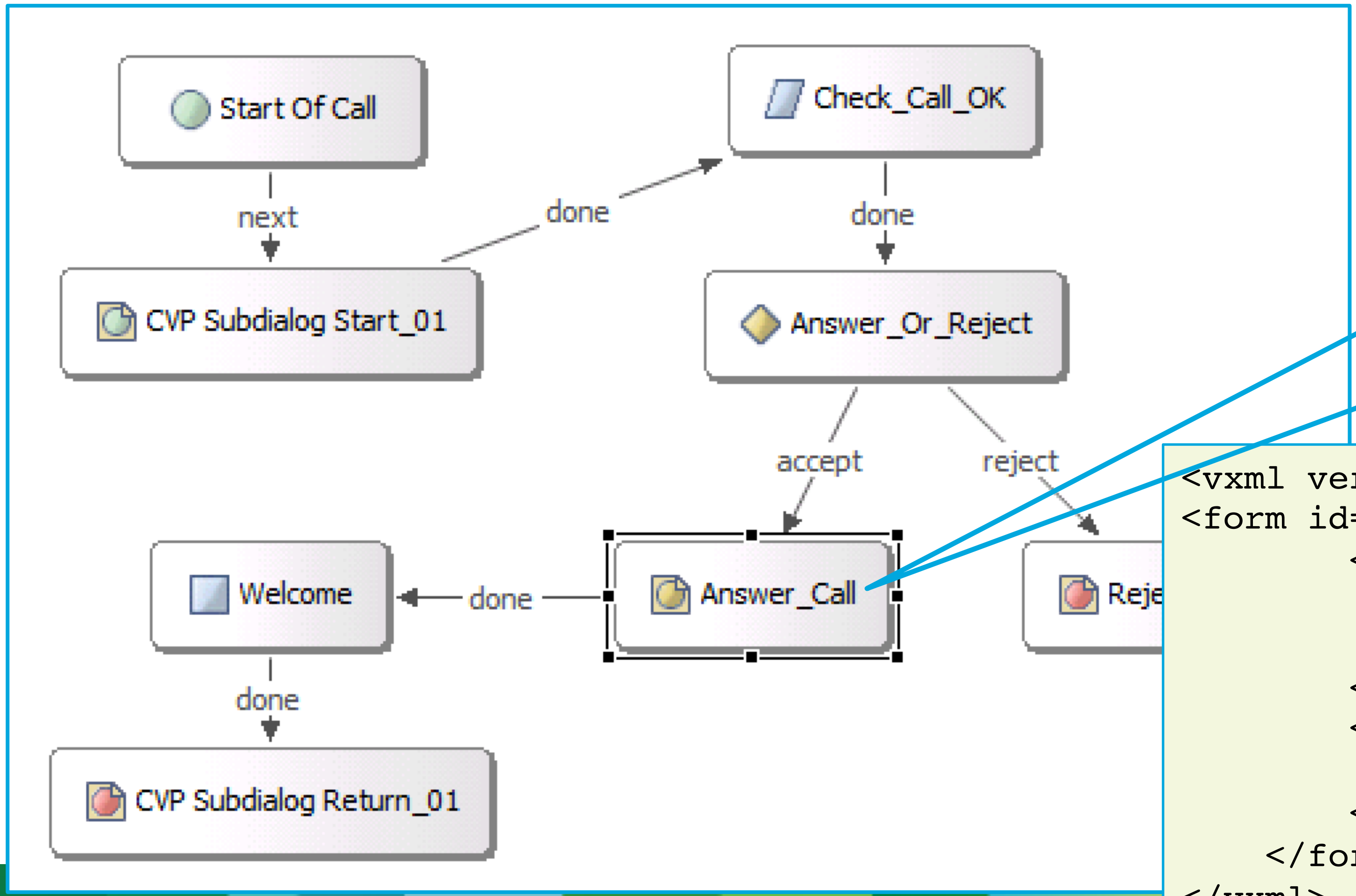
```
Jan  3 15:38:37.489: ISDN BR0/1/0 Q931: RX <- SETUP pd = 8  callref = 0x44
Jan  3 15:38:37.525: ISDN BR0/1/0 Q931: TX -> CALL_PROC pd = 8  callref = 0xC4
Jan  3 15:38:37.597: ISDN BR0/1/0 Q931: TX -> DISCONNECT pd = 8  callref = 0xC4
            Cause i = 0x8091 - User busy
Jan  3 15:38:37.689: ISDN BR0/1/0 Q931: RX <- RELEASE pd = 8  callref = 0x44
Jan  3 15:38:37.689: ISDN BR0/1/0 Q931: TX -> RELEASE_COMP pd = 8  callref = 0xC4
```

Intelligent Call Rejection

CVP Standalone Model

- Not quite finished yet – still two things remaining
  1. Not answering the call automatically when it hits the gateway
  2. Explicitly answering it from the application





invoke_answer.vxml

```
<vxml version="2.0”>
<form id="answer_call">
        <object name="handoff" classid="builtin://com.cisco.callhandoff”>
            <param name="return" expr="true"/>
            <param name="app-uri" expr="'builtin://answer'"/>
        </object>
        <block>
            <return/>
        </block>
    </form>
</vxml>
```

VoiceXML cannot directly answer the call so use an <object> element to temporarily handoff to a TCL application via service ‘answer’ to perform ‘leg connect’

Intelligent Call Rejection

CVP Standalone Model – Explicitly Answering Call

```
# Answer call and return to calling application ---------------------------

proc handoff_event { } {
            set leg1 [infotag get evt_legs]
            puts ">>> ANSWER.TCL assumed control of call leg $leg1"
            leg connect $leg1
            handoff return $leg1
}

# State machine --------------------------------

set FSM(CALL_INIT,ev_handoff)                           "handoff_event,same_state"
set FSM(any_state,ev_disconnect_done)                    "cleanup,same_state"
set FSM(any_state,ev_disconnected)                      "disconnect,same_state"

fsm define FSM CALL_INIT
```

2. Connect the call
3. Return control to VoiceXML

1. Handoff from VoiceXML triggers handoff_event procedure

Note. Disconnect handling omitted for clarity

- Handoff event handler is invoked via the state machine
- It retrieves the ID of the call leg that it now has control of
- Performs a connect on the call leg
- Returns control back to VoiceXML

Intelligent Call Rejection

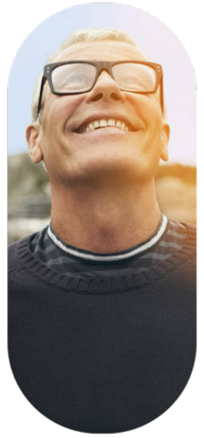
CVP Standalone Model – Preventing Immediate Answer

- CVPSelfSevice.tcl script has to be modified to not answer the call
- This does mean a minor change to a released/supported CVP TCL script but unfortunately no alternative approach
- Comment-out signalling so that only leg proceeding is executed

```
#---------------------------------------------------------------------------
# Procedure act_Start
#
# This procedure is executed when it receives an ev_setup indication event.
# A setup acknowledgement, call proceeding, and connect message are sent to
# the incoming call leg and finally the call is handed off to the CVPSelfService app.
#---------------------------------------------------------------------------
proc act_Start { } {

        ...
#       leg setupack leg_incoming
        leg proceeding leg_incoming
#       leg alert leg_incoming
#       leg connect leg_incoming
        ...
```

Thank you

Multiple Codec Handling



Multiple Codec Handling

Use Case / Challenge

- Calling endpoints making calls to automated call handling services may use different codecs
- Bigger issue now because of SIP trunking. How can we handle that?

Problem: CVP and IOS Voice Browser require the prompt encoding matches the codec for the call

Transcode all calls delivered to CVP to use a common codec

Have multiple CVP/VoiceXML applications using different prompt sets accessed via different DNs

Dynamically use the correct prompt set encoding for the call with a single application

Lookup The Codec

Dynamically Selecting The Prompt Set

- IOS 15.1(3)T – enhancement that allows TCL and VoiceXML applications to determine the currently negotiated codec dynamically at run time
- Cisco session variable session.connection.com.cisco.codec contains the name of the current codec
- Use very simple VoiceXML to access that session variable and return it to the application

```
<?xml version="1.0"?>
<vxml version="2.0">

    <form id="getcodec">
        <var name="caller_input"/>

        <block>
            <assign name="caller_input" expr="session.connection.com.cisco.codec"/
            <return namelist="caller_input"/>
        </block>
    </form>
</vxml>
```

Example custom VoiceXML called from ICM Get Speech (GS) CVP microapp

Dynamically Selecting The Prompt Set

Using ICM Script / CVP Microapps

- Set the default media server path

  For example, http://cvpsvr:7000/CVP/audio

- Invoke external VoiceXML subdialog to return the codec name

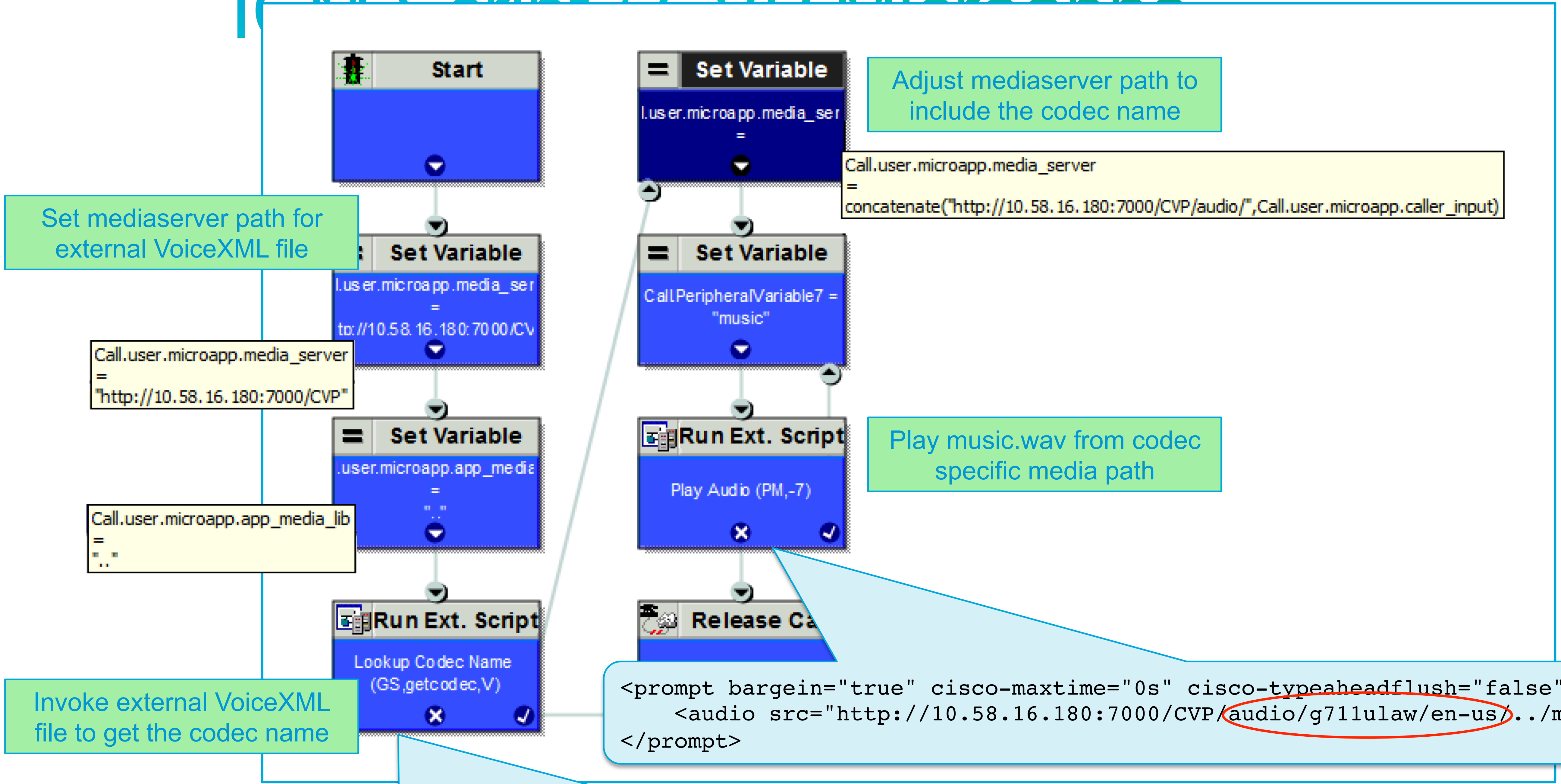
  GS xxx,V microapp references the external VoiceXML file xxx.vxml at the application prompt library media file path

- Modify the media server path to include the codec name component

  For example, http://cvpsvr:7000/CVP/audio/g711ulaw

- Remember to convert your prompts for all the required codecs and locate in the media folder structure

ICM Script / CVP Microapps




```
<prompt bargein="true" cisco-maxtime="0s" cisco-typeaheadflush="false" >
    <audio src="http://10.58.16.180:7000/CVP/audio/g711ulaw/en-us/../music.wav" fetchtimeout="4s" />
</prompt>
```

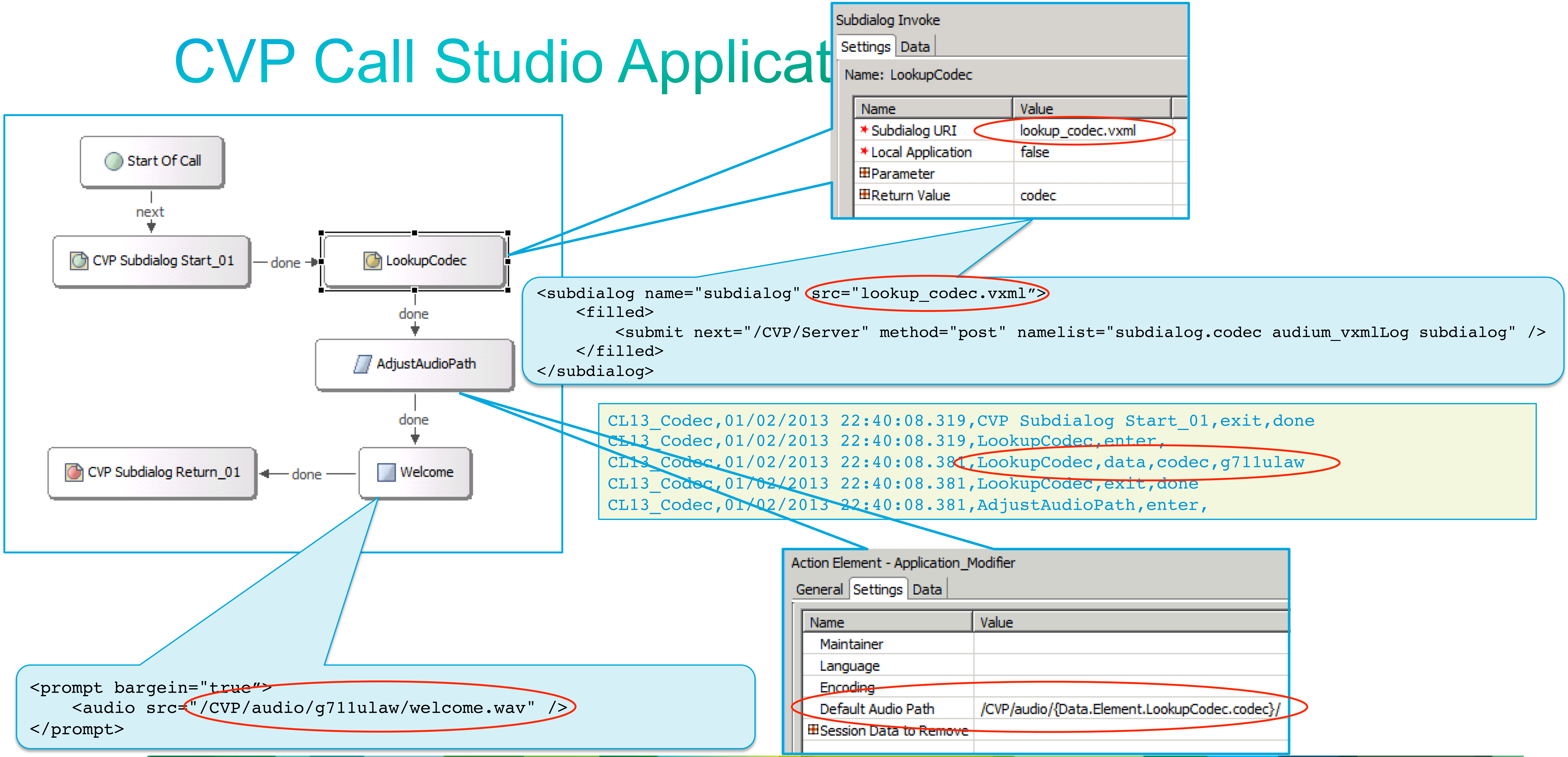
```
<subdialog name="getcodec"  src= "http://10.58.16.180:7000/CVP/en-us/../getcodec.vxml" >
GET /cvp/VBServlet?MSG_TYPE=CALL_RESULT&CALL_ID=621C5F71547411E280EA001B0CFAA768&CALL_SEQ_NUM=1&ERROR_CODE=0&CALLCED=g711ulaw
```

Dynamically Selecting The Prompt Set

Using CVP Call Studio Application

- Invoke external VoiceXML using a Subdialog Invoke element to retrieve the codec name
  - Typically locate VoiceXML file in folder Cisco/CVP/Tomcat/webapps/CVP
- Use an Application Modifier element to modify the default audio path and append the codec name to it
- Play audio files as required (from the correct folder)
- Alternatively, could build a custom voice element using Java
  - Generate the VoiceXML to retrieve the codec
  - Update the default audio path to include the codec name

CVP Call Studio Applicat



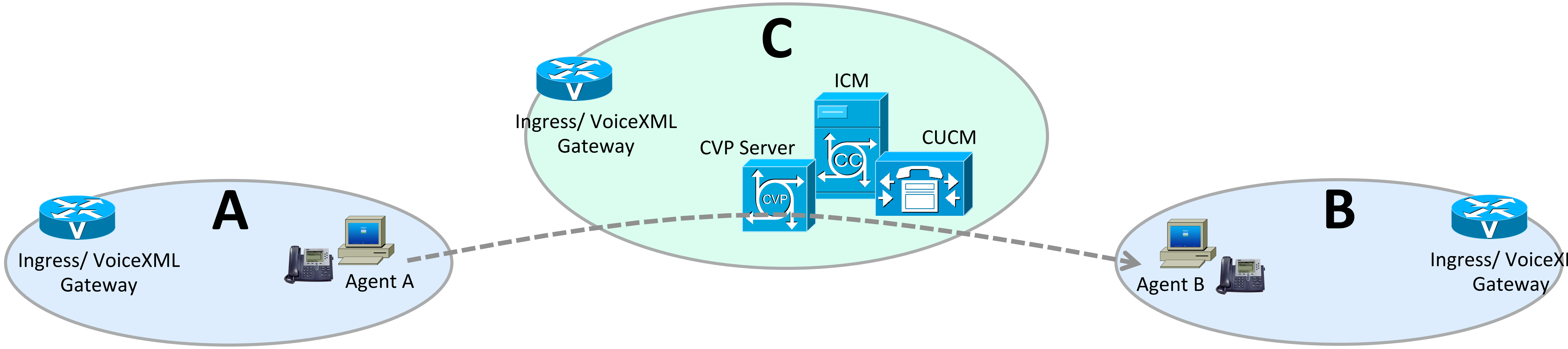
cisco

Correct Codec Selection In Multi-Region Scenarios

Use Case / Challenge

Codec Selection Across Regions

- Consider the architecture



Calls originating at Region A transferred by CVP to Region B

G.711 required for calls within the region

G.729 required for calls between regions

Didn't CVP 8.x Locations Awareness Fix This?

So, What's The Problem?

- CVP sits in the signalling path between calling and called user agents
- CVP receives originating endpoint location from UCM

```
INVITE sip:8808@10.58.16.180:5060 SIP/2.0
From: <sip:4708@10.58.16.175>;tag=407506~0ae7cfdd-e991-41e8-80de-0f6cf4701aa2-20499468
To: <sip:8808@10.58.16.180>
User-Agent: Cisco-CUCM9.0
Call-Info: <urn:x-cisco-remotecc:callinfo>;x-cisco-loc-id=0b8c38fb-d63c-646b-7b79-b71c62ec64da;x-cisco-loc-name=London; ...
```

- CVP sends originating endpoint location and user agent IP address to UCM

```
INVITE sip:4441@10.58.16.175;transport=udp SIP/2.0
From: 4708 <sip:4708@10.58.16.180:5060>;tag=dse9c1f888
To: <sip:4441@10.58.16.175;transport=udp>
User-Agent: CVP 9.0 (1) Build-670
Call-Info: <sip:10.58.16.175:5060>;purpose=x-cisco-origIP
Call-Info: <urn:x-cisco-remotecc:callinfo>;x-cisco-loc-id=0b8c38fb-d63c-646b-7b79-b71c62ec64da;x-cisco-loc-name=London; ...
```

So, What's The Problem?

Didn't CVP 8.x Locations Awareness Fix This?

- UCM uses the Call-Info field location information
  - Knows calling and called endpoint locations
  - Performs correct bandwidth pegging

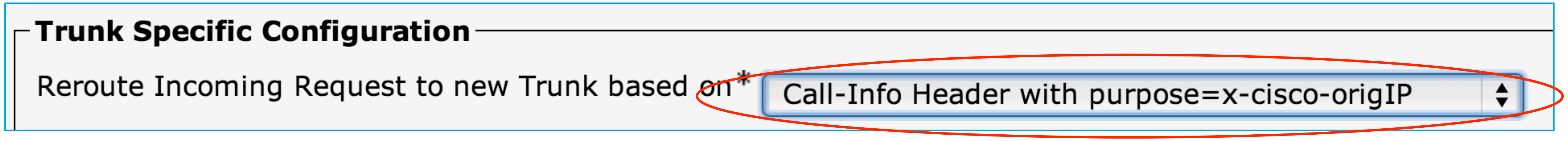
- Successfully addresses locations-based CAC when CVP is present
  - But codec selection is still based on CVP to UCM SIP trunk region

Problem: UCM needs to know/use the calling endpoint region for calls received from CVP

No Problem With Gateways

Ingress Gateway → CVP → Agent Phone

- Originating IP address enables a fix-up if the call is from a gateway
  - Use this option on SIP trunk profile



- Selects an alternative SIP trunk based on x-cisco-origIP
  - Call is processed as though it came directly from the gateway

- Both CAC and codec selection are handled correctly

Agent A → CVP → Agent B

Problem Happens When Agents Call

- If call is from an IP phone, Call-Info x-cisco-origIP is the UCM address
  - Unlike calls originating from gateways, it doesn't help us at all

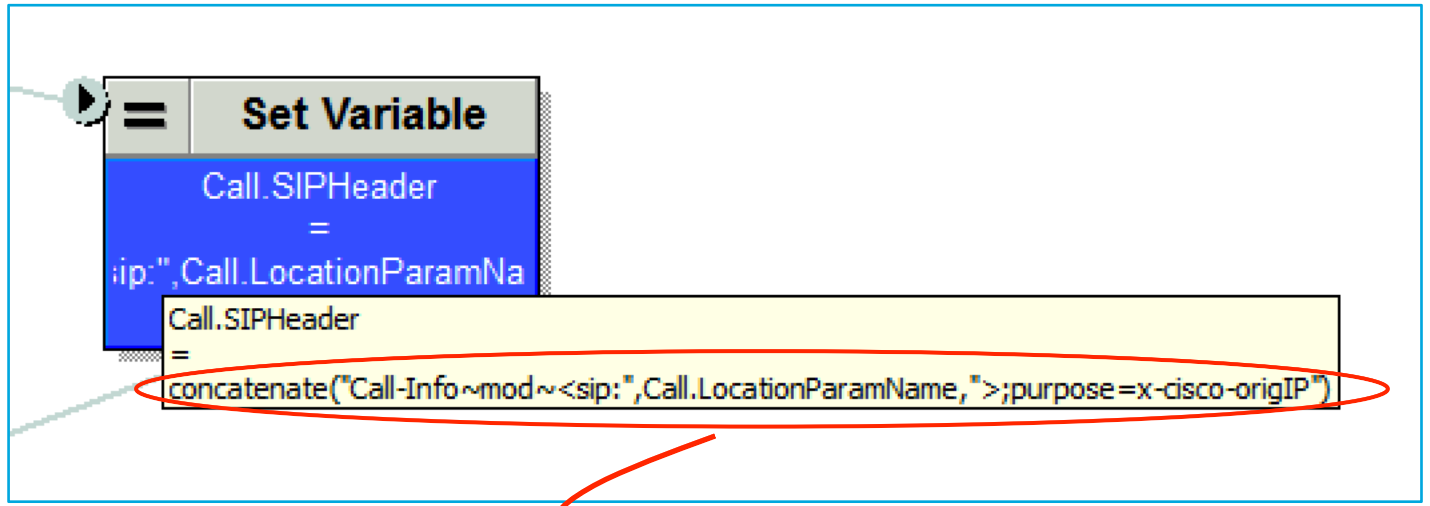
- Call-Info x-cisco-loc-name contains the IP phone's location
  - UCM can't use that to determine the codec, it needs the region
  - UCM has no mechanism currently to derive region from the Call-Info header even if it was present

Workaround

Forcing The Calling Party Region

1. CVP/ICM knows the calling party location
   - Either receives it on incoming INVITE from UCM
   - Or, mapped from originating IP by CVP using its own location table

2. ICM script sets Call-Info SIP header x-cisco-origIP to that caller **location name**

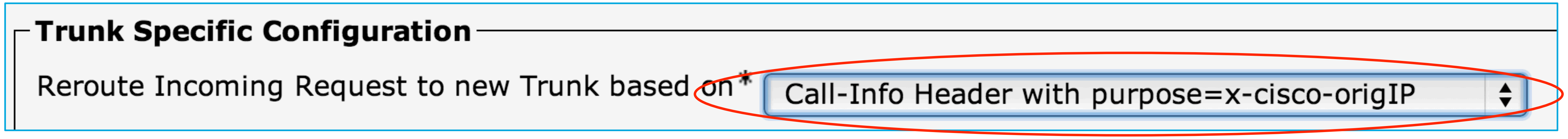



```
Request-Line: INVITE sip:4441@10.58.16.175;transport=udp SIP/2.0
From: 4708 <sip:4708@10.58.16.180:5060>;tag=dsd5048ee4
To: <sip:4441@10.58.16.175;transport=udp>
User-Agent: CVP 9.0 (1) Build-670
Call-Info: <sip:London>;purpose=x-cisco-origIP
```

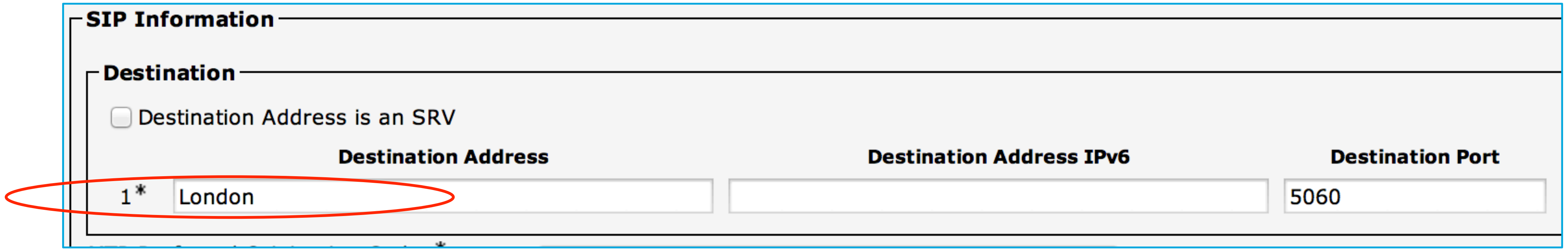
Workaround

Forcing The Calling Party Region

3. UCM does reroute to new trunk based on originating IP



4. Locates a dummy trunk with its IP address matching the **location name** (contents of the Call-Info header)



Workaround

Forcing The Calling Party Region

5. UCM uses the trunk device pool / region as normal to select the correct codec for the call

6. Needs a dummy trunk to be added for each calling party location

		Name	Description	Calling Search Space	Device Pool	Route Pattern	Partition	Route Group	Priority	Trunk Type
	SIP	Backwell_Dummy_Trunk			Paul_Lab					SIP Trunk
	SIP	London_Dummy_Trunk			London_Lab					SIP Trunk
	SIP	Rome_Dummy_Trunk			Rome_Lab					SIP Trunk

7. Same approach is compatible with calls originating at gateways

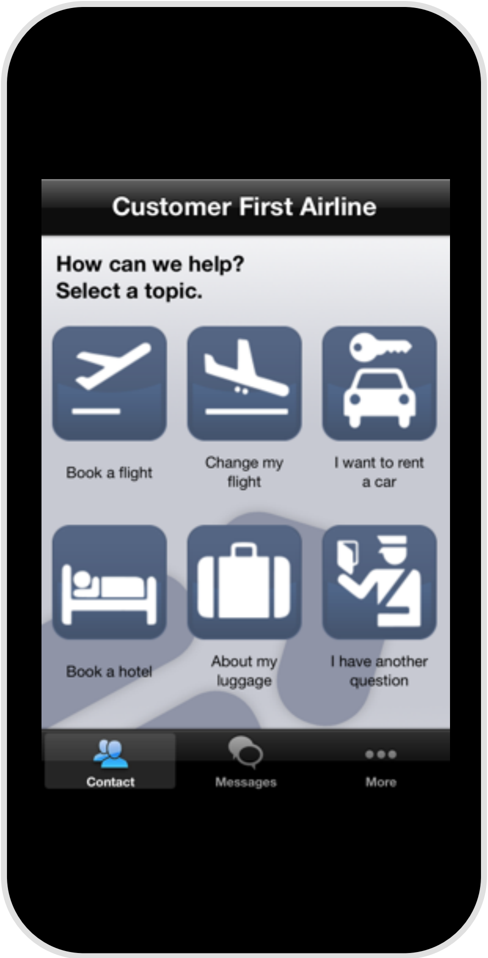
Common dummy trunk per location rather than one per gateway IP address

Thank you

Mobile Device App Integration

Mobile Contact Use Cases

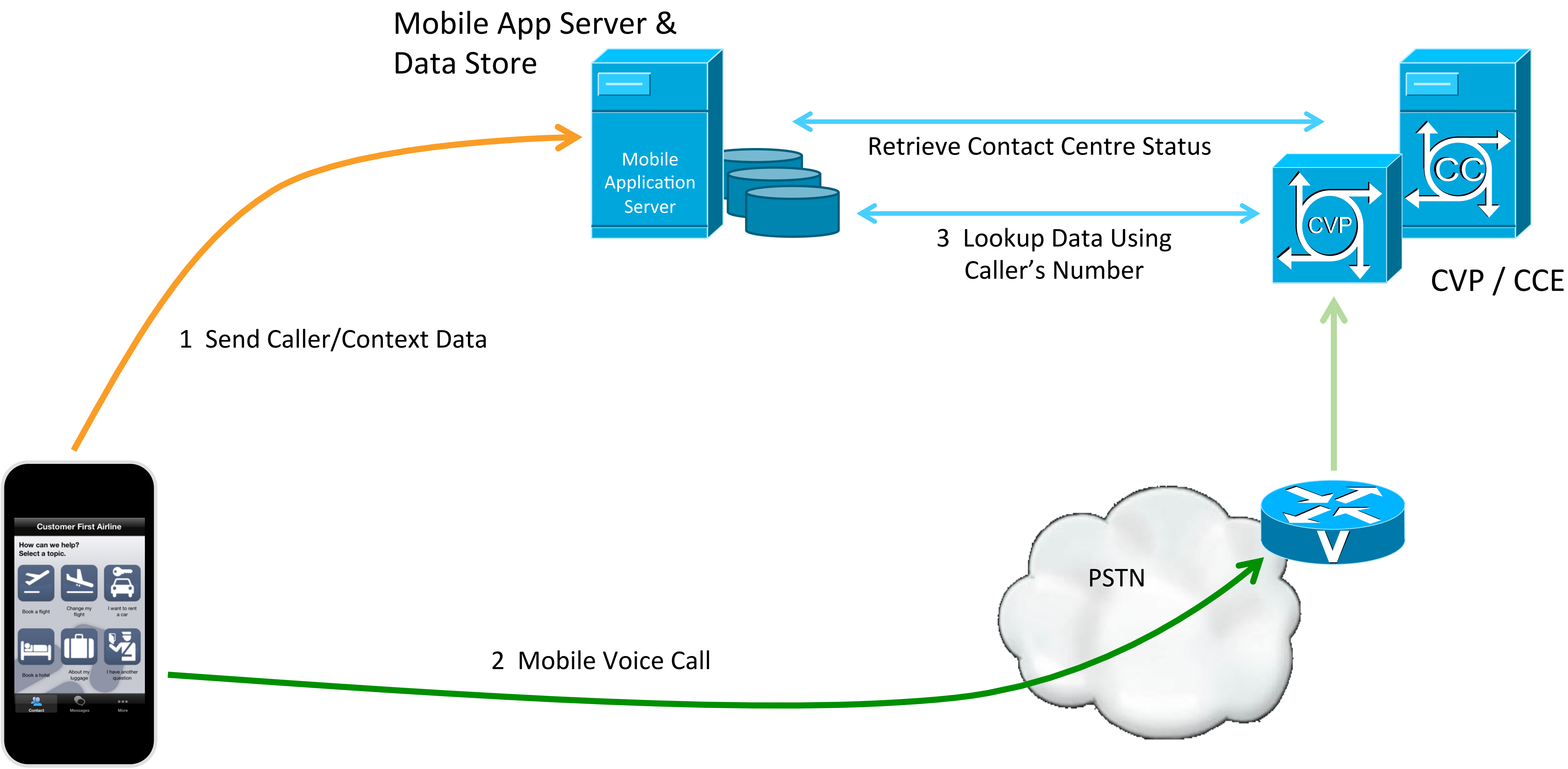
- Call customer services from a mobile device
- Pass caller/context information, such as:
  - Location, Contact reason, Name, Account Reference
  - Section of mobile app at which contact request made
  - Bar code, photo
- Eliminate IVR interaction



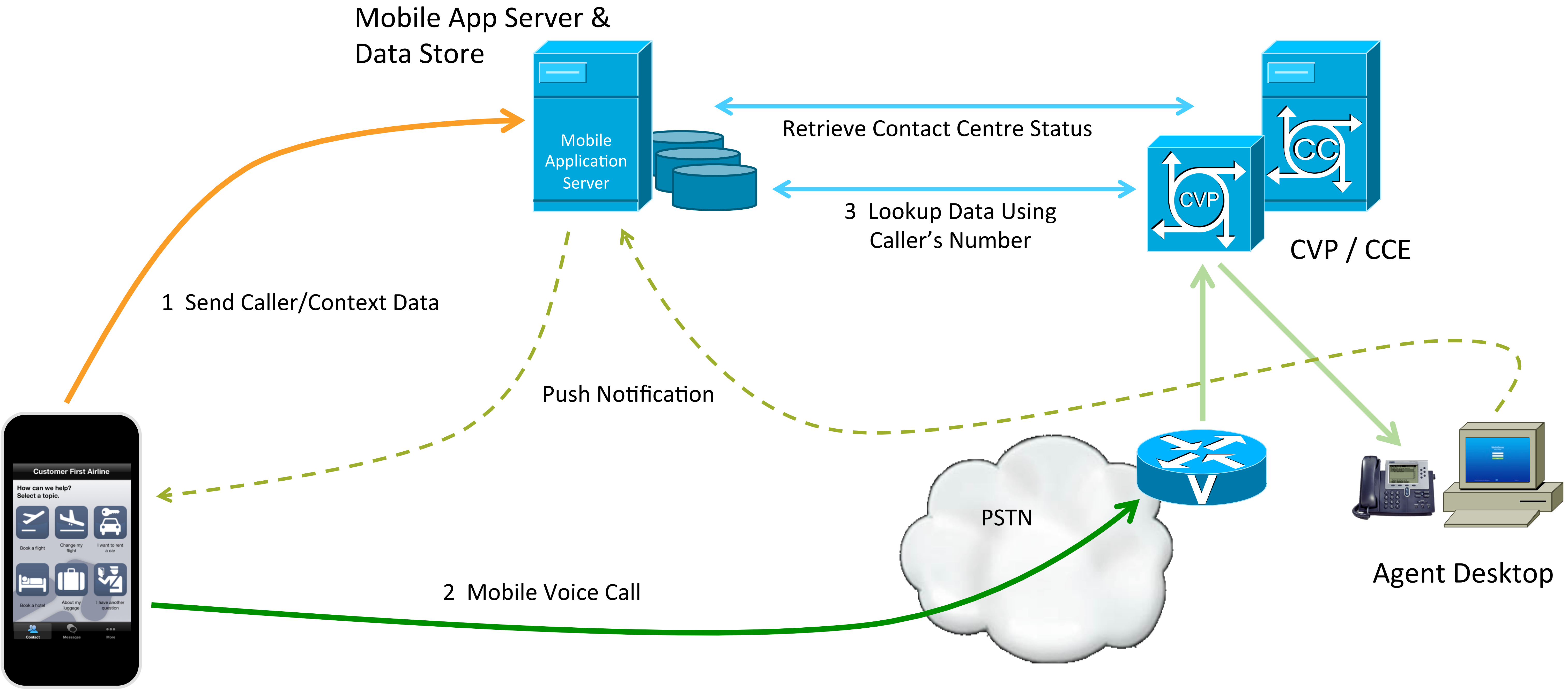
Mobile Contact Use Cases

- Alternatively, request a callback
- Make contact request via data path
  - Sit in virtual queue watching progress
  - Either make call or receive callback when agent becomes available
- Agent or IVR pushes information back to mobile device
- Visual IVR – caller navigates dynamically generated dialogue using IM chat style interface
- Mobile chat with agent

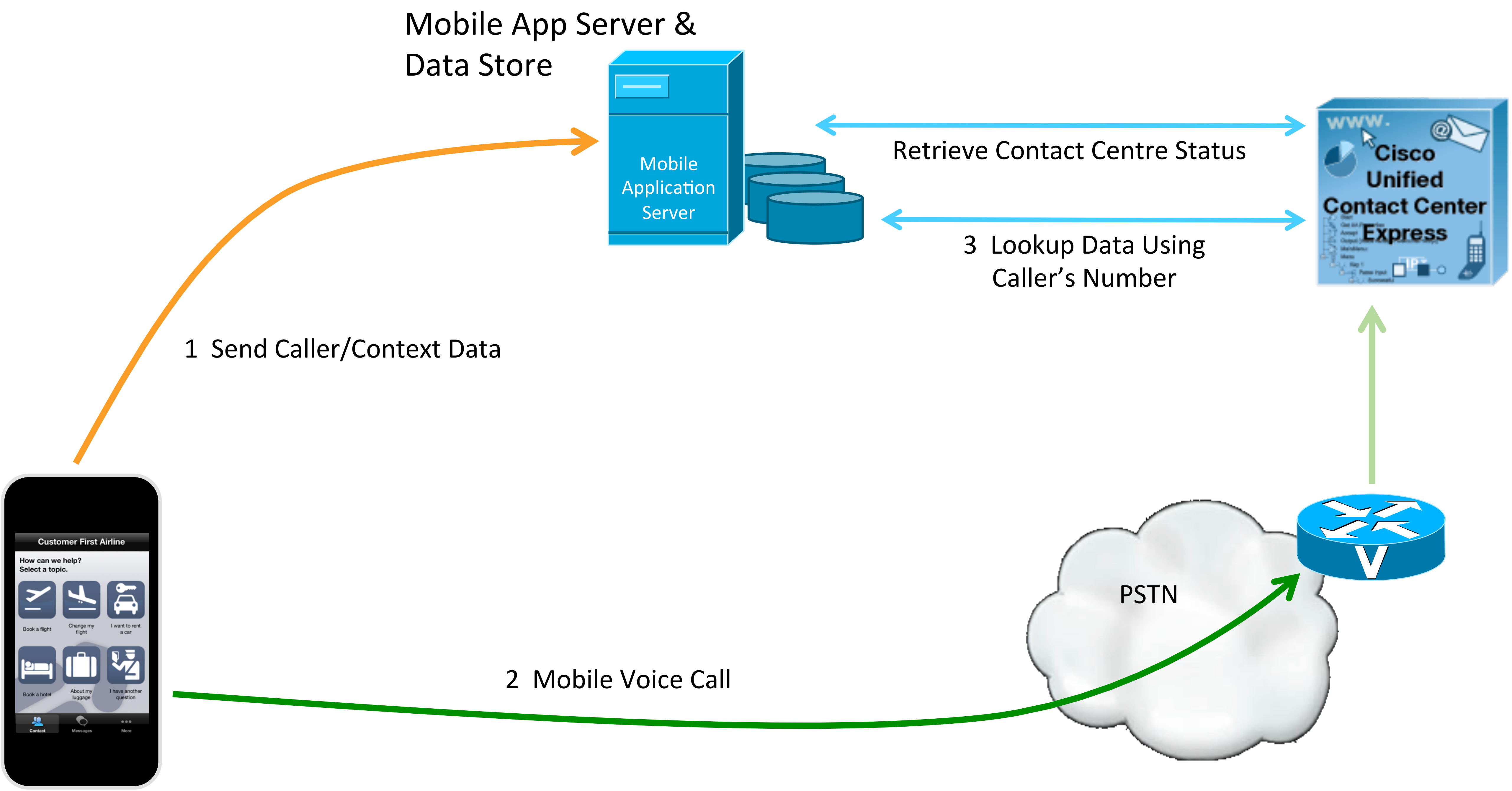
Mobile Voice Call With Data Passing



Mobile Voice Call With Data Passing



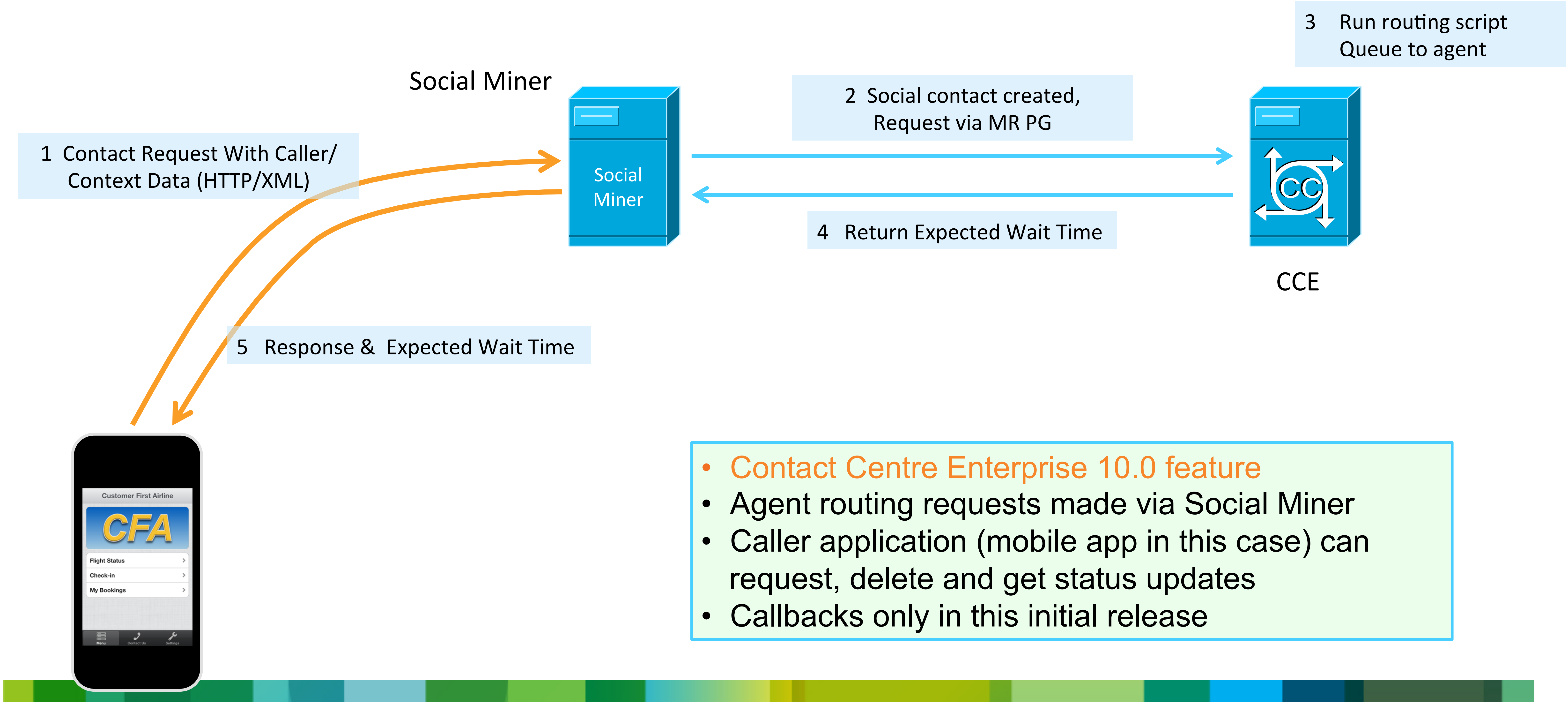
Same Approach Will Work With CCX



Thank you

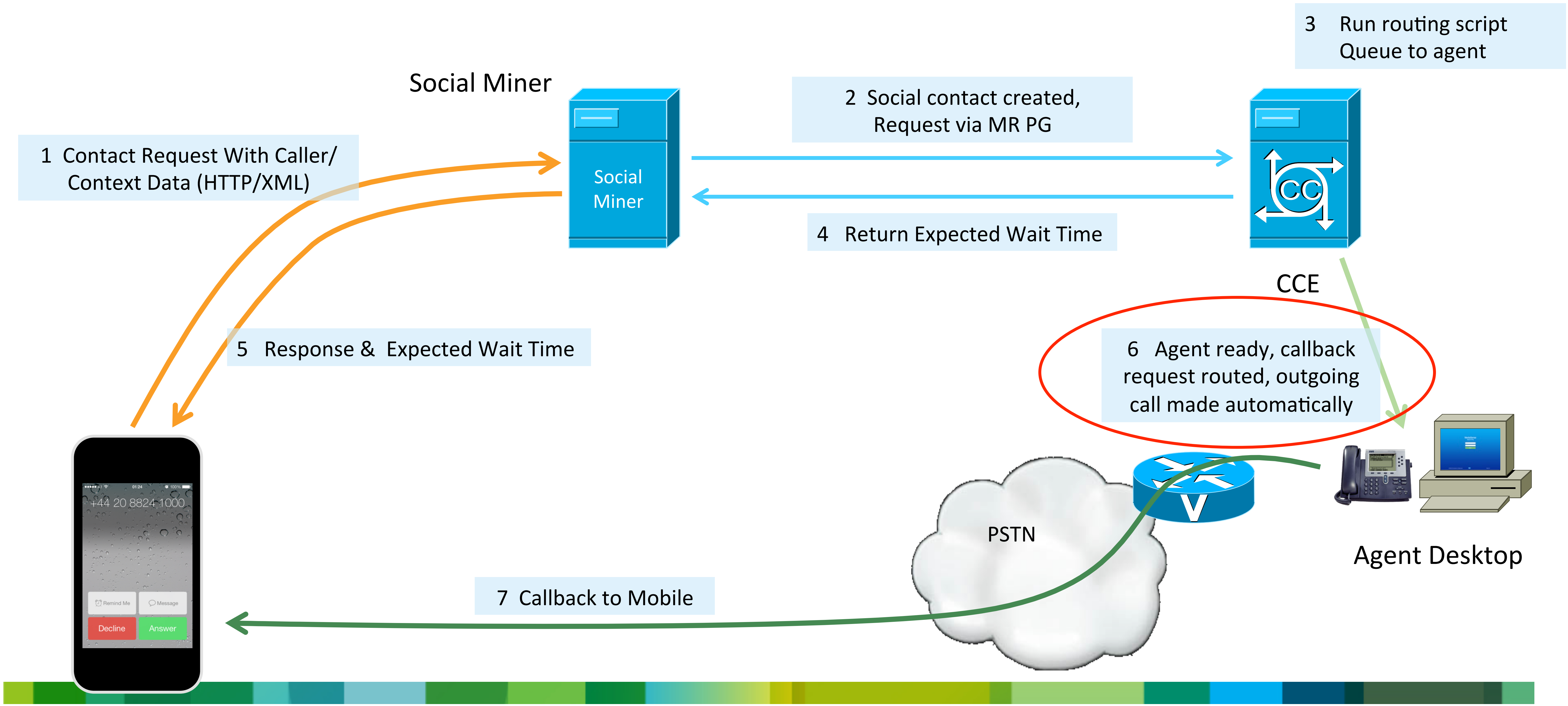
Agent Request API For Callbacks

Why Not Queue Request Without The Call?



- Contact Centre Enterprise 10.0 feature
- Agent routing requests made via Social Miner
- Caller application (mobile app in this case) can request, delete and get status updates
- Callbacks only in this initial release

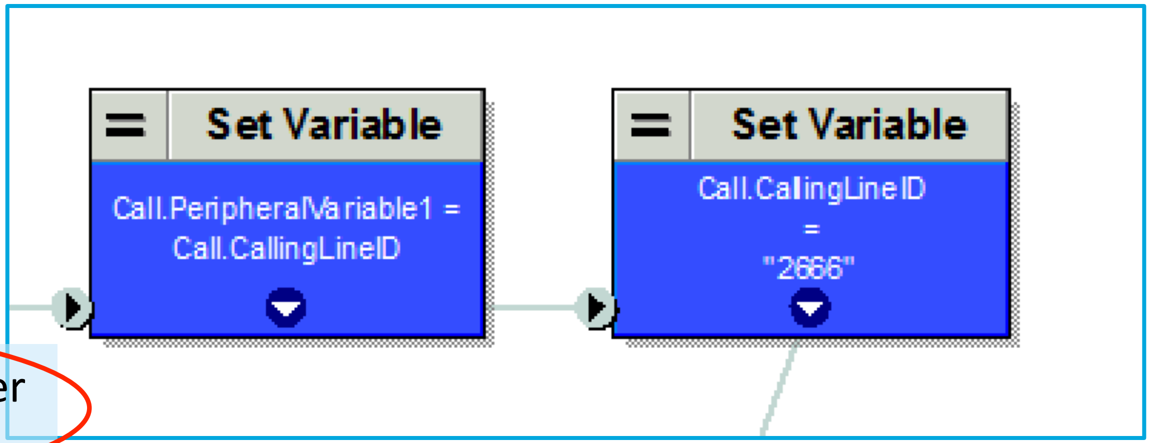
Why Not Queue Request Without The Call?



What If I Need a Preview Mode Callback?

- There is a way to intercept the callback
- When the initial request via the API is processed
  - In the ICM routing script, set the callback number to a route point
  - Store the actual callback number in another call variable

http://10.52.200.163/ccp/callback/feed/100000?name=Paul&title=Test_1&mediaAddress=447740220066



CallingLineID contains the number to which the callback will be made, Route Point DN 2666 in this example

Callback number in mediaAddress parameter is passed to the ICM as CallingLineID

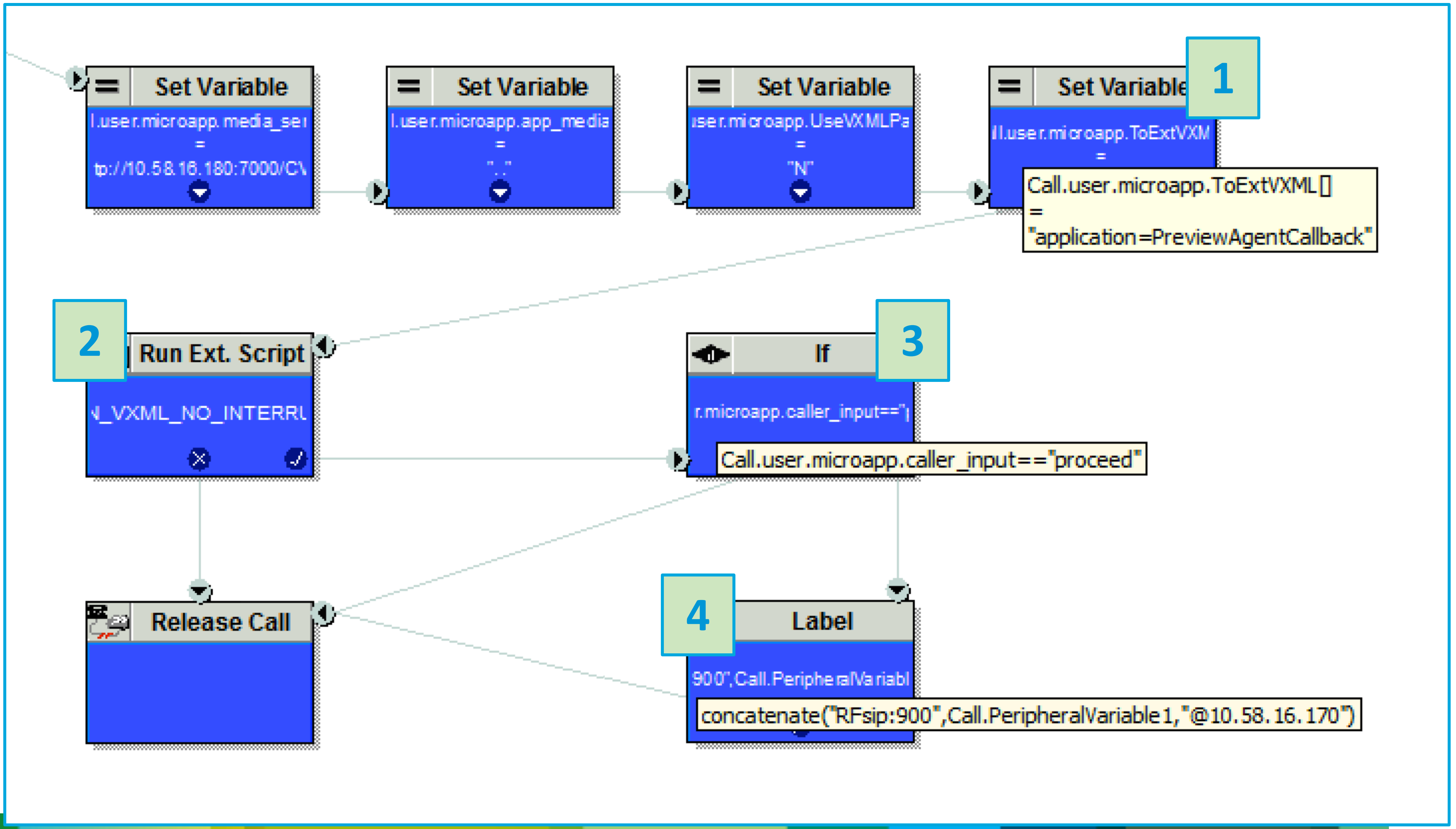
- Save the actual callback number
- And, use a route point instead

What If I Need a Preview Mode Callback?

- When the request is routed to the agent

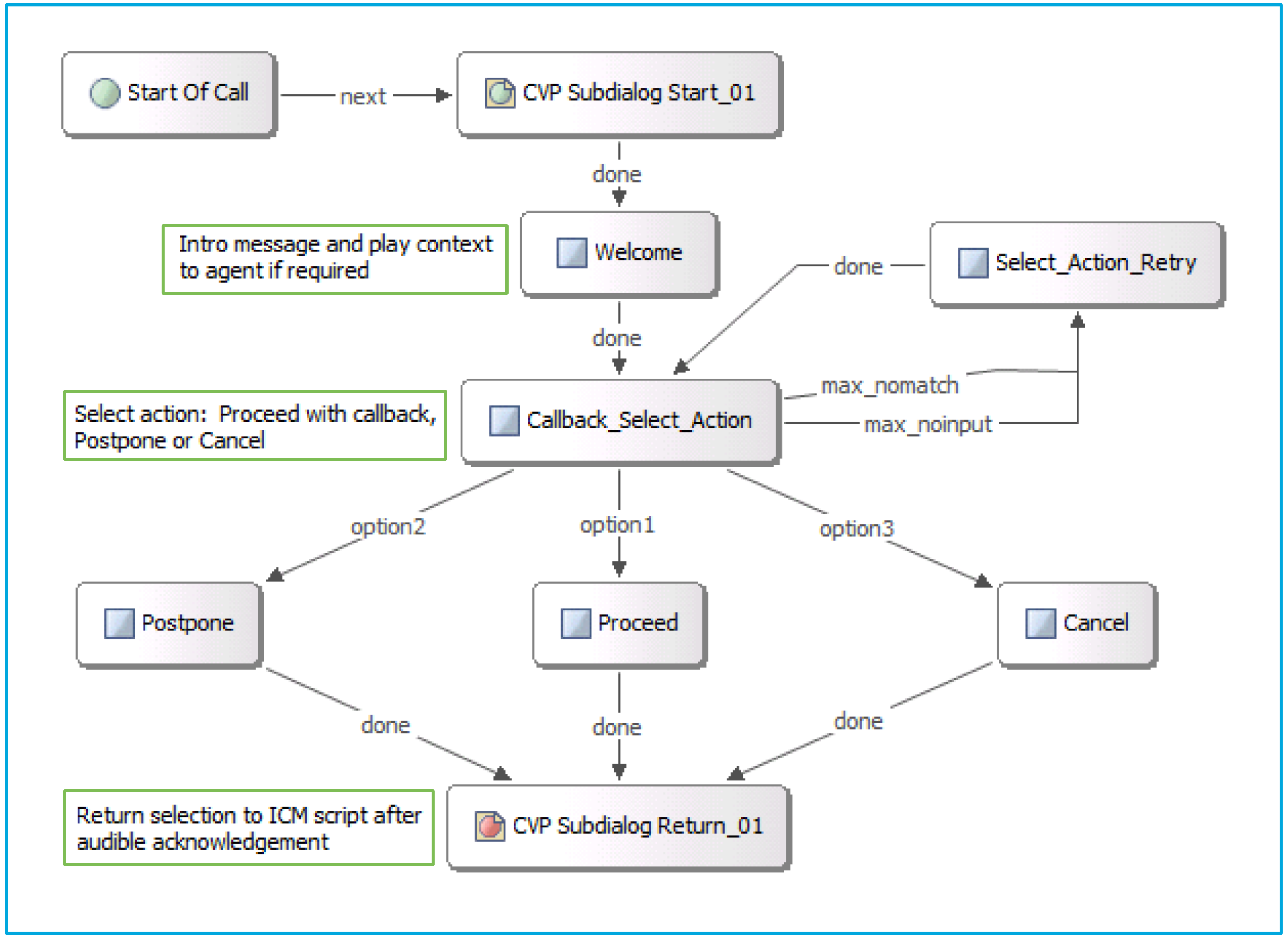
  Callback is established immediately to the route point and invokes script

1. Sends the call to CVP to run a simple IVR application
2. Prompts the agent to Proceed, Postpone, Cancel
3. Agent opts to proceed
4. SIP REFER sent back to CUCM to transfer the agent to the actual callback number

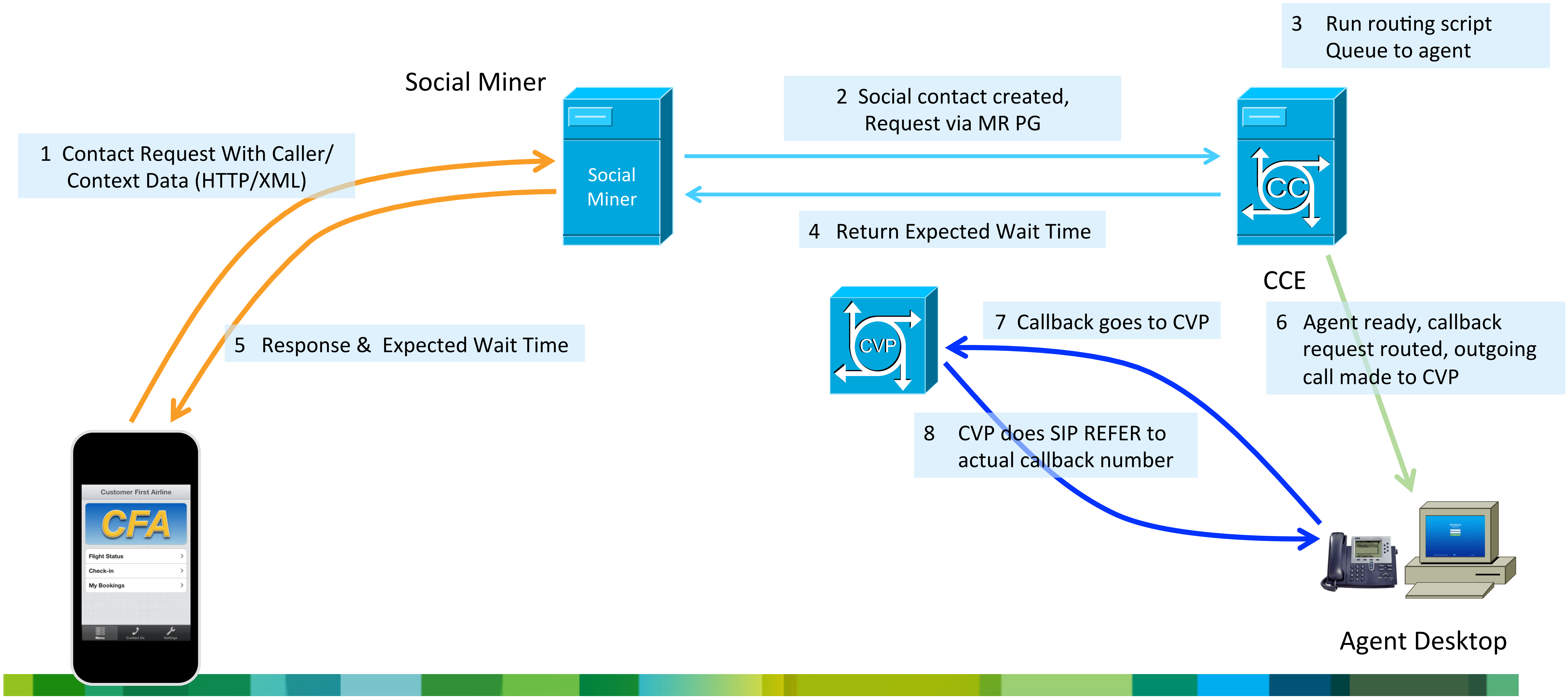


Preview Intercept – Sample Call Studio Application

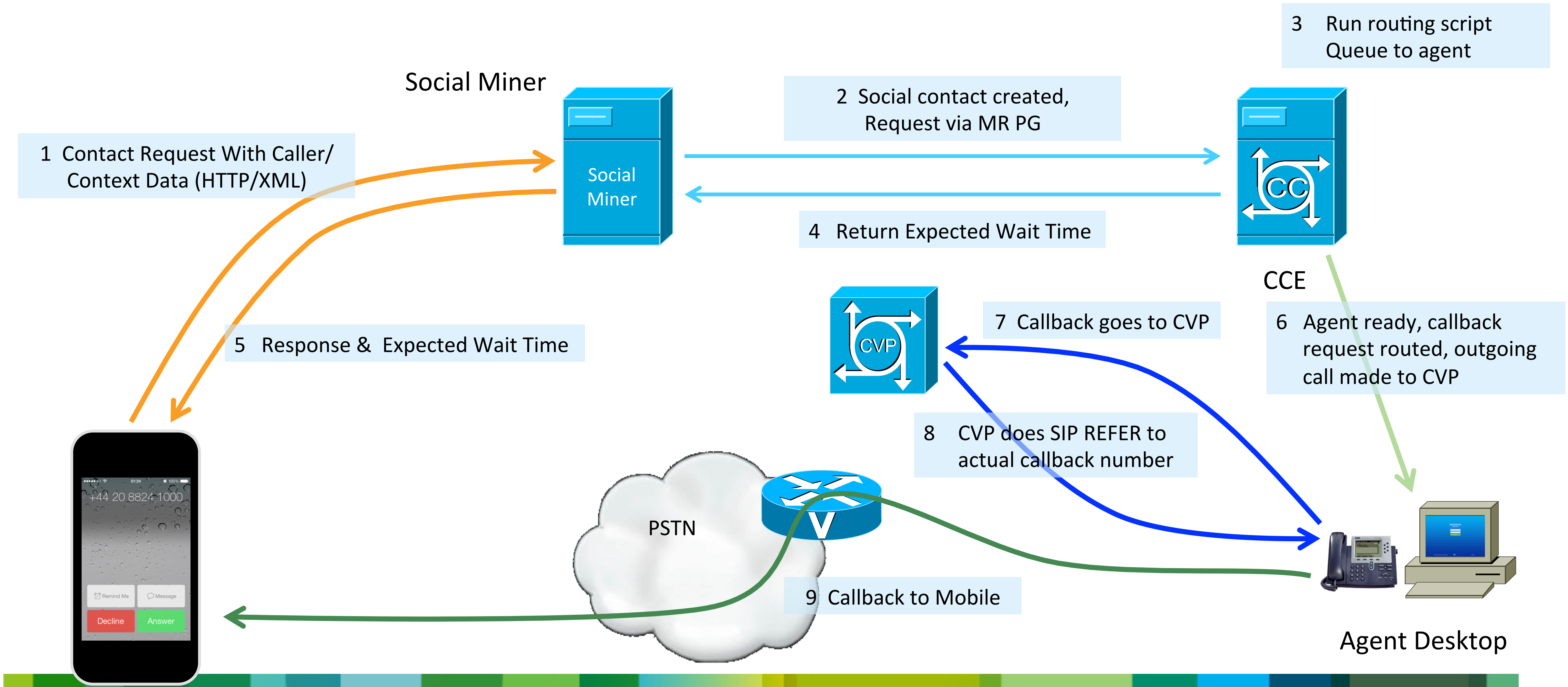
- Play welcome / context to agent
- Agent selects whether to accept, cancel or delay the callback
- Provide audible acknowledgement to agent
- Return the agent selection to the ICM script to perform REFER transfer or continue queuing



Preview Mode Callback



Preview Mode Callback



What About Other Request Types?

- Same intercept mechanism could be used to handle other media
- For example,

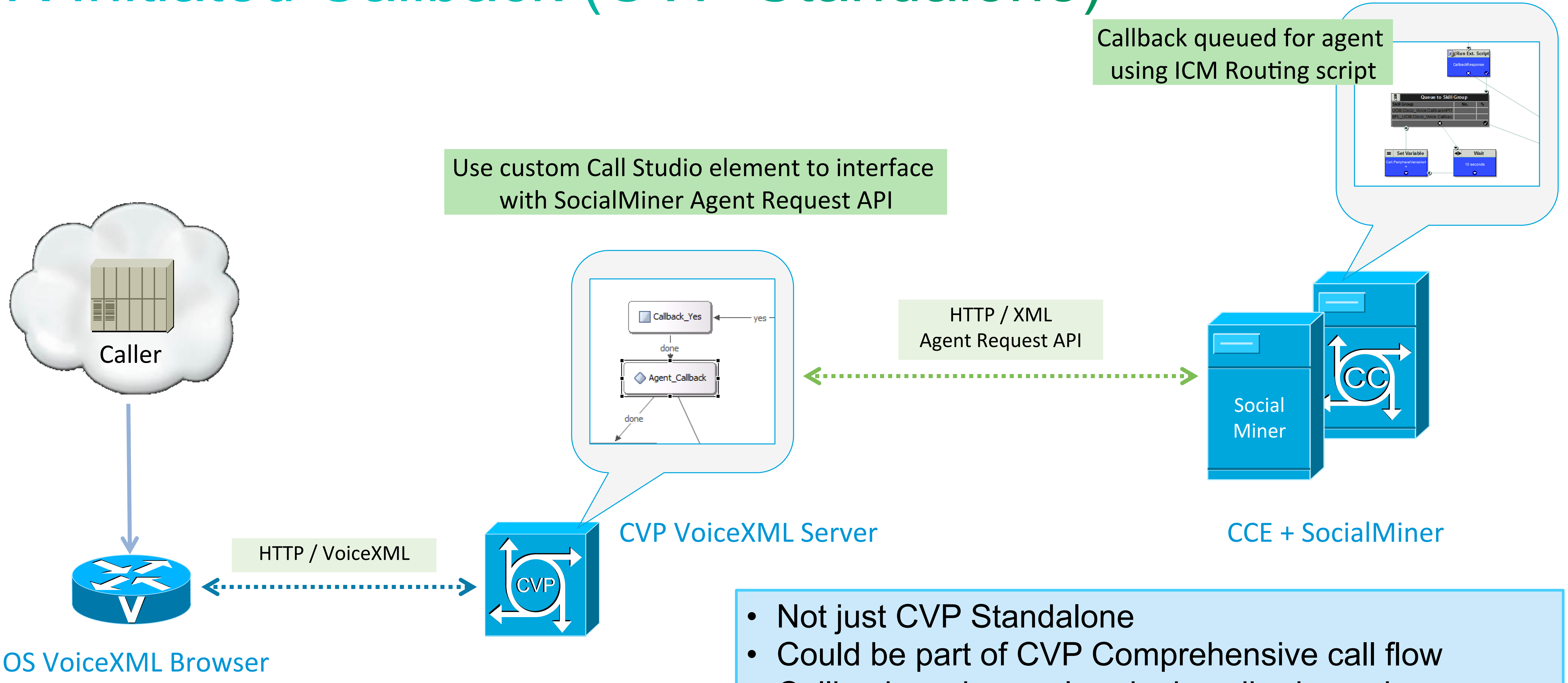
  Incoming SMS or Chat triggers Agent Request API

  Request routed to agent with message detail and request channel type

  Callback to IVR to intercept and play relevant whisper message to agent
    (Could play caller message to agent if callback was queued by IVR)

  Desktop displays customer data / SMS or Chat contact handling gadgets

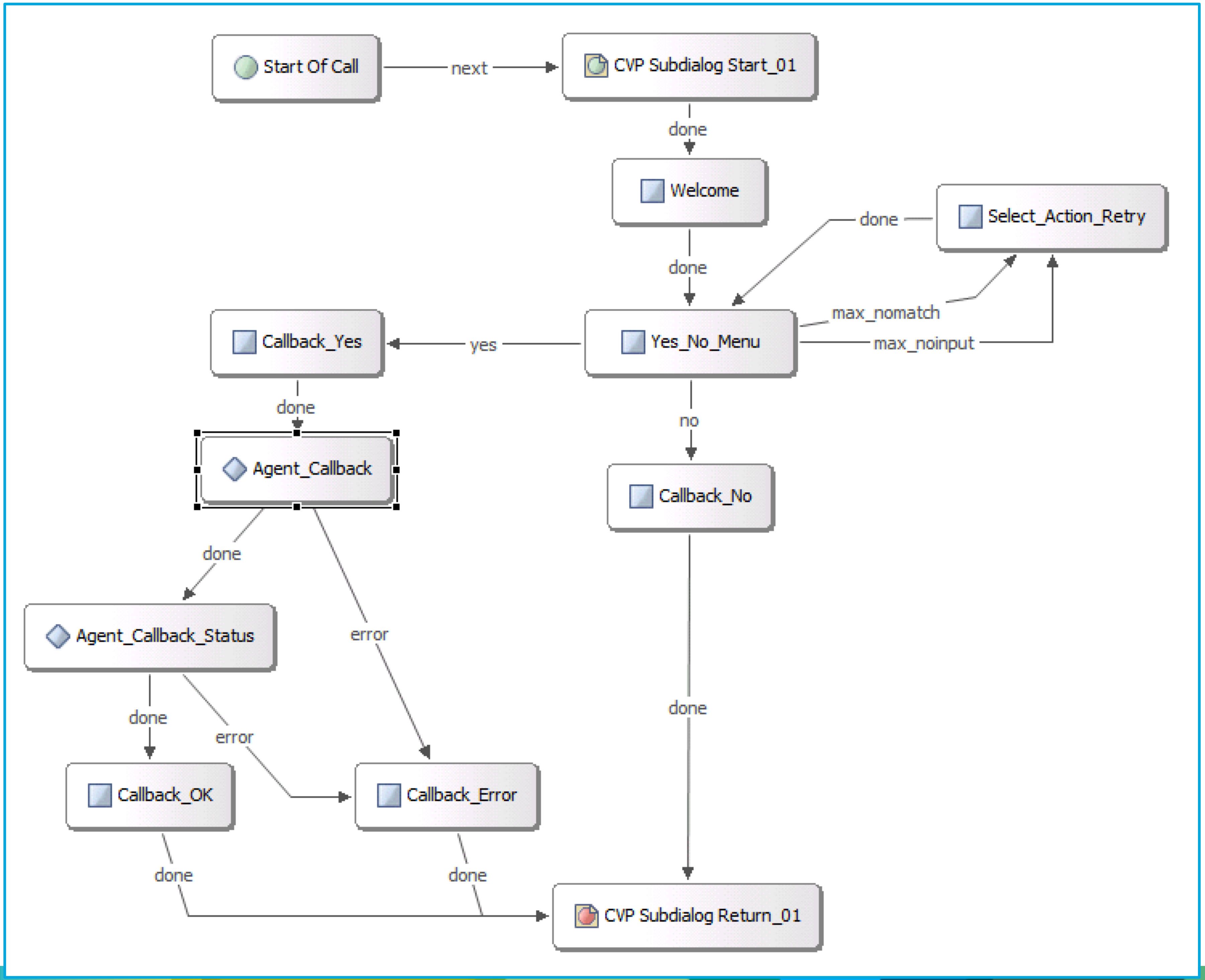
IVR Initiated Callback (CVP Standalone)



- Not just CVP Standalone
- Could be part of CVP Comprehensive call flow
- Callback works as already described, preview or immediate

IVR Initiated Callback (CVP Standalone)

- Play welcome to caller
- Determine whether callback is required
- Make callback request
- Check callback status
- Play outcome to the caller and hangup call or continue dialogue as required



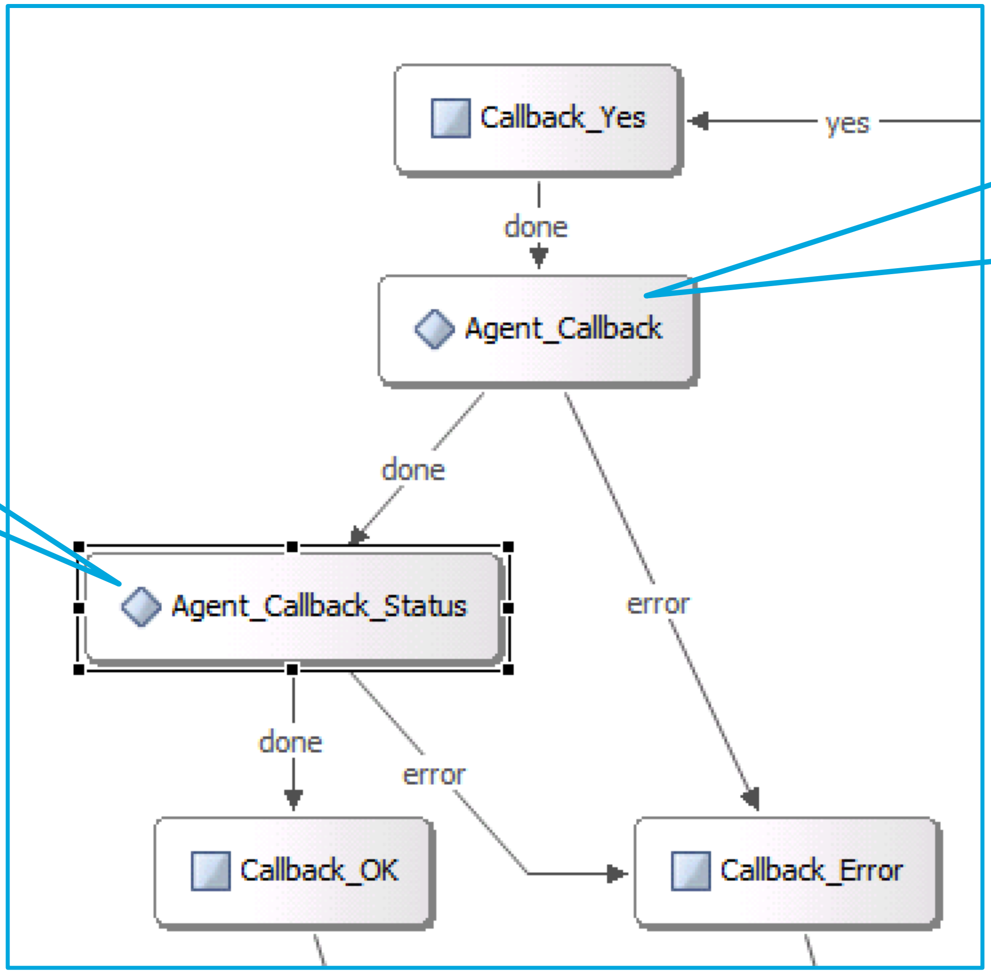
Agent Request API From Call Studio

Decision Element - Agent_Callback_Request

General | Settings | Data

Name	Value
* Operation	STATUS
Accept Certificates	false
* Callback Reference URL	{Data.Element.Agent_Callback.reference}

- Custom decision element
- CREATE, STATUS or DELETE
- Data sent as
    - Call Variables
    - ECC Variables
    - Tags
- Tags determine script selection



Decision Element - Agent_Callback_Request

General | Settings | Data

Name	Value
* Operation	CREATE
* HTTP(S)	HTTPS
Accept Certificates	false
* Social Miner Host	10.52.200.163
Social Miner Port	
* Social Miner Feed ID	100000
Request Title	
Request Name	
Request Description	Test callback with variables
* Callback Number	900{CallData.ANI}
Call Variable 1	1
Call Variable 2	2
Call Variable 3	3
Call Variable 4	4
Call Variable 5	5
Call Variable 6	6
Call Variable 7	7
Call Variable 8	8
Call Variable 9	9
Call Variable 10	10
⊞Tag	testwithtagsadded
⊞Tag	rometestcall
⊞Tag	poclab
⊞ECC variable	user.fruit=banana
⊞ECC variable	user.microapp.caller_input=test...

- Create request returns a Reference URL
- Used in subsequent STATUS or DELETE operations

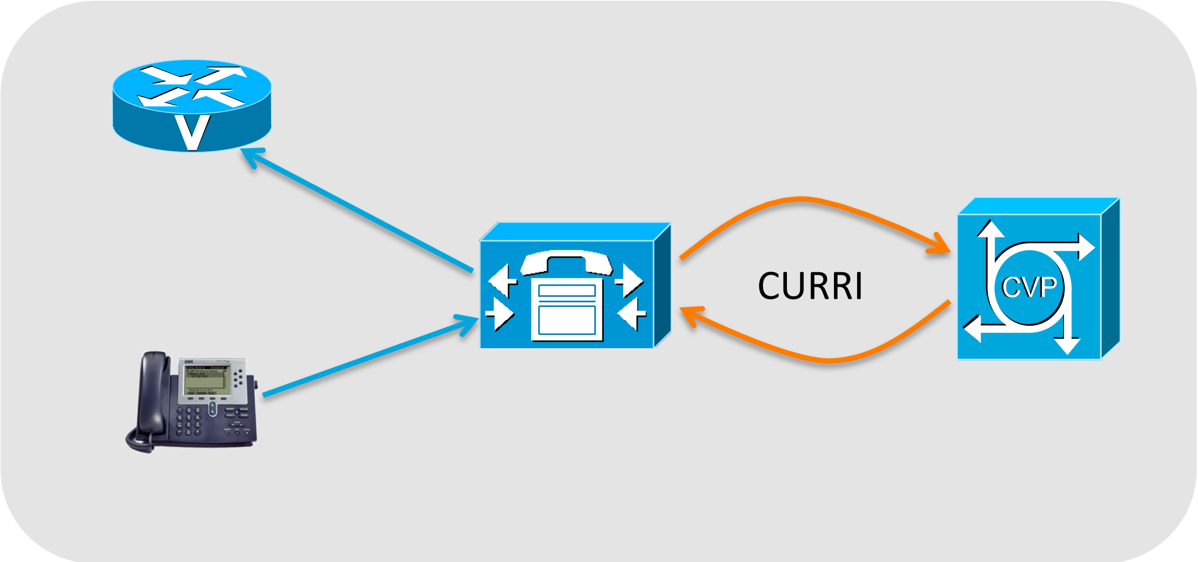
Thank you

CURRI And The Contact Centre

Paul Tindall
@tindallpaul

November 2014

CURRI – What Is It?



- Cisco Unified Routing Rules Interface
  - Simple HTTP / XML request/response API
- CUCM makes External Call Control (ECC) request to external server
  - Triggered via dial plan – DNs, Translation Patterns, Route Patterns
  - Passes calling/called party signalling information
- External server sends back ECC response, containing:
  - Policy decision – allow or deny call
  - Modified signalling – calling/called numbering and display information
  - Optional announcement for the caller

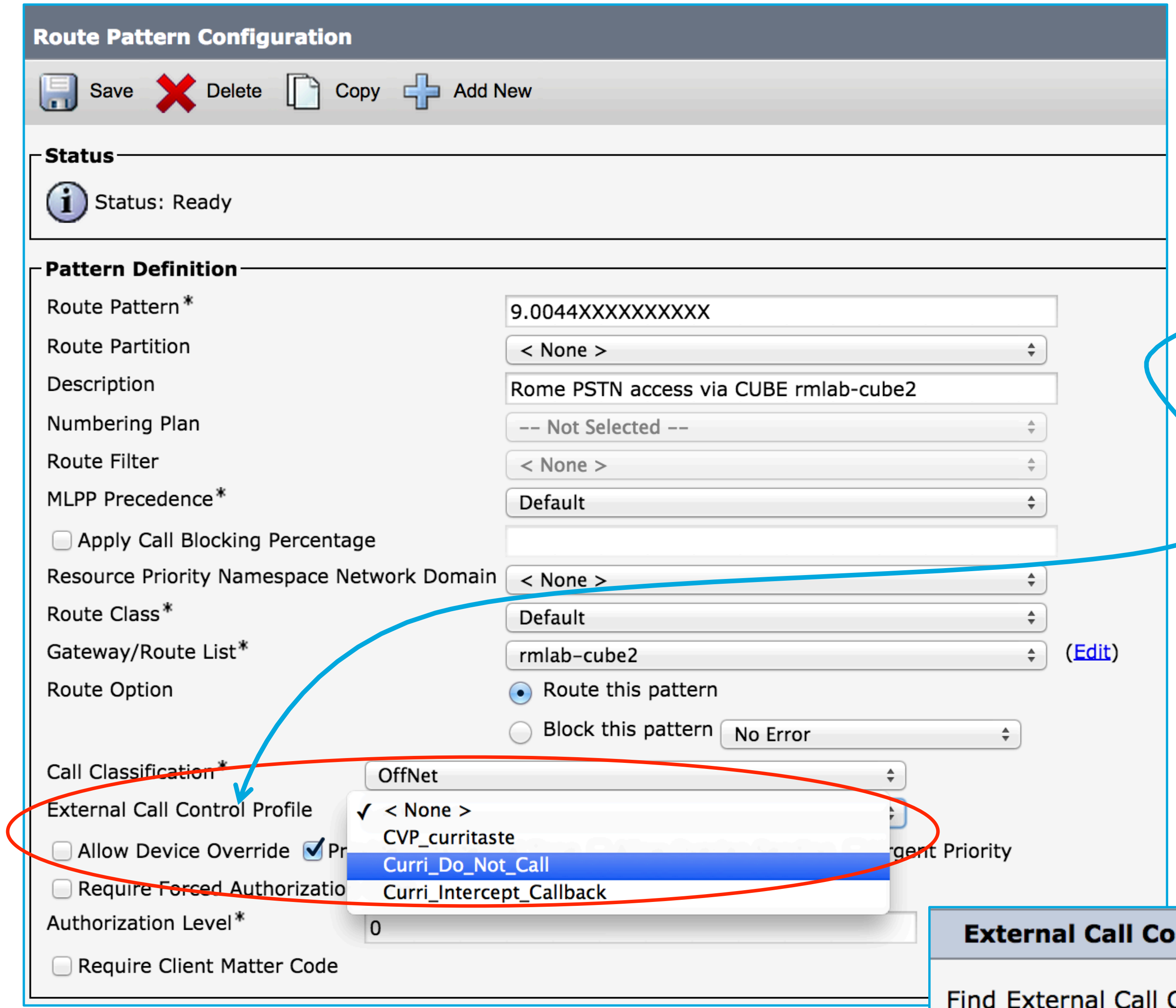
CURRI – Why So Useful?

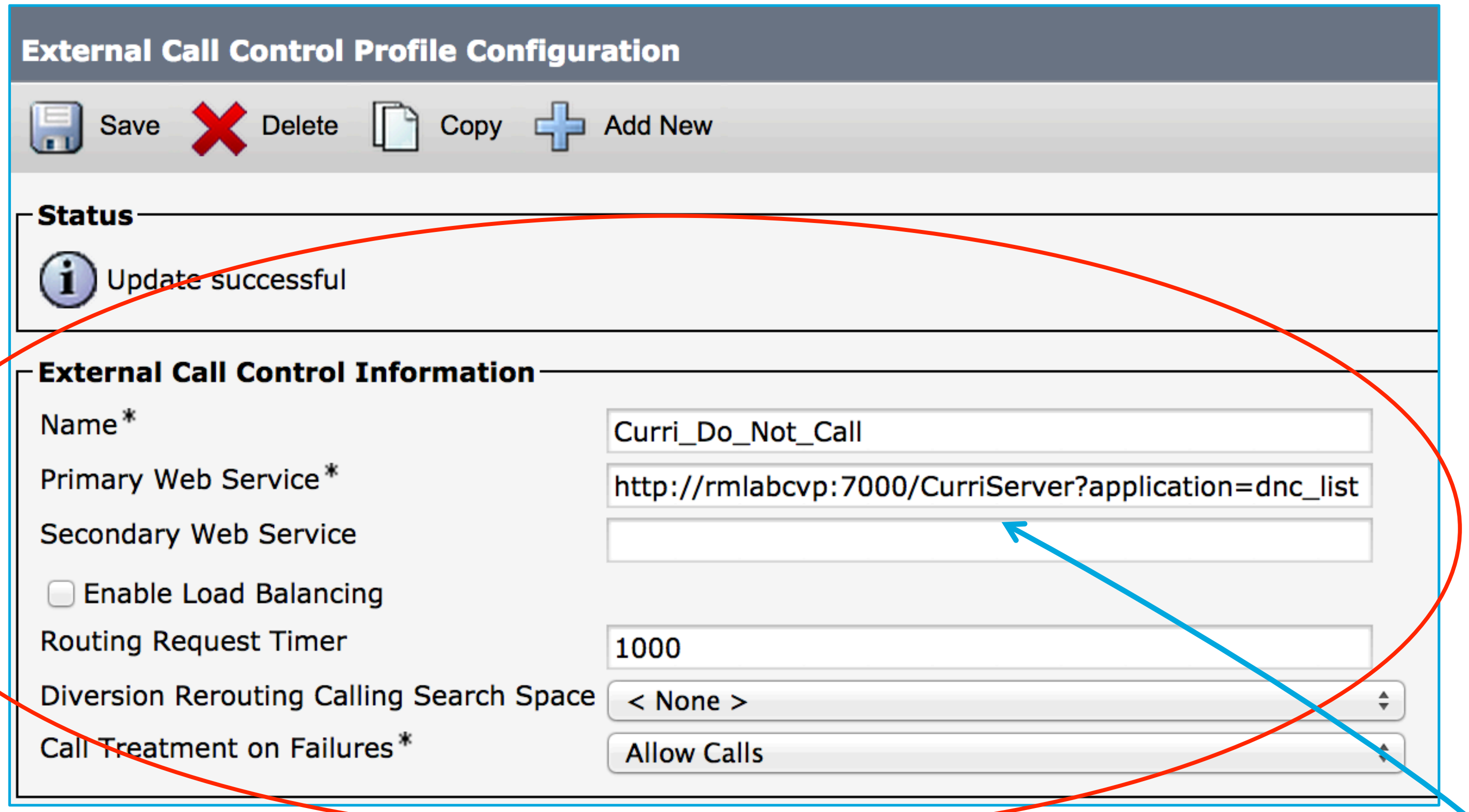
- Applying policy
  - Blacklisting / blocking calls
  - Could be outright block or selective based on caller
  - Perhaps allow call if agent is logged on
- Playing announcements based on calling/called parties
  - Reason for blocking
  - Recorder warning

CURRI – Why So Useful?

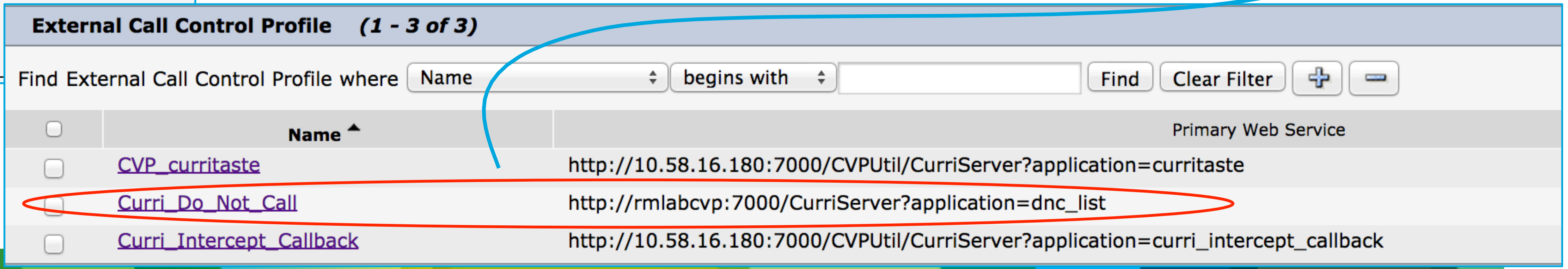
- Modify the signalling
  - Override CLI to provide meaningful callback number
  - Possibly a regional number based on called party
- Meaningful display information
  - Type of call / service
  - Customer name from lookup
  - Override internal system generated info (for example, --CVP_10_5_ ...)

CUCM Configuration





1. Add ECC definition – destination URLs, timeout, fallback action
2. Enable dial plan to trigger ECC requests



What About The Server-Side?

- Could be simple static XML document on a web server

```
<Response>
  <Result>
    <Decision>Deny</Decision>
    <Obligations>
      <Obligation FulfillOn="Deny" obligationId="reject.simple">
        <AttributeAssignment AttributeId="Route:reject.simple">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            <cixml version="1.0"> <reject> <announce identification="VCA_00121"/> </reject> </cixml>
          </AttributeValue>
        </AttributeAssignment>
      </Obligation>
    </Obligations>
  </Result>
</Response>
```

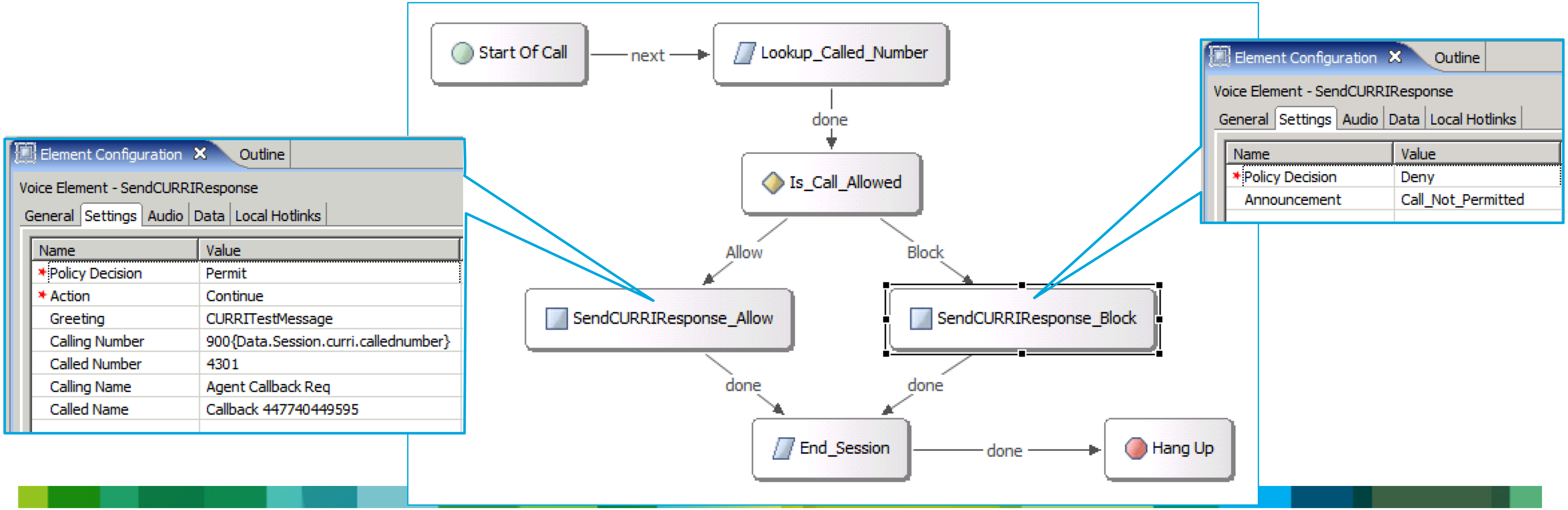
```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Obligations>
      <Obligation FulfillOn="Permit" obligationId="continue.simple">
        <AttributeAssignment AttributeId="Route:continue.simple">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            <cixml version="1.0”><continue><greeting identification="International"/>
            <modify callingNumber="0800400600"/></continue></cixml>
          </AttributeValue>
        </AttributeAssignment>
      </Obligation>
    </Obligations>
  </Result>
</Response>
```

Server-Side Decision Making

- What if more complex policy/treatment business logic?
  - Need a servlet to generate dynamic XML response
- Don't have to resort to coding
  - Use CVP VoiceXML Server to generate CURRI XML response
  - Advantages –
    - Script business logic rather than code
    - Graphical builder
    - Use existing server component
    - Built-in elements for backend integration
    - Add custom elements if needed

CVP As CURRI Server-Side

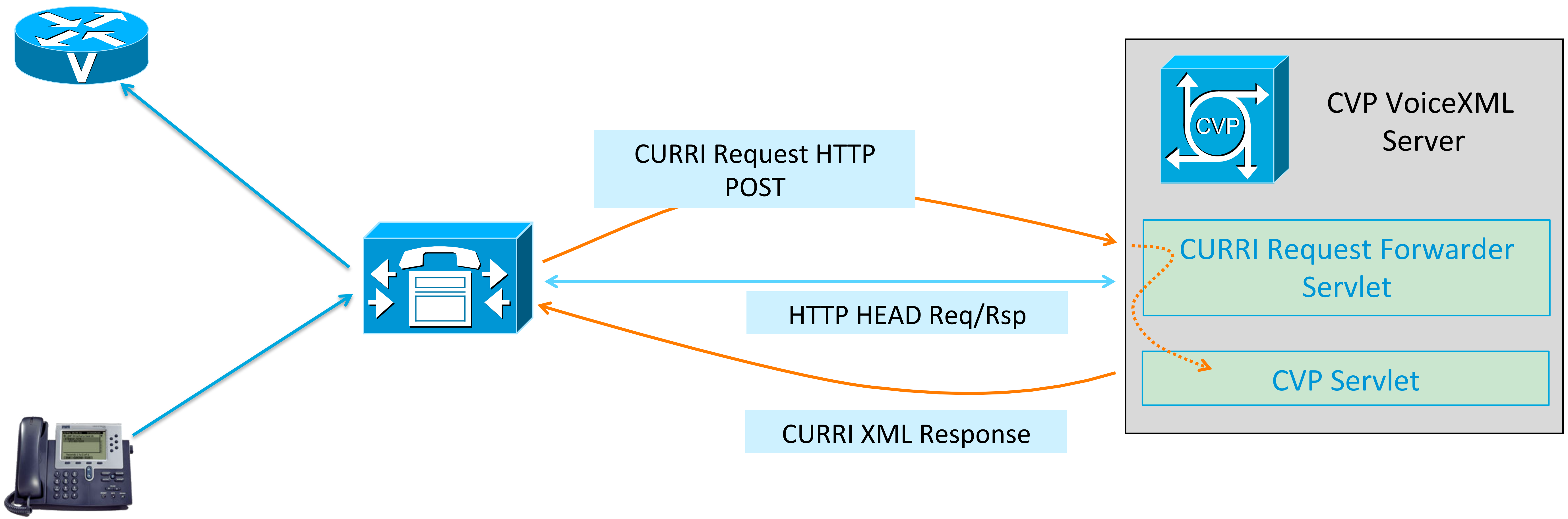
- CVP Call Studio Application
- Custom Element to generate XML output in CURRI response format



Problem: CURRI Request / CVP Parameter Format

- CURRI submits XML block containing call information
- CVP expects key=value pairs (URL params typically)
- Also, CUCM sends HTTP HEAD messages as keep-alives
- Add request forwarder servlet to CVP VoiceXML Tomcat instance
  - Handles keep-alives
  - Builds URL param string:
    - Content of XACML in CURRI HTTP request body
    - Plus parameters from original request URL
  - Internally dispatches request to CVP servlet

CURRI Request Forwarding To CVP

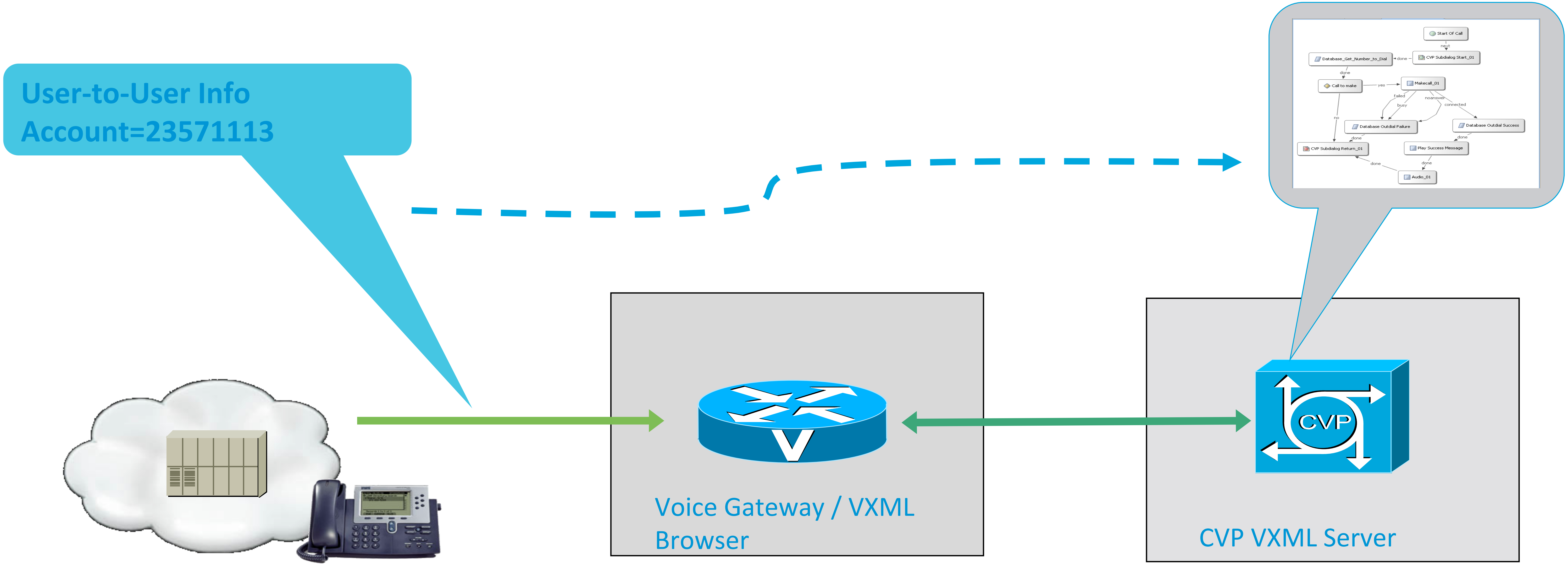


Forwarder servlet reformats request parameters to be CVP compatible

Thank you

Call Signaling – Accessing Incoming UUI

Script Access To Incoming User-User Info



Accessing Incoming User-User Info

- GTD (Generic Transparency Descriptor) used to transport call signalling information between VoIP endpoints
- More information available than just ANI and DNIS, for example (but network dependent)
  - Calling Party information including presentation/screening indicators
  - User-User information
  - Redirection information
- Both TCL and VoiceXML can read and modify GTD content
- UUI and other incoming signalling information can readily be accessed on the VoiceXML gateway
- How is it done? …

Referencing GTD content

- Access in TCL or VoiceXML
- VoiceXML uses format *attribute[instance].field*
- For example: UUS[0].dat, CGN[0].pi
- UUS,pd,dat
  pd – protocol discriminator
  dat – user-to-user info
- CGN,noa,cni,npi,pi,si,#
  noa – nature of address
  cni – complete number indicator
  npi – numbering plan indicator
  pi – presentation indicator
  si – screening indicator
  # – address
- Useful reference guide ITU-T Recommendation Q.1980.1

Accessing Incoming User-User Info

- Could extract from GTD in CVPSelfService.tcl and include as parameter on the initial URL
  - Requires modification to standard CVP TCL script
  - Better to use alternative approach driven from application
- Serve a custom VoiceXML document from the CVP VoiceXML application that assigns GTD content to element or session variables
  1. VoiceXML Insert element to deliver custom VoiceXML
  2. Custom element that uses VFC's and specify GTD item via configuration settings

  (More flexible although bit more difficult approach than VoiceXML Insert)

Retrieving UUS from GTD content (VoiceXML Insert Example)

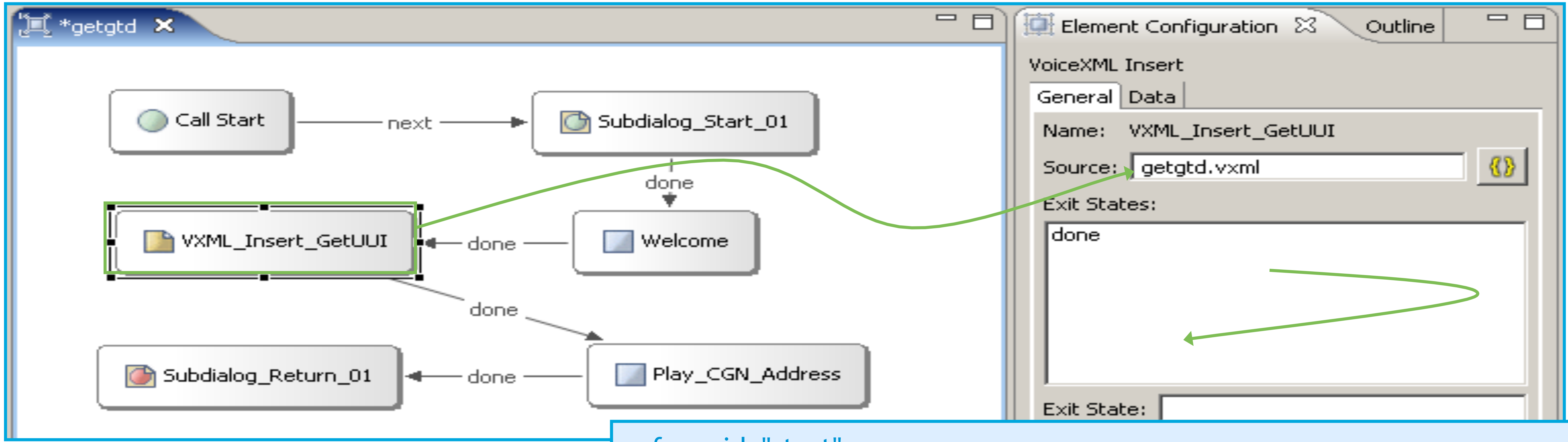
```
<vxml version="1.0" application="/CVP/Server?audium_vxml_root=true&amp;calling_into=getgtduus&amp;namelist=element_log_uusdat">

  <form id="audium_start_form">
    <block>
      <assign name="audium_vxmlLog" expr="''" />
      <assign name="audium_element_start_time_millisecs" expr="new Date().getTime()" />
      <goto next="#start" />
    </block>
  </form>

  <form id="start">
    <var name="setup_gtd" expr="com.cisco.signal.gtdlist['setup_indication']"/>
    <var name="element_log_uusdat" expr="setup_gtd.UUS[0].dat"/>

    <block>
      <assign name="audium_exit_state" expr="'done'" />
      <return namelist="audium_exit_state element_log_uusdat audium_vxmlLog audium_hotlink audium_hotevent
     audium_error audium_action" />
    </block>
  </form>
</vxml>
```

Assign required GTD field to variable

Return to VoiceXML server and populate element data

- VoiceXML Insert element references subdialog document above
- VoiceXML browser assigns variable(s) with the required GTD content and CVP element variables are populated when subdialog returns

GetGTD Example – Calling Party Number




```
<form id="start">
    <var name="setup_gtd" expr="com.cisco.signal.gtdlist['setup_indication']"/>
    <var name="element_log_cgnpi" expr="setup_gtd.CGN[0].pi"/>
    <var name="element_log_cgnsi" expr="setup_gtd.CGN[0].si"/>
    <var name="element_log_cgnaddr" expr="setup_gtd.CGN[0]['#']"/>
```

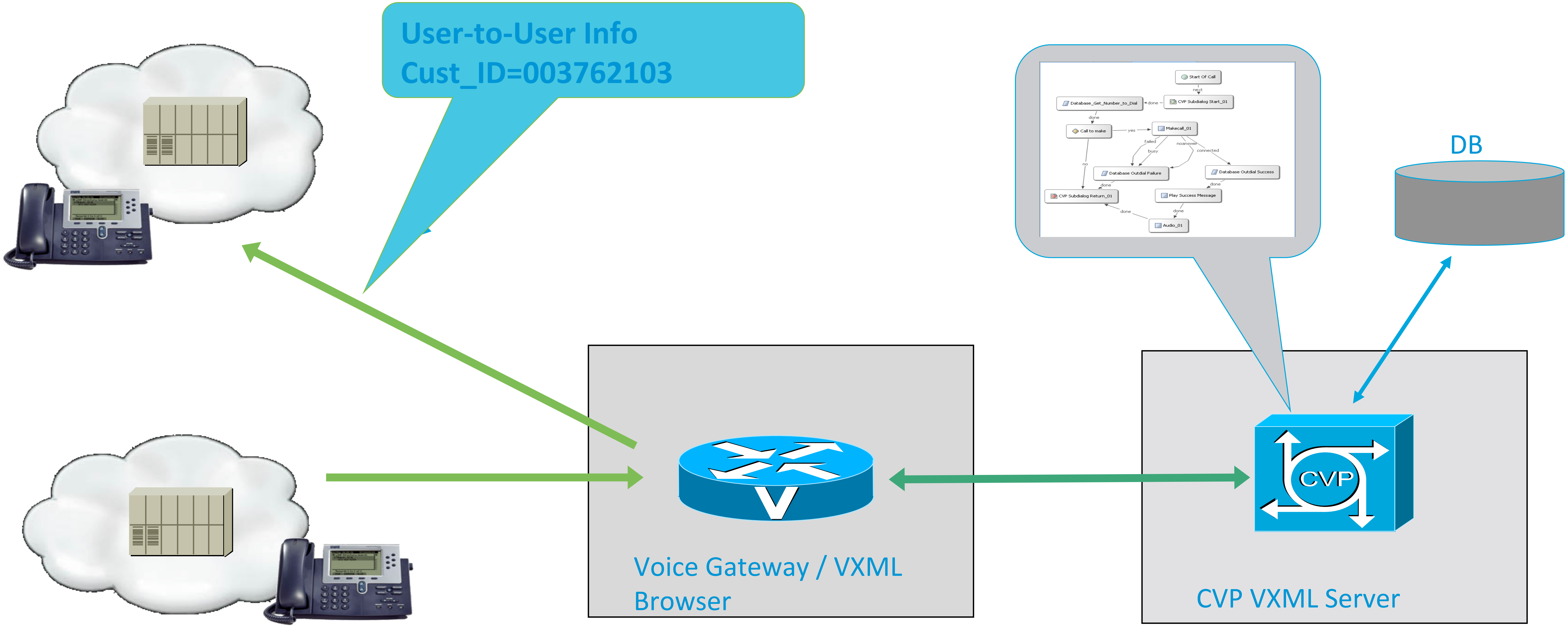
```
*Jan 13 22:41:33.922: ISDN Se0/0/0:15
Q931: RX <- SETUP pd = 8  callref = 0x00F2
        Bearer Capability i = 0x8090A3
                Standard = CCITT
                Transfer Capability = Speech
                Transfer Mode = Circuit
                Transfer Rate = 64 kbit/s
        Channel ID i = 0xA98382
                Exclusive, Channel 2
        Calling Party Number i = 0x0081, '1000'
                Plan:Unknown, Type:Unknown
        Called Party Number i = 0x80, '7784423'
                Plan:Unknown, Type:Unknown
```

```
USI,rate,c,s,c,1
USI,lay1,alaw
TMR,00
CPN,00,,u,7784423
CGN,00,,u,y,2,1000
UUS,0,4e6574776f726b657273
CPC,09
FCI,,,,,,,y,
GCI,29d51fd7e0fa11dd807400169db58e40"
```

```
4,VXML_Insert_GetUUI,data,cgnaddr,1000
4,VXML_Insert_GetUUI,exit,done
```

Call Signaling – Sending UUI

Sending User-User Info on Transfers



Sending User-User Info on Transfers

- Cisco VoiceXML Transfer Element supports additional proprietary attributes including “cisco-gtd”
- Allows application-created GTD containing UUI and other signalling data to be sent on the transferred call leg
- Not exposed as standard in CVP Call Studio transfer script element
- However, it can be included via straightforward customisation
- How is it done? …

Sending User-User Info

- Serve a custom VoiceXML document from the CVP VoiceXML application
- Builds GTD containing User-to-User attribute UUS[0].dat
- Performs the transfer including Cisco proprietary attributes
- Use either –
  1. VoiceXML Insert element to deliver custom VoiceXML as a subdialog
  2. Custom transfer element
     - Use VFC's to build the transfer VoiceXML
     - Use addProprietaryAttribute("cisco-gtd", …) method to include GTD on the transfer element
     - Specify this and other Cisco attributes via element configuration settings
- 2 is more flexible although bit more difficult approach than VoiceXML Insert

Extending Transfer Element Attributes (VoiceXML Insert Example)

```
<form id="start">
   <var name="new_gtd" expr="new com.cisco.objclass.gtd()"/>
   <var name="new_gtd.message_type" expr="'IAM'"/>
   <var name="new_gtd.UUS[0].pd" expr="0"/>
   <var name="new_gtd.UUS[0].dat" expr="audium_session_xferuus"/>

   <transfer name="xfer" bridge="true" connecttimeout="10s" destexpr="audium_session_xferdest" cisco-gtd="new_gtd">
      <filled>
         <if cond=" ( xfer ==  'far_end_disconnect' ) ">
            <assign name="audium_exit_state" expr="'done'" />
         <elseif cond=" ( xfer == 'busy' ) " />
            <assign name="audium_exit_state"
         <elseif cond=" ( xfer == 'noanswer' ) " />
            <assign name="audium_exit_state" expr="'noanswer'" />
         <elseif cond=" … " />
            <assign name=" … " />
         <else/>
            <assign name="audium_exit_state" expr="'phone_error'" />
         </if>

         <return namelist="audium_exit_state audium_vxmlLog audium_hotlink audium_hotevent audium_error        audium_action" />
      </filled>
   </transfer>
</form>
```

Create GTD and add UUS attribute with dat field set to contents of CVP session variable xferuus

Use session variable xferdest as transfer destination

Include cisco-gtd attribute on transfer element

- VoiceXML Insert element references subdialog document containing GTD assignment and transfer element with cisco-gtd attribute

Call Signaling – Display Name and Calling Party

Sending Display Name on Transfers

- Can use the same approach as sending UUI to send Display-Name on transfers
- Build GTD containing Information-for-Display DIS[0].info
- Transfer using Cisco proprietary attribute “cisco-gtd”
- Can use to pass context from IVR application that will be displayed on CUCM IP Phone
- So, several ways to forward information when call is transferred: UUI, Display-Name and also CLI override

Overriding ANI on transfers

- Could use GTD but there is a simpler method
- Can just add Cisco proprietary attribute “cisco-ani” or “cisco-aniexpr” to transfer element
- Set it to the required digit string, tel:0123456789
- Useful last resort if destination system can’t use UUI or Display-Name and only can receive called and calling party numbers

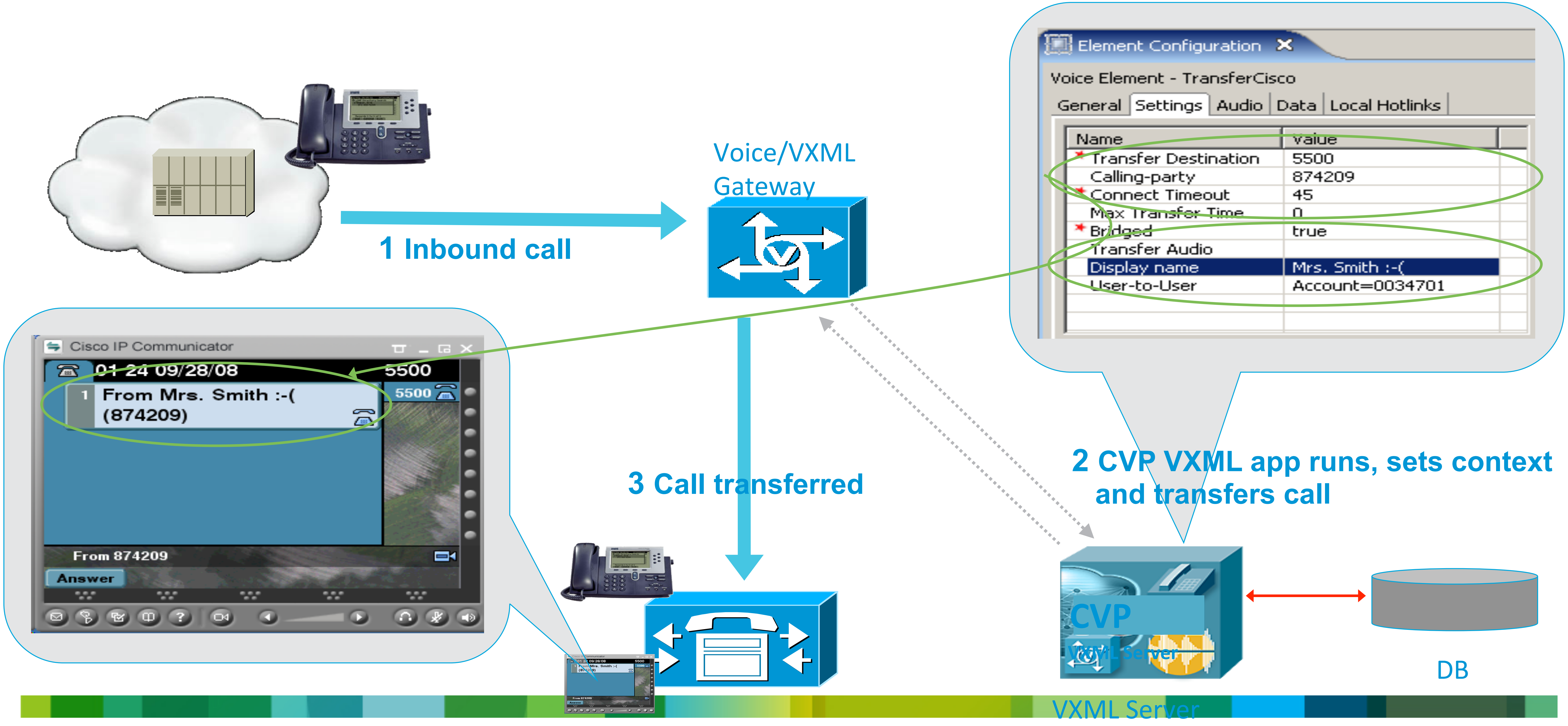
Including Display-Name and Modified ANI (VoiceXML Insert Example)

```
<form id="start">
  <var name="new_gtd" expr="new com.cisco.objclass.gtd()"/>
  <var name="new_gtd.message_type" expr="'IAM'"/>
  <var name="new_gtd.UUS[0].pd" expr="0"/>
  <var name="new_gtd.UUS[0].dat" expr="audium_session_xferuus"/>
  <var name="new_gtd.DIS[0].info" expr="audium_session_xferdisp"/>


  <transfer name="xfer" bridge="true" connecttimeout="10s"
    destexpr="audium_session_xferdest" cisco-gtd="new_gtd" cisco-aniexpr="audium_session_xferani" >

    <filled>
      <if cond=" ( xfer == 'far_end_disconnect' ) ">
        <assign name="audium_exit_state" expr="'done'" />
      <elseif cond=" ( xfer == 'busy' ) " />
        <assign name="audium_exit_state" expr="'busy'" />
      <elseif cond=" ( xfer == 'noanswer' ) " />
        <assign name="audium_exit_state" expr="'noanswer'" />
      <elseif cond=" … " />
        <assign name="…" />
      <else/>
        <assign name="audium_exit_state" expr="'phone_error'" />
      </if>

      <return namelist="audium_exit_state audium_vxmlLog audium_hotlink audium_hotevent audium_error      audium_action" />
```

Create GTD and add DIS attribute with info field set to contents of CVP session variable xferdisp

Include cisco-aniexpr attribute set to session variable xferani

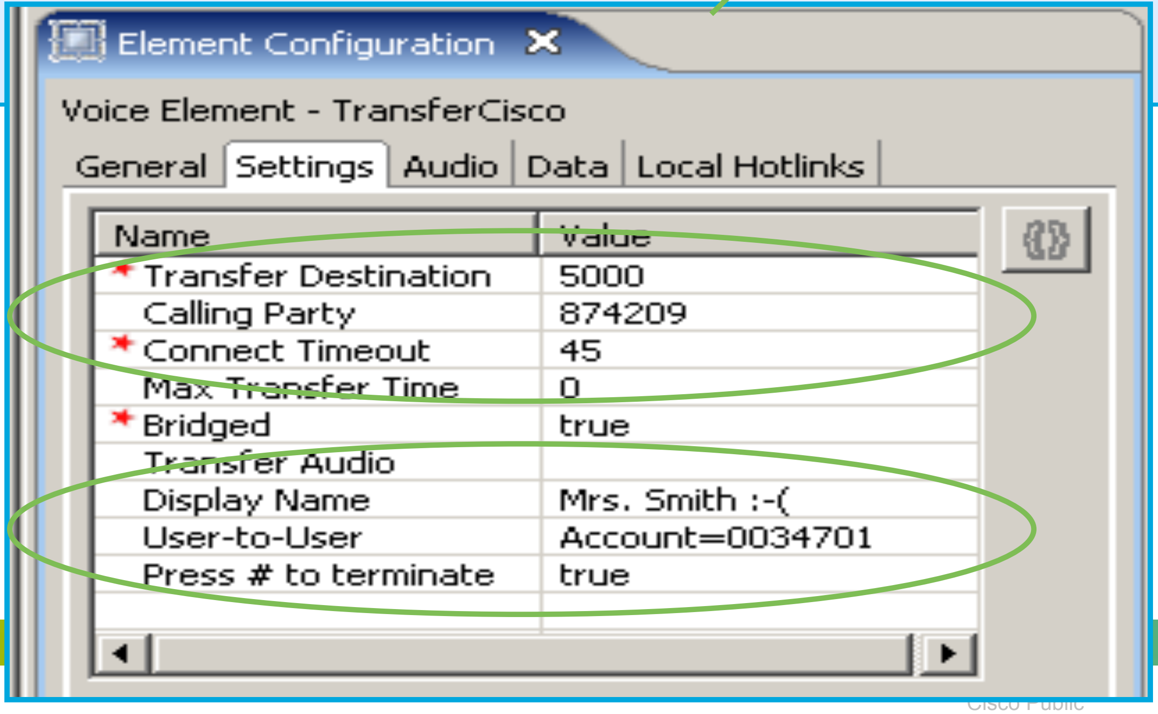
CVP VoiceXML Passing Context (Using Custom Transfer Element)



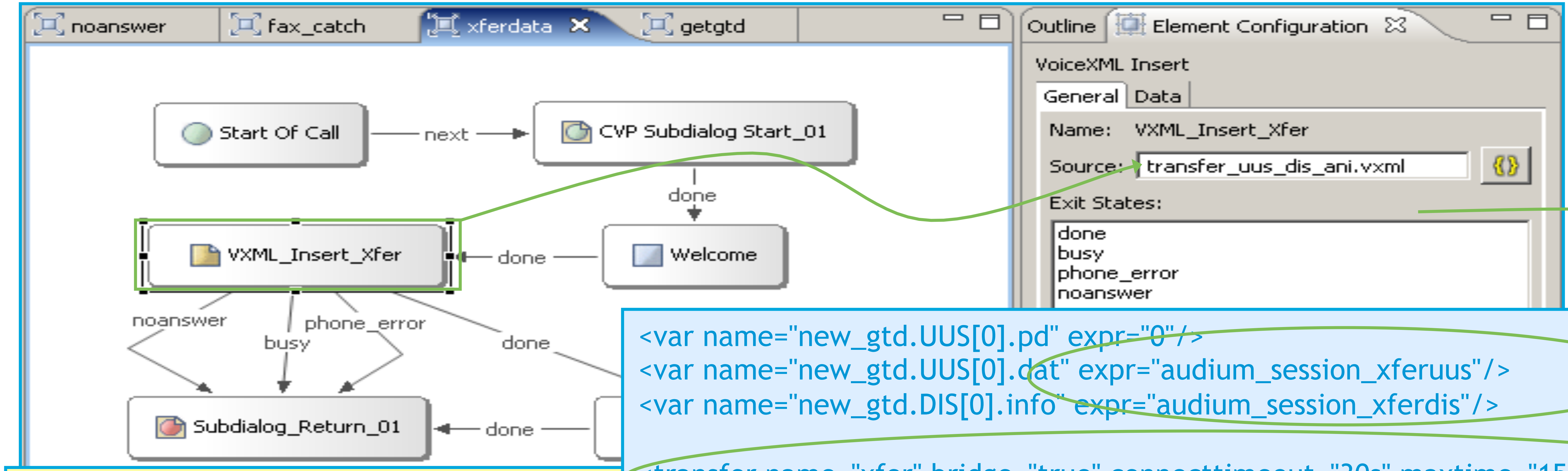
VoiceXML Form Generated by Custom Transfer Element

```
<form id="start">
    <var name="gtd" expr="new com.cisco.objclass.gtd()" />
    <var name="gtd.message_type" expr="'IAM'" />
    <var name="gtd.UUS[0].pd" expr="0" />
    <var name="gtd.UUS[0].dat" expr="'Account=0034701'" />
    <var name="gtd.DIS[0].info" expr="'Mrs. Smith :-('" />
    <transfer name="xfer" bridge="true" connecttimeout="45s" maxtime="0s" dest="phone://5000" cisco-ani="tel:874209" cisco-gtd="gtd" cisco-longpound="true" />

    <filled mode="all">
      <submit next="/CVP/Server" method="post" namelist="audium_vxmlLog xfer" />
    </filled>
  </form>
```

- Shows GTD creation and attribute setting
- Shows Cisco proprietary attributes
- Also includes cisco-longpound which allows the caller to terminate the transfer using #
- This example used configurable settings



Transfer With Data Example



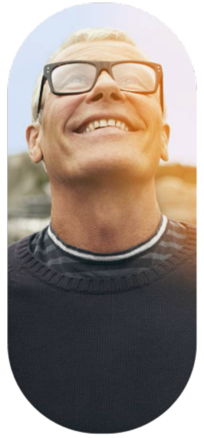

```
<var name="new_gtd.UUS[0].pd" expr="0"/>
<var name="new_gtd.UUS[0].dat" expr="audium_session_xferuus"/>
<var name="new_gtd.DIS[0].info" expr="audium_session_xferdis"/>

<transfer name="xfer" bridge="true" connecttimeout="30s" maxtime="15s“
    destexpr="'phone://' + audium_session_xferdest"
    cisco-aniexpr="'tel:' + audium_session_xferani" cisco-gtd="new_gtd">
```

```
Sent:
INVITE sip:6681000@192.168.1.78:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.78:5060;x-route-tag="tgrp
Remote-Party-ID: "Cisco CVP Transfer Test" <sip:0123456789@192.168.1.78>;party=calling;screen=yes;privacy=off
From: "Cisco CVP Transfer Test" <sip:0123456789@192.168.1.78>;tag=66DB74-1EA3
To: <sip:6681000@192.168.1.78>
Diversion: <sip:7784423@192.168.1.78>;privacy=off;reason=follow-me;screen=no

--uniqueBoundary
Content-Type: application/gtd
```

```
*Jan 14 14:06:37.955: ISDN Se0/0/0:15
Q931: RX <- SETUP pd = 8  callref = 0x0089
        Calling Party Number i = 0x0081, '1000'
                Plan:Unknown, Type:Unknown
        Called Party Number i = 0x80, '7784423'
                Plan:Unknown, Type:Unknown
```

```
6,VXML_Insert_Xfer,data,xferdis,Cisco CVP Transfer Test
6,VXML_Insert_Xfer,data,xferuus,Hello from Barcelona
```

MediaSense Recording Control



MediaSense Recording Control

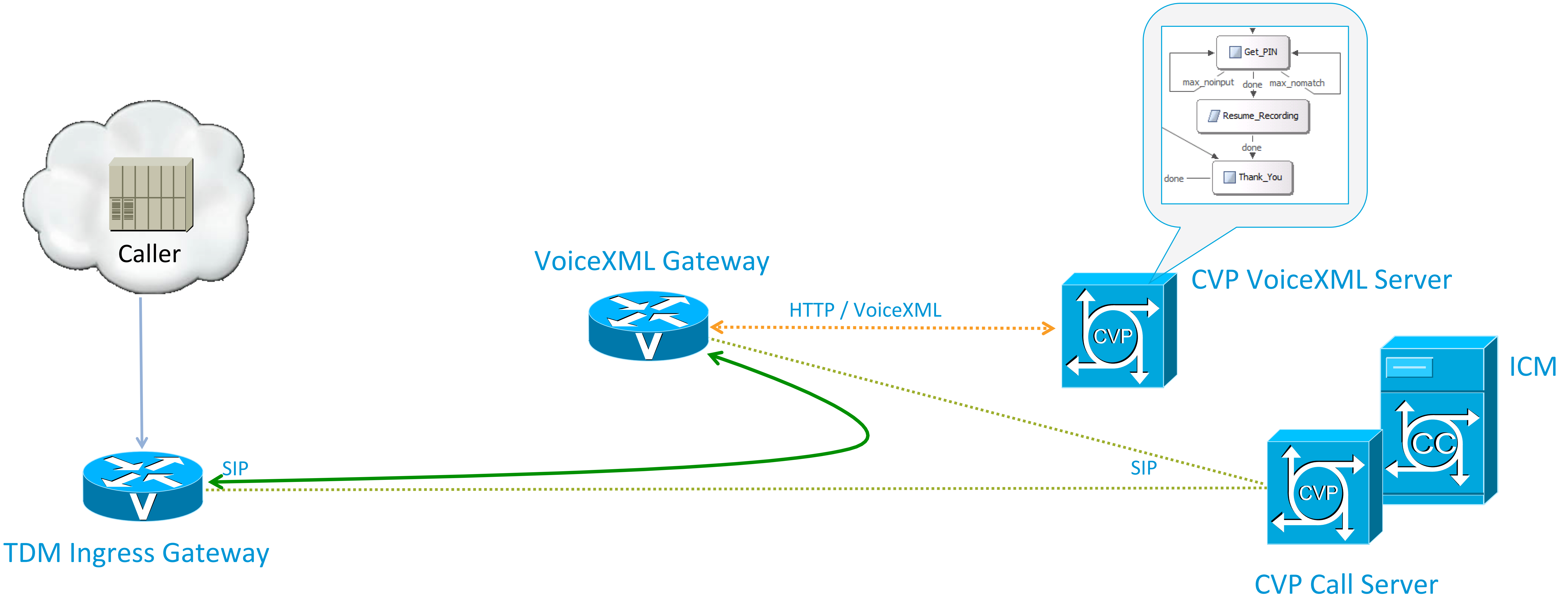
Use Case / Challenge

- Trunk-side call recording model
  - Media-forking to MediaSense from ingress CUBE
- Using IVR to collect sensitive data such as payment card details or authentication code
- Need to turn recording off/on around sensitive data collection

Problem: How does a CVP Call Studio application control a MediaSense recording session for the current call?

CVP Comprehensive
MediaSense Recording Control

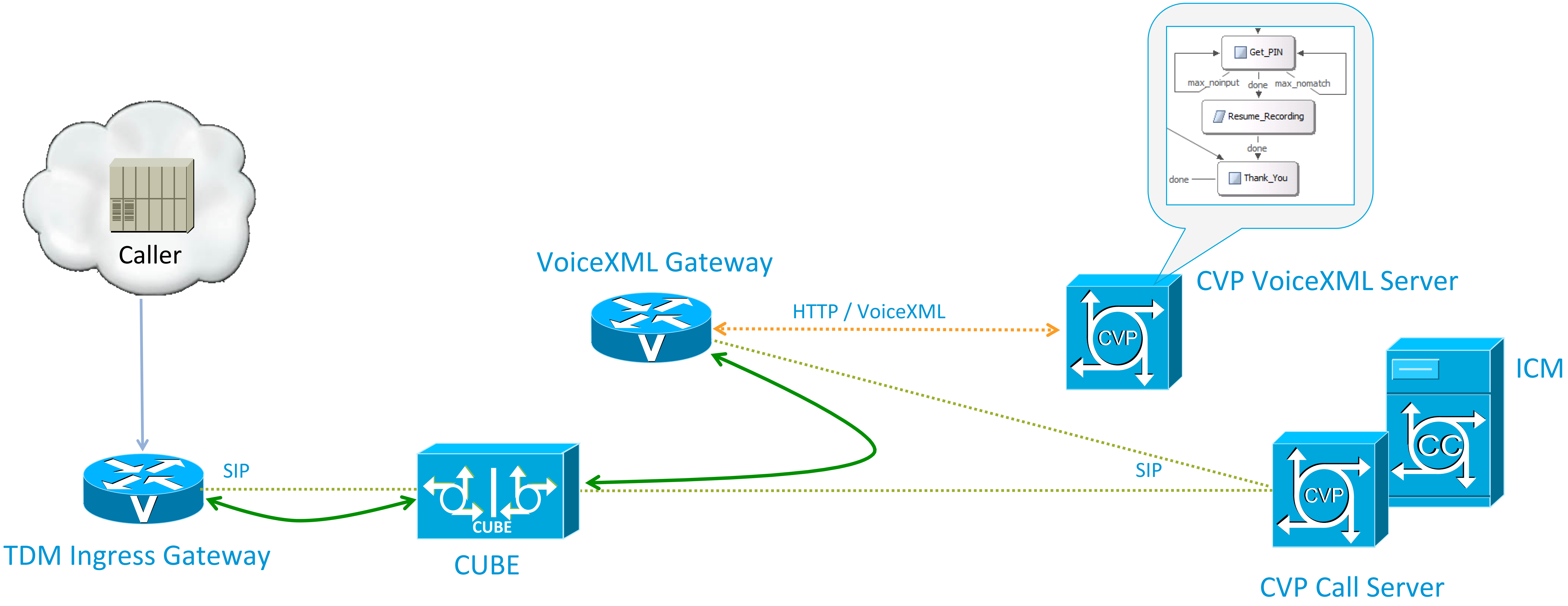
Incoming call leg
SIP Signalling
RTP Media Stream



MediaSense Recording Control

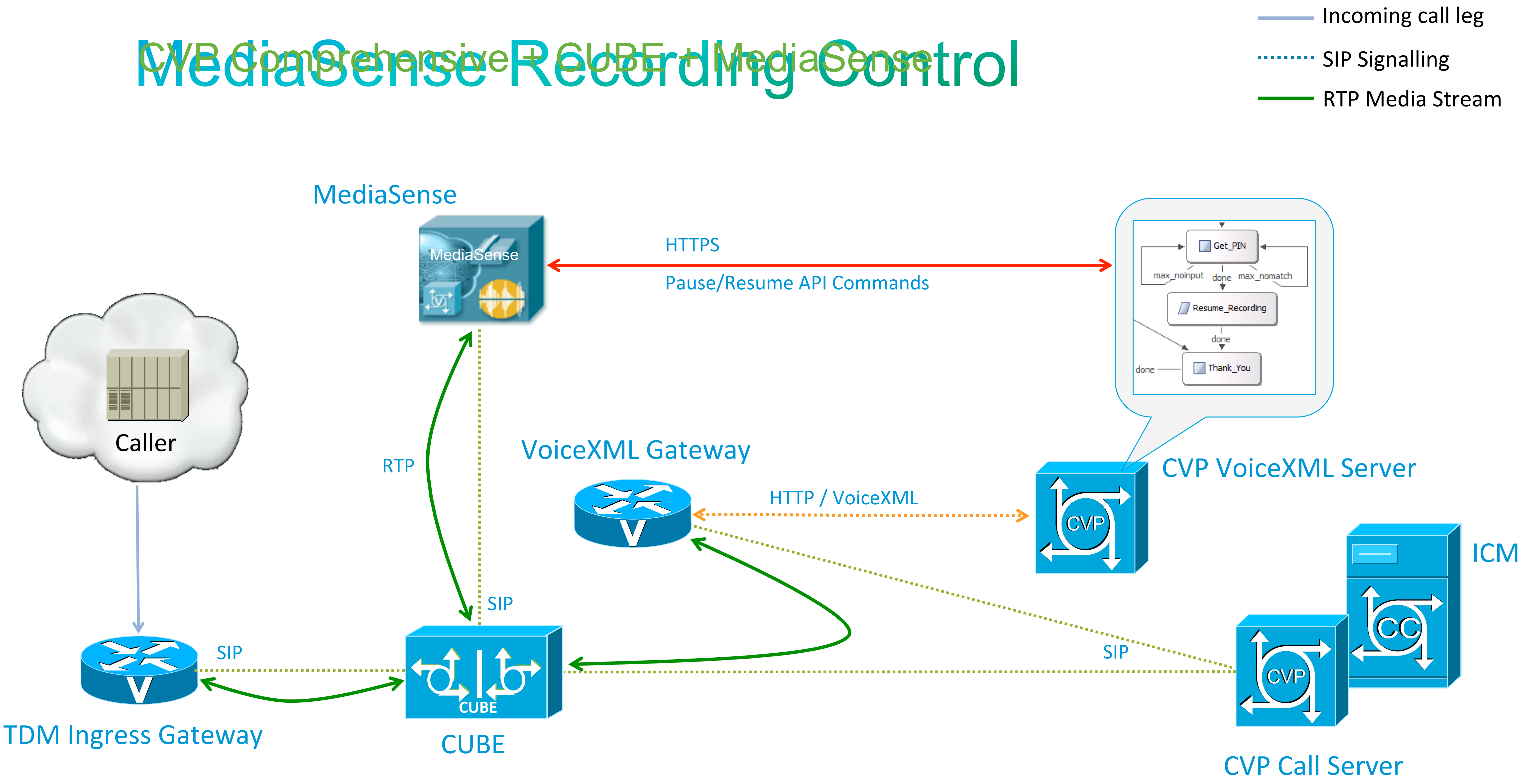
CVP Comprehensive + CUBE

Incoming call leg
SIP Signalling
RTP Media Stream



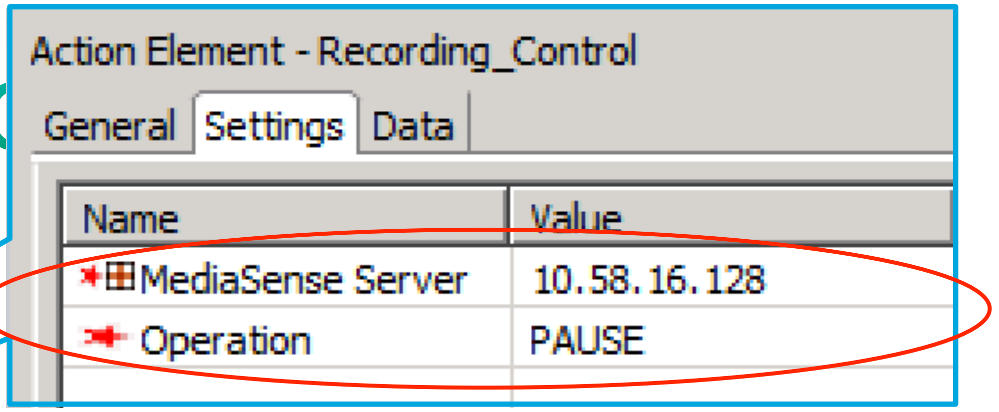
MediaSense Recording Control

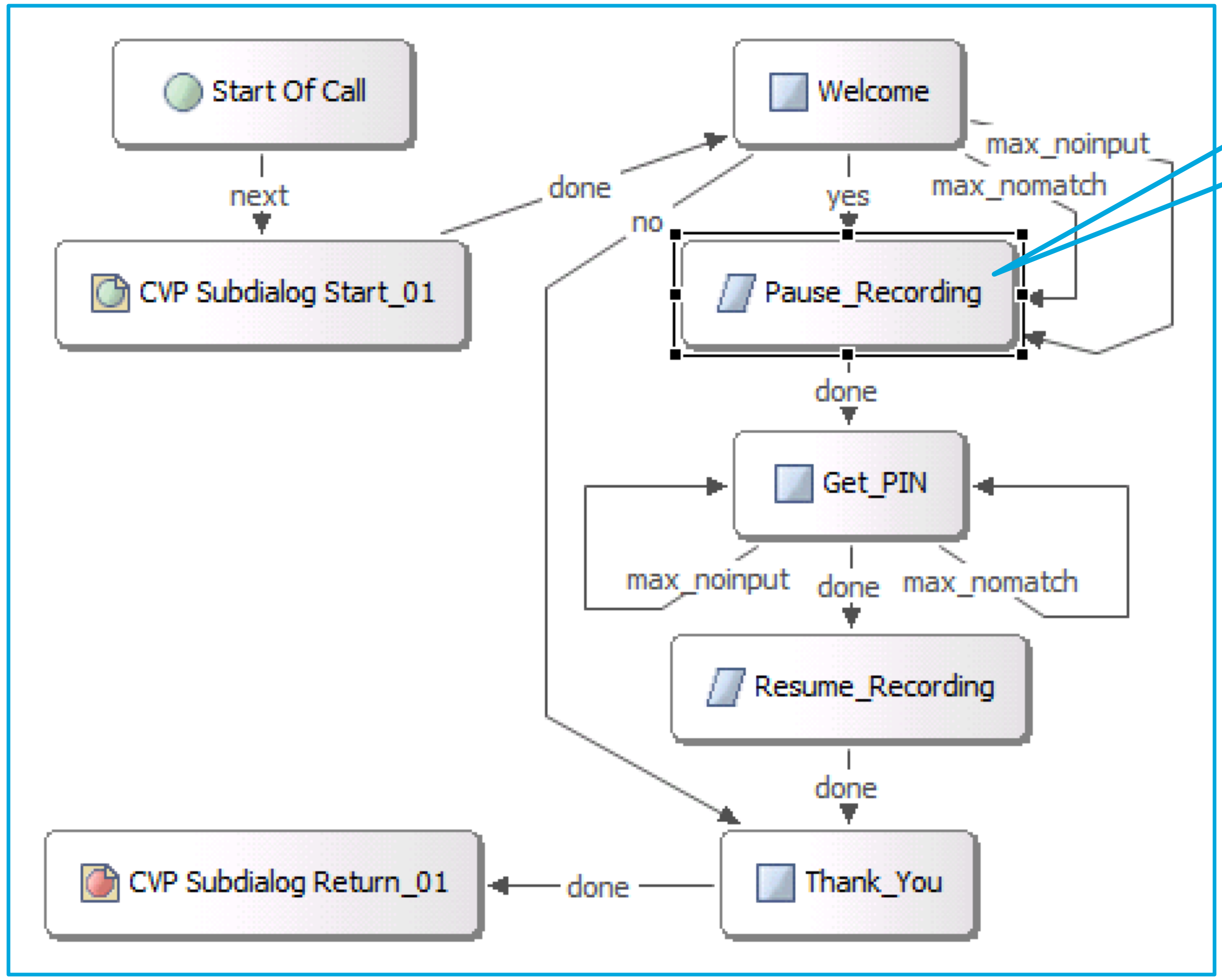
CVP Comprehensive + CUBE + MediaSense



Script Elements

MediaSense Recording Co

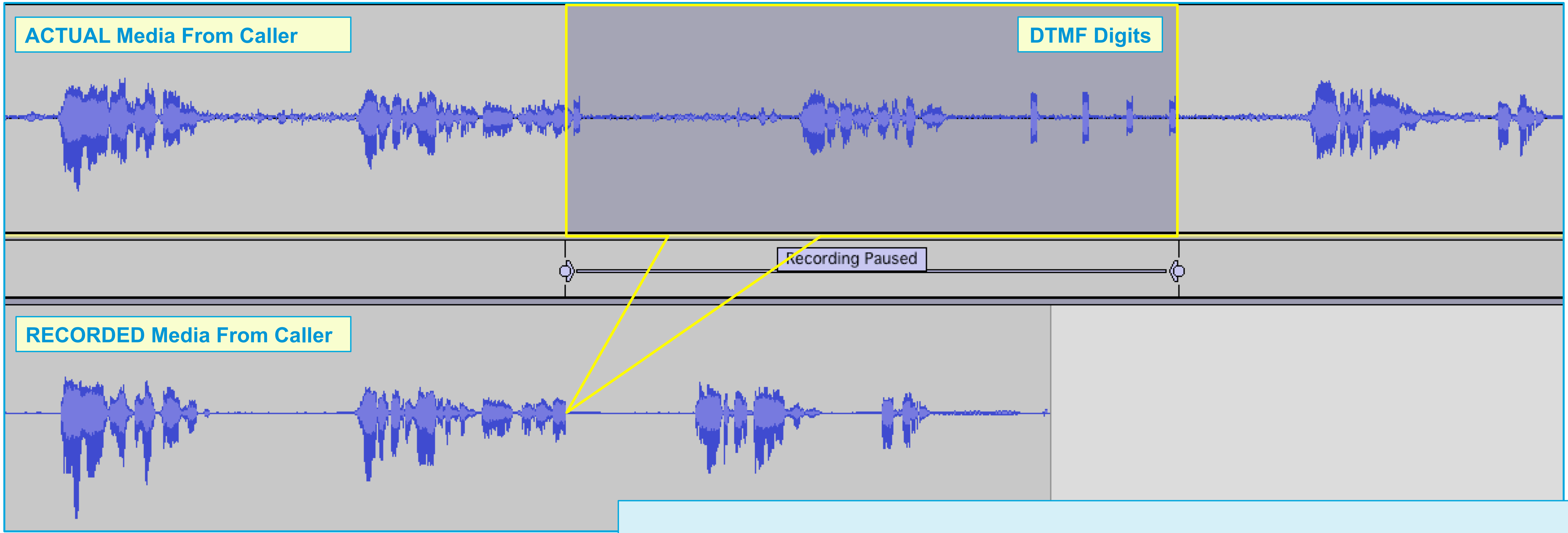




- Custom Action Element using MediaSense REST-like API over HTTPS
- Convert session data callid into Mediasense Call Correlation ID format (hex to decimal), for example:
  - Call ID = 06D0FDDE67B111E281E1001B0CFAA768
  - CCID = 0114359774-1739657698-2179006491-0217753448
- Sign-in to the MediaSense server and authenticate, JSESSIONID returned
- Use MediaSense API getSessionsByCCID method to retrieve the MediaSense session ID for the CCID
- Extract session ID from the response and use it as the parameter on subsequent pauseRecording and resumeRecording methods
- CVP VoiceXML session data can be used to store the MediaSense API session context and the recording session ID across multiple recording control requests

MediaSense Recording Control

Media Waveform



- Media recording is skipped for the duration of the pause
- Tags with relative timestamps are added to the recording session data when pause and resume are executed

```
!!! Session Event: recording STARTED, state ACTIVE, stream rtsp://10.58.16.128/8ef13c7be82fbe1
    -- Track 1, type AUDIO, participant 3899, started Sun 27 Jan 12:07:59
    -- Track 0, type AUDIO, participant 396298, started Sun 27 Jan 12:07:59
```

Lookup recording session ID

```
https://10.58.16.128:8440/ora/queryService/query/getSessionsByCCID?value=0114359774-1739657698-2179006491-0217753448&offset=0&limit=1
```

MediaSense response data


```
{"sessions": {
     "sessionState": "ACTIVE",
     "callControllerType": "Cisco-SIPGateway",
     "sessionId": "8ef13c7be82fbe1",
     "urls": [...
```


Pause recording session

```
https://10.58.16.128:8440/ora/controlService/control/pauseRecording
{"requestParameters": {"sessionId": "8ef13c7be82fbe1”}}
```

```
+++ Tag Event: SYSTEM_DEFINED tag "Paused" ADDED 11 secs into session 8ef13c7be82fbe1
```

Resume recording session

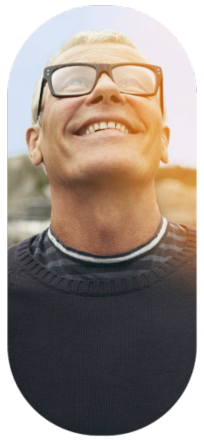
```
https://10.58.16.128:8440/ora/controlService/control/resumeRecording
{"requestParameters": {"sessionId": "8ef13c7be82fbe1”}}
```

```
+++ Tag Event: SYSTEM_DEFINED tag "Resumed" ADDED 21 secs into session 8ef13c7be82fbe1
```

```
!!! Session Event: recording ENDED, state CLOSED_NORMAL, stream rtsp://10.58.16.128/archive/8ef13c7be82fbe1
    -- Track 1, type AUDIO, participant 3899, duration 28 secs, http://10.58.16.128:8081/mma/ExportRaw?recording=8ef13c7be82fbe1-TRACK1
    -- Track 0, type AUDIO, participant 396298, duration 28 secs, http://10.58.16.128:8081/mma/ExportRaw?recording=8ef13c7be82fbe1-TRACK0
```

cisco

Dynamic Menus



Use Case / Challenge

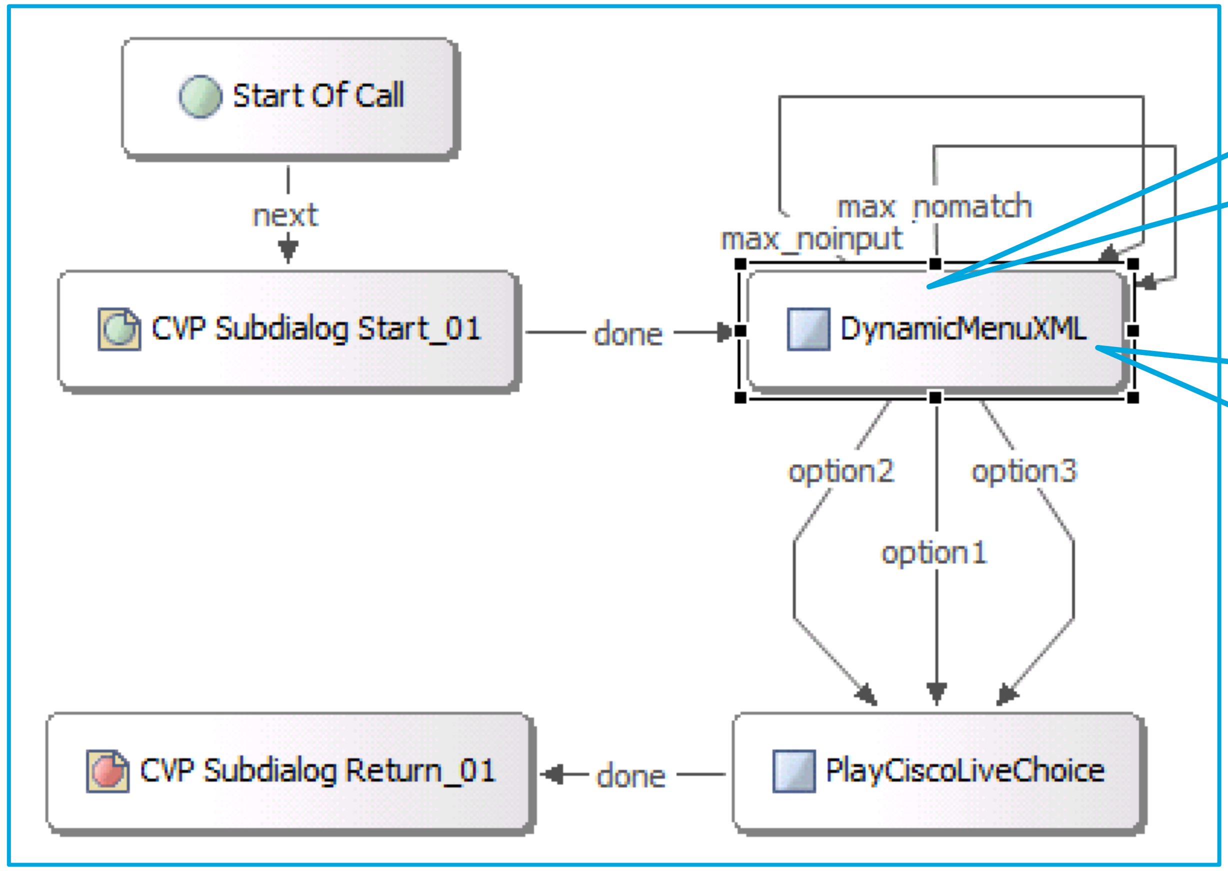
Dynamic Menus

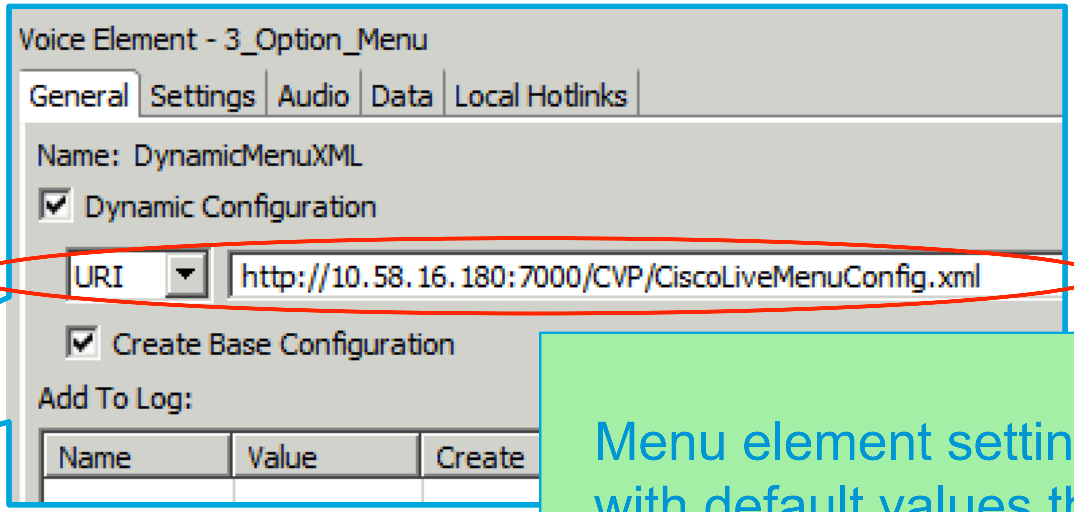
- Predictive IVR – presenting menus to the caller based on the most likely reasons for calling
  - Have they just made a booking and is this most likely a follow-up call?
  - Have they just been sent a monthly bill (possibly with exceptional charges)?
  - Derive possible reasons from back-end system caller profile/history
- Randomise the menu items
  - Possibly for tele-voting application candidate lists
- Personalised menus
  - Only offer the menu items relevant to a particular customer

Problem: CVP Call Studio menu elements are statically configured with a preselected number of options

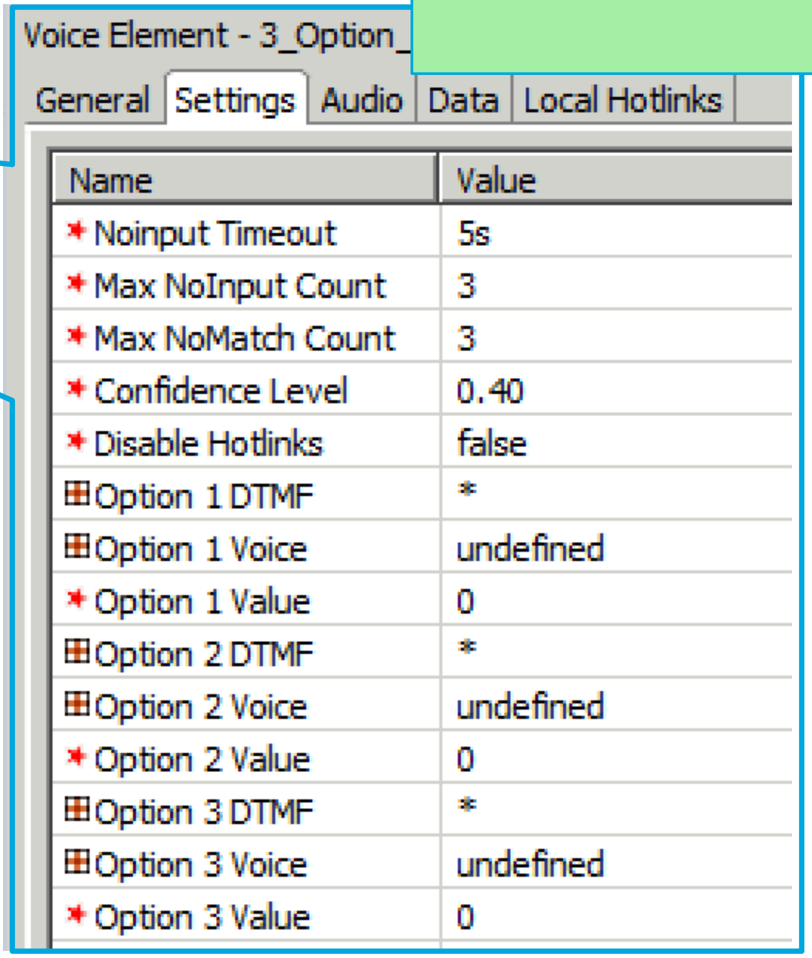
Run-time Configuration (XML)

Dynamic Menus





Menu element settings configured with default values that can be overridden at run-time

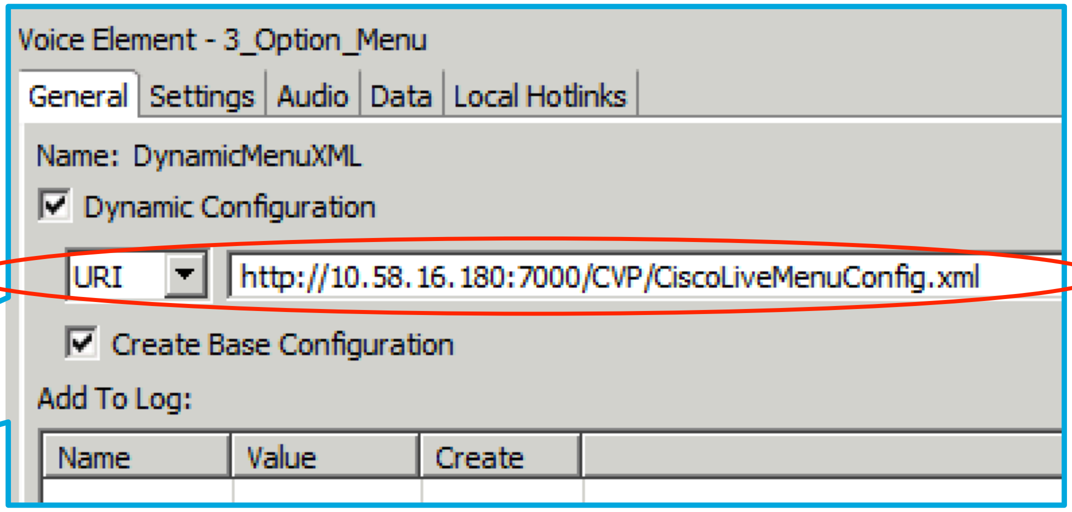

Voice Element - 3_Option_

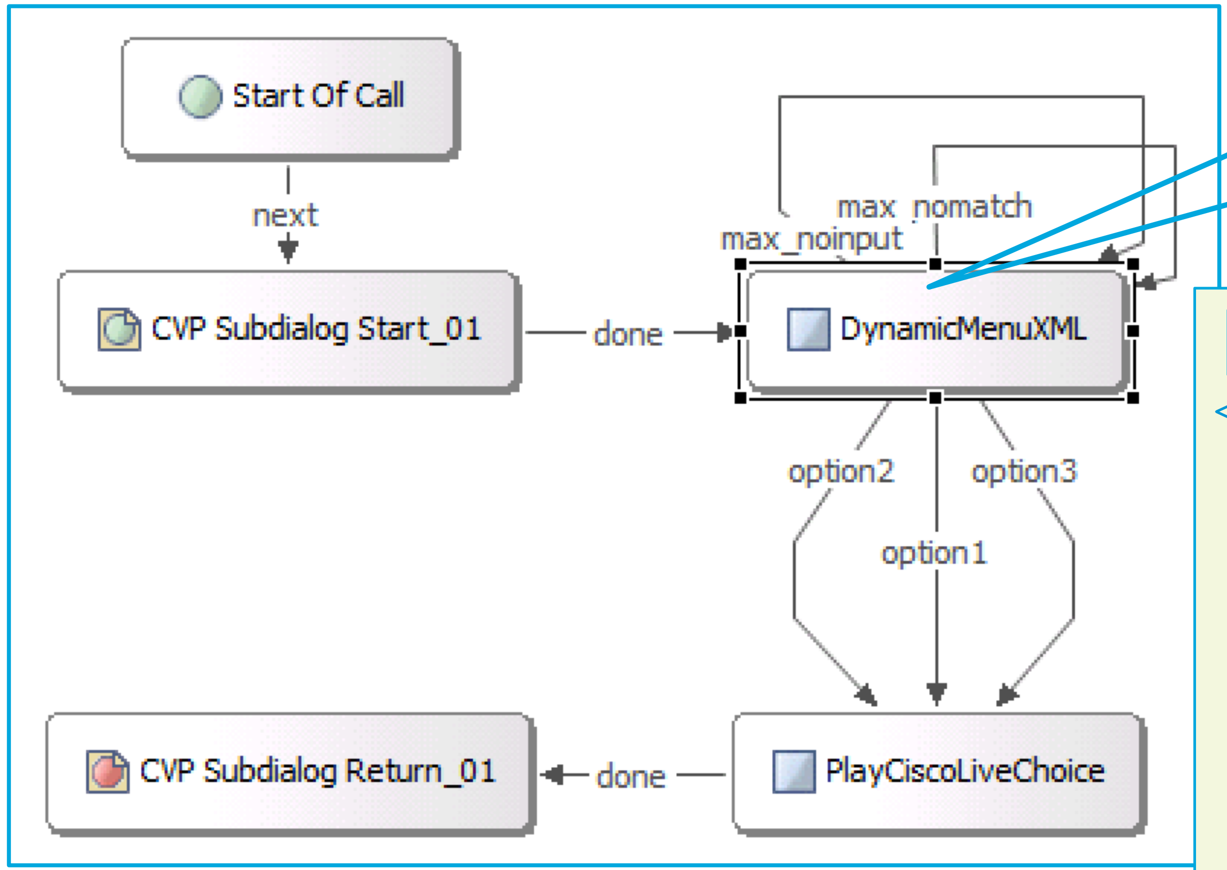
General | Settings | Audio | Data | Local Hotlinks

Name	Value
* Noinput Timeout	5s
* Max NoInput Count	3
* Max NoMatch Count	3
* Confidence Level	0.40
* Disable Hotlinks	false
⊞Option 1 DTMF	*
⊞Option 1 Voice	undefined
* Option 1 Value	0
⊞Option 2 DTMF	*
⊞Option 2 Voice	undefined
* Option 2 Value	0
⊞Option 3 DTMF	*
⊞Option 3 Voice	undefined
* Option 3 Value	0

Run-time Configuration (XML)

Dynamic Menus





CiscoLiveMenuConfig.xml

```
<configuration class="com.audium.server.voiceElement.menu.MFoundation3OptionMenu">
    <setting name="inputmode">both</setting>

    <audio_group name="initial_audio_group" bargein="true">
        <audio name="audio item">Welcome to Cisco Live London. </audio>
        <audio name="audio item">Press 1 for Collaboration sessions,
                                 2 for Security or 3 for Data Centre</audio>
    </audio_group>

    <setting name="option1_dtmf">1</setting>
    <setting name="option1_voice">Collaboration</setting>
    <setting name="option1_value">collab</setting>

    <setting name="option2_dtmf">2</setting>
    <setting name="option2_voice">Security</setting>
    <setting name="option2_value">sec</setting>

    <setting name="option3_dtmf">3</setting>
    <setting name="option3_voice">Data Centre</setting>
    <setting name="option3_value">data</setting>
</configuration>
```

Configure audio

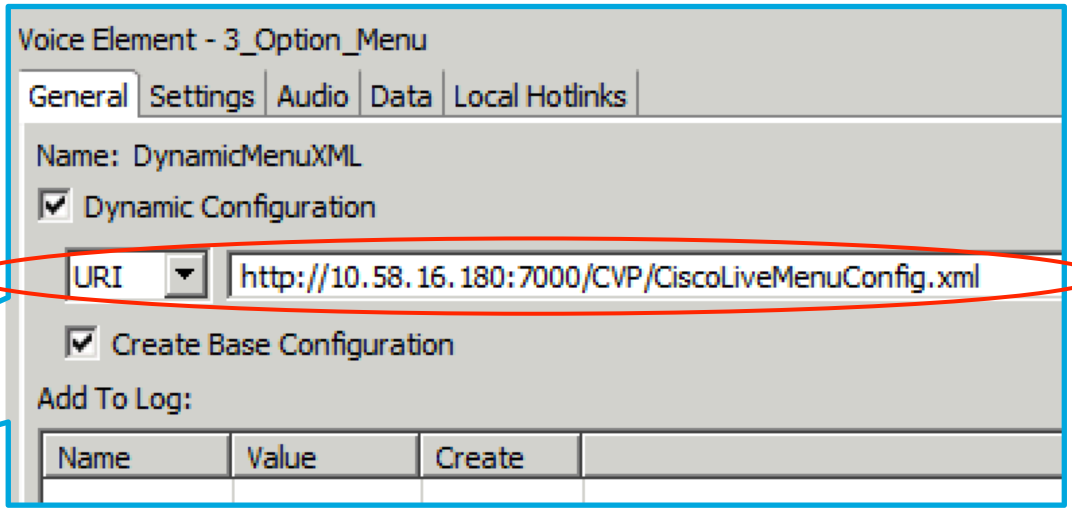
Override option 1 settings

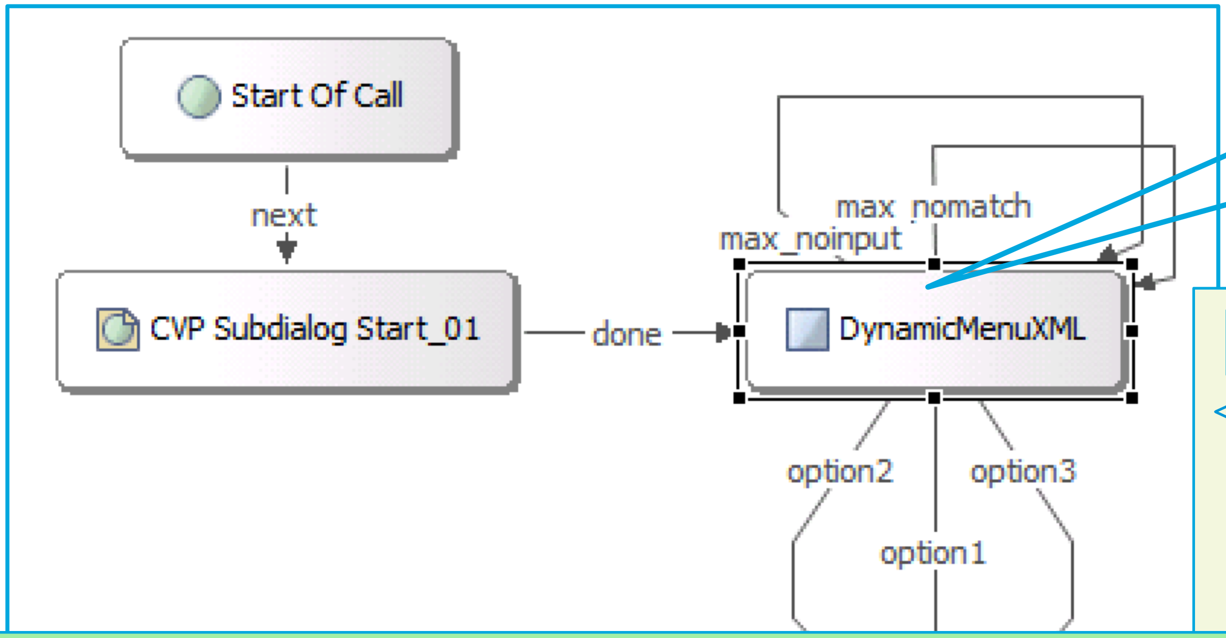
Override option 2 settings

Override option 3 settings

Run-time Configuration (XML)

Dynamic Menus





CiscoLiveMenuConfig.xml

```
<configuration class="com.audium.server.voiceElement.menu.MFoundation3OptionMenu">
    <setting name="inputmode">both</setting>

    <audio_group name="initial_audio_group" bargein="true">
        <audio name="audio item">Welcome to Cisco Live London. </audio>
        ...m">Press 1 for Collaboration sessions,
                2 for Security or 3 for Data Centre</audio>

...f">1</setting>
...ce">Collaboration</setting>
...ue">collab</setting>

...f">2</setting>
...ce">Security</setting>
...ue">sec</setting>

...f">3</setting>
...ce">Data Centre</setting>
...ue">data</setting>
```

Configure audio

Override option 1 settings

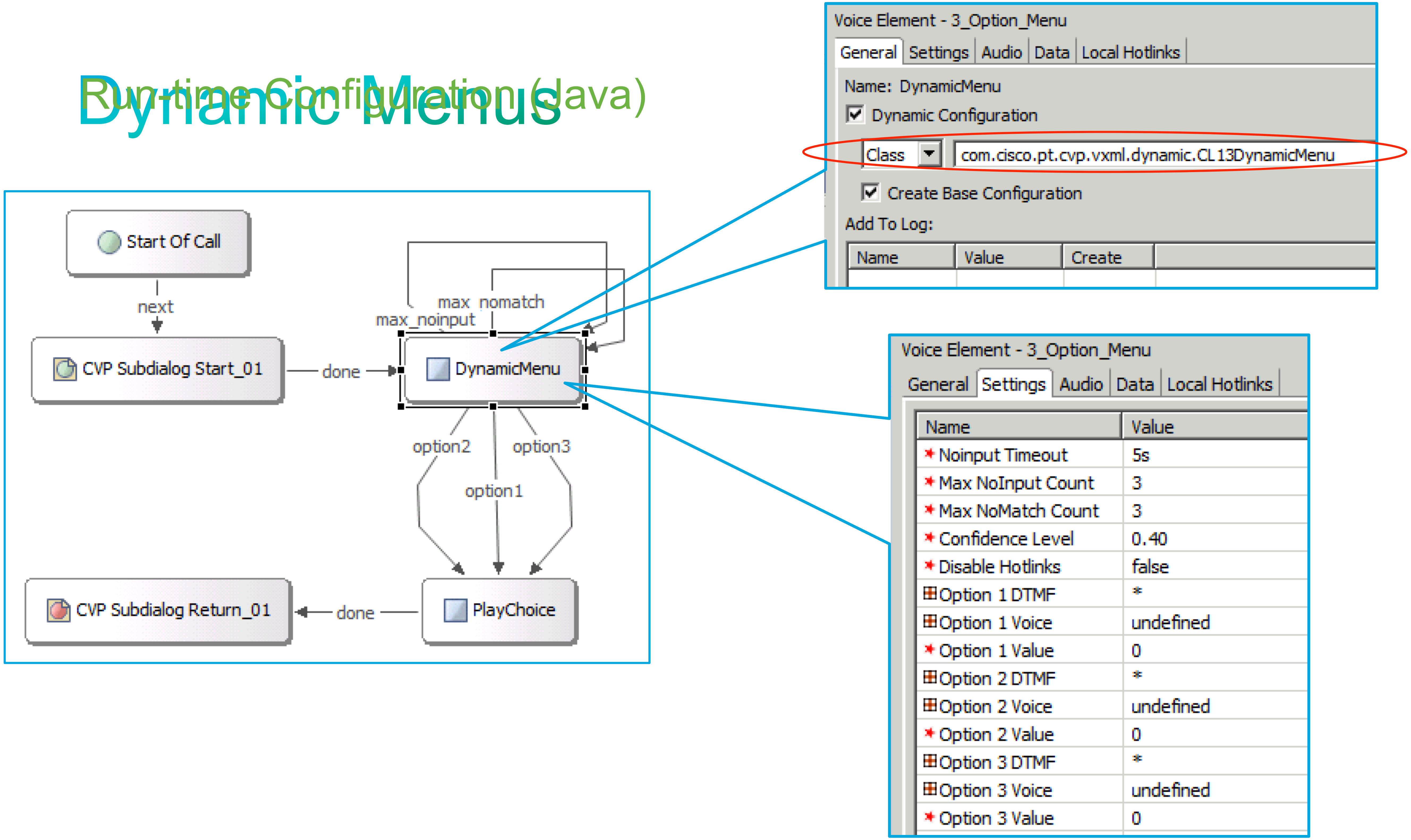
Override option 2 settings

Override option 3 settings

```
<form id="start”>
  <field name="choice_fld" modal="false”>
    <property name="inputmodes" value="dtmf voice" />

    <prompt bargein="true">Welcome to Cisco Live London. Press 1 for
       Collaboration sessions, 2 for Security or 3 for Data Centre</prompt>

    <option value="collab" dtmf="1">Collaboration</option>
    <option value="sec" dtmf="2">Security</option>
    <option value="data" dtmf="3">Data Centre</option>

    <filled>
       ...
    </filled>
  </field>
</form>
```

Dynamic Menus

Run-time Configuration (Java)

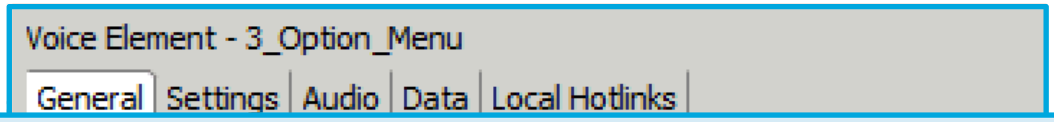


Voice Element - 3_Option_Menu

General | Settings | Audio | Data | Local Hotlinks

```
public class CL13DynamicMenu implements VoiceElementInterface {

// Dynamically configues the menu element from an array of data objects, each one
// specifying the items required for a single menu option. In this example the
// data is statically defined but it could be retrieved from anywhere.

    String prompt = "Please select one of the following options";

    MenuOption[ ] optList = new MenuOption[ ]{new MenuOption("booking", "Booking", "If you're calling about a booking you've just made, press "),
                                             new MenuOption("account", "Account", "If you'd like to talk to us about your frequent flyer account, press "),
                                             new MenuOption("baggage", "Baggage", "If you're trying to locate lost baggage, press ")};
```

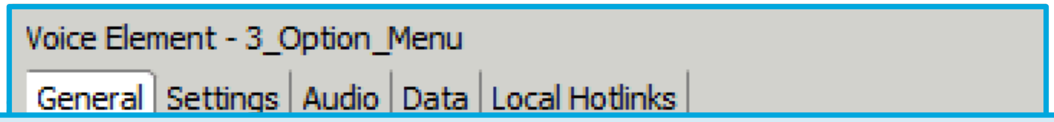
Create configuration sample data as an array of menu option objects containing text strings. Data could come from anywhere you choose.




```
public class CL13DynamicMenu implements VoiceElementInterface {

// Dynamically configues the menu element from an array of data objects, each one
// specifying the items required for a single menu option. In this example the
// data is statically defined but it could be retrieved from anywhere.

   String prompt = "Please select one of the following options";

   MenuOption[ ] optList = new MenuOption[ ]{new MenuOption("booking", "Booking", "If you're calling about a booking you've just made, press "),
                                              new MenuOption("account", "Account", "If you'd like to talk to us about your frequent flyer account, press "),
                                              new MenuOption("baggage", "Baggage", "If you're trying to locate lost baggage, press ")};


   public VoiceElementConfig getConfig(String name,
                                        ElementAPI elementAPI,
                                        VoiceElementConfig cfg) throws AudiumException
   {
```

Create configuration sample data as an array of menu option objects containing text strings. Data could come from anywhere you choose.

Implement getConfig method




```
public class CL13DynamicMenu implements VoiceElementInterface {

// Dynamically configues the menu element from an array of data objects, each one
// specifying the items required for a single menu option. In this example the
// data is statically defined but it could be retrieved from anywhere.

    String prompt = "Please select one of the following options";

    MenuOption[ ] optList = new MenuOption[ ]{new MenuOption("booking", "Booking", "If you're calling about a booking you've just made, press "),
                                              new MenuOption("account", "Account", "If you'd like to talk to us about your frequent flyer account, press "),
                                              new MenuOption("baggage", "Baggage", "If you're trying to locate lost baggage, press ")};


    public VoiceElementConfig getConfig(String name,
                                        ElementAPI elementAPI,
                                        VoiceElementConfig cfg) throws AudiumException
    {
        AudioGroup init_audgrp = cfg.new AudioGroup("initial_audio_group", false);

        StaticAudio tts = cfg.new StaticAudio(prompt, null);
        init_audgrp.addAudioItem(tts);
        init_audgrp.setBargein(true);
        cfg.setAudioGroup(init_audgrp);
```

Create configuration sample data as an array of menu option objects containing text strings. Data could come from anywhere you choose.

Implement getConfig method

Create audio group and add the first prompt

Voice Element - 3_Option_Menu

General | Settings | Audio | Data | Local Hotlinks

```
public class CL13DynamicMenu implements VoiceElementInterface {

// Dynamically configues the menu element from an array of data objects, each one
// specifying the items required for a single menu option. In this example the
// data is statically defined but it could be retrieved from anywhere.

   String prompt = "Please select one of the following options";

   MenuOption[ ] optList = new MenuOption[ ]{new MenuOption("booking", "Booking", "If you're calling about a booking you've just made, press "),
                                              new MenuOption("account", "Account", "If you'd like to talk to us about your frequent flyer account, press "),
                                              new MenuOption("baggage", "Baggage", "If you're trying to locate lost baggage, press ")};


   public VoiceElementConfig getConfig(String name,
                                       ElementAPI elementAPI,
                                       VoiceElementConfig cfg) throws AudiumException
   {
       AudioGroup init_audgrp = cfg.new AudioGroup("initial_audio_group", false);

       StaticAudio tts = cfg.new StaticAudio(prompt, null);
       init_audgrp.addAudioItem(tts);
       init_audgrp.setBargein(true);
       cfg.setAudioGroup(init_audgrp);

       for (int i = 1; i <= optList.length; i++)
       {
          MenuOption m = optList[i - 1];

          StaticAudio opt_tts = cfg.new StaticAudio(m.prompt + i, null);
          init_audgrp.addAudioItem(opt_tts);



       }

       return cfg;
    }
}
```

Create configuration sample data as an array of menu option objects containing text strings. Data could come from anywhere you choose.

Implement getConfig method

Create audio group and add the first prompt

For each menu option add its audio prompt to the group

Voice Element - 3_Option_Menu

General | Settings | Audio | Data | Local Hotlinks

```
public class CL13DynamicMenu implements VoiceElementInterface {

// Dynamically configues the menu element from an array of data objects, each one
// specifying the items required for a single menu option. In this example the
// data is statically defined but it could be retrieved from anywhere.

    String prompt = "Please select one of the following options";

    MenuOption[ ] optList = new MenuOption[ ]{new MenuOption("booking", "Booking", "If you're calling about a booking you've just made, press "),
                                             new MenuOption("account", "Account", "If you'd like to talk to us about your frequent flyer account, press "),
                                             new MenuOption("baggage", "Baggage", "If you're trying to locate lost baggage, press ")};


    public VoiceElementConfig getConfig(String name,
                                        ElementAPI elementAPI,
                                        VoiceElementConfig cfg) throws AudiumException
    {
        AudioGroup init_audgrp = cfg.new AudioGroup("initial_audio_group", false);

        StaticAudio tts = cfg.new StaticAudio(prompt, null);
        init_audgrp.addAudioItem(tts);
        init_audgrp.setBargein(true);
        cfg.setAudioGroup(init_audgrp);

        for (int i = 1; i <= optList.length; i++)
        {
            MenuOption m = optList[i - 1];

            StaticAudio opt_tts = cfg.new StaticAudio(m.prompt + i, null);
            init_audgrp.addAudioItem(opt_tts);

            cfg.setSettingValue("option" + i + "_value", m.value);
            cfg.setSettingValue("option" + i + "_voice", m.voice);
            cfg.setSettingValue("option" + i + "_dtmf", Integer.toString(i));
        }

        return cfg;
    }
}
```

Create configuration sample data as an array of menu option objects containing text strings. Data could come from anywhere you choose.

Implement getConfig method

Create audio group and add the first prompt

For each menu option add its audio prompt to the group

For each menu option set the DTMF, voice and value settings

Voice Element - 3_Option_Menu

General | Settings | Audio | Data | Local Hotlinks

```
public class CL13DynamicMenu implements VoiceElementInterface {

// Dynamically configues the menu element from an array of data objects, each one
// specifying the items required for a single menu option. In this example the
// data is statically defined but it could be retrieved from anywhere.

    String prompt = "Please select one of the following options";

    MenuOption[ ] optList = new MenuOption[ ]{new MenuOption("booking", "Booking", "If you're calling about a booking you've just made, press "),
                                              new MenuOption("account", "Account", "If you'd like to talk to us about your frequent flyer account, press "),
                                              new MenuOption("baggage", "Baggage", "If you're trying to locate lost baggage, press ")};
```

Create configuration sample data as an array of menu option objects containing text strings. Data could come from anywhere you choose.

```
<form id="start”>
    <field name="choice_fld" modal="false”>
      <property name="inputmodes" value="dtmf voice" />

      <prompt bargein="true">Please select one of the following options If you're calling about a booking you've just made, press 1 If
   you'd like to talk to us about your frequent flyer account, press 2 If you're trying to locate lost baggage, press 3</prompt>

      <option value="booking" dtmf="1">Booking</option>
      <option value="account" dtmf="2">Account</option>
      <option value="baggage" dtmf="3">Baggage</option>

      <filled>
         ...
      </filled>
    </field>
  </form>
```

In this case, the DTMF input mapping was assigned implicitly when we iterated through the menu options list

```
        }

        return cfg;
    }
}
```