AsteriskCTI

Project documentation

Authors:
Bruno Salzano
Niksa Baldun

Todo:

- *Complete AsteriskCTI Client part*
- *Add protocol command to poll queue status*

Summary

1 Introduction

1.1 Overview

In my idea AsteriskCTI should be basically a multi-platform “easy to use” client/ server CTI solution integrated with the Open Source Digium’s Asterisk PBX Server.

C.T.I. is an acronym of Computer Telephony Integration: so a CTI is a software that makes possible to transfer from the telephone interface to the client PC a set of useful informations like calling party telephone number, digits grabbed by IVRs and so on. This information should be made available to a client software for database queries and/or other usage.

Actually, the advent of Web 2.0 technologies has to reflect on the concept of CTI: today a lot of back-office softwares runs like web pages into browsers, and thus the most important innovation that a new CTI platform can carry is the web integration. Call Data generated on the PBX logic should be passed to a middle-ware able to perform the most disparate operations such as execute an external application or open a browser, allowing to transfer them the variables as telephone parameters.

This new model make possible to build callcenter logic without the knowledge of complex API manageable only with advanced C/C++ skills, like TAPI.

Finally, this application should be available and runnable, client and server, on all (or almost all) modern operating systems: to achieve this objective I’ve selected, after trying various solutions (like mono/.net on GTK), to use the QT toolkit and – by consequence - C++ language. QT seems to be very easy to learn, very quick and stable, really multiplatform and now also with a multiplatform IDE. The same sources can be ported from Win32 to Linux just by compiling and packaging them.

2 Architecture

2.1 Overview

By initial design, AsteriskCTI is composed of various parts: the core is Digium’ Asterisk PBX with it’s AMI (Asterisk Manager Interface). The Asterisk Manager allows a client program to connect to an Asterisk instance and issue commands or read PBX events over a TCP/IP stream. The AMI is well documented on voip-info.org (check the pages at the url http://www.voip-info.org/wiki/view/Asterisk+manager+API).

Actually, various mini-CTI application that make use of AMI opens a lot of AMI connections, once per client: this approach is not optimal because it may slow down Asterisk PBX, multiply network traffic (don’t forget that a lot of PBX events flow over AMI) and generally decentralize the application logic on the CTI Client.

For this reason, AsteriskCTI Server should provide a daemon that opens just one AMI connection and listens for AsteriskCTI Clients connections. This means that on large installations, AsteriskCTI server can be run on a server different from the one that runs Asterisk PBX: calls, queues, IVRs runs separate from CTI server and without a great waste of resources. On small installations both software may run on a single Linux server lowering TCO.

The AsteriskCTI Server should also be able to watch dial-plan flow and match the events that carry in variables and store them for notifying the client later. This information will be collected on a per-call basis: each call will be tracked as an object.

For client integration, AsteriskCTI should provide a Client software that has to be installed on all PCs of the call-center. The AsteriskCTI Client will be paired to the seat’s telephone by the phone's extension number: the Client will know which phone the server should monitor for incoming calls by its configuration.

When the call is answered by that phone, the paired Client will perform the actions preconfigured on the Server.

This allows another reflection: the client does not need to be notified about all calls, but only the ones that arrive on a predetermined telephone service (this means a specified call queue).

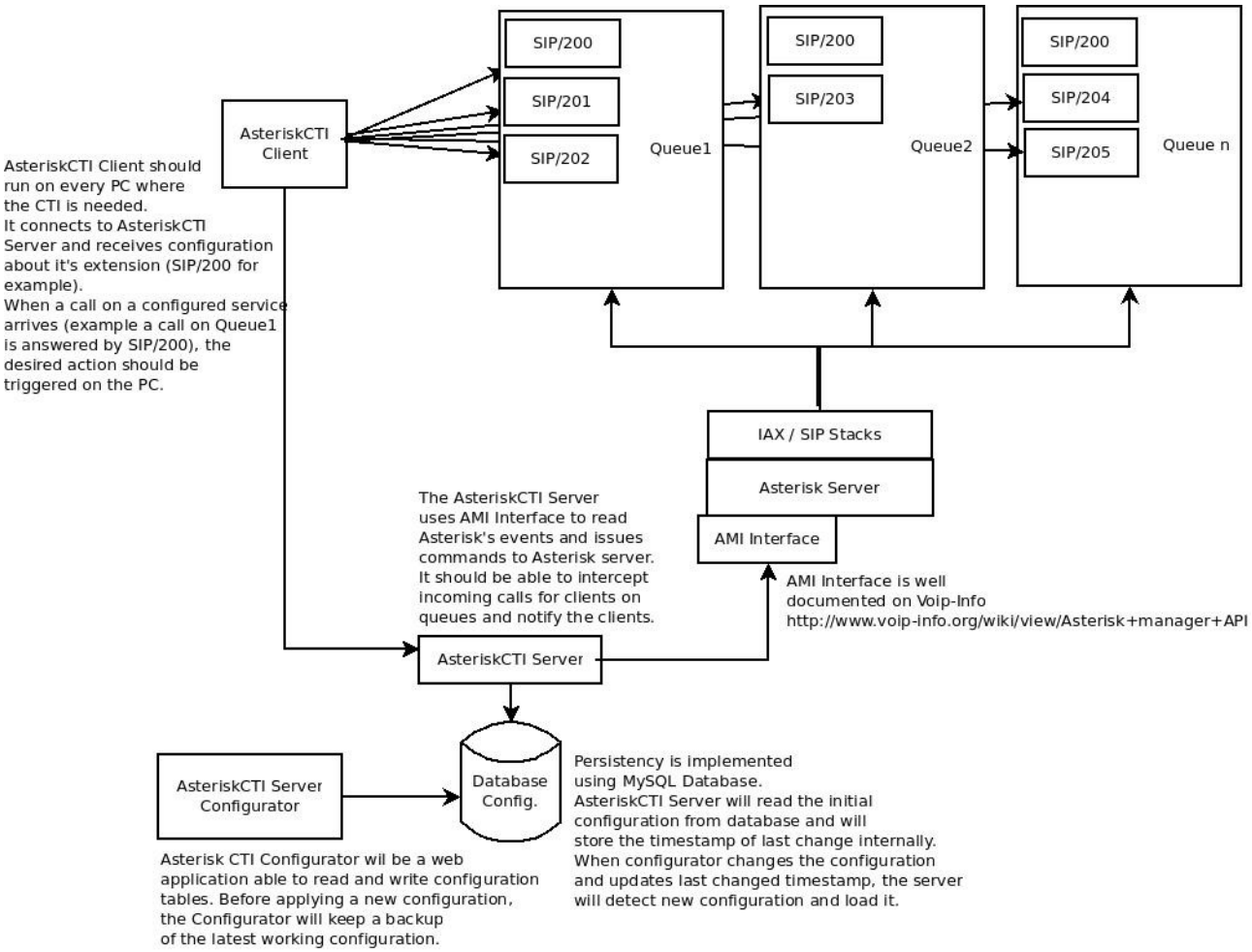

Figure 1 – Diagram

The last part of our software is a AsteriskCTI Configurator that will be a web application that reads and stores CTI configuration in a MySQL database. AsteriskCTI Server will load all the configuration schema in memory each time it finds that the configuration timestamp has been changed.

Summary:

1) Asterisk PBX – core telephony.
2) AsteriskCTI Server – uses AMI to read PBX events.
3) AsteriskCTI Client – Connects to AsteriskCTI Server and receives incoming call notifications to popup applications or web pages.
4) Asterisk CTI Configurator – Used to configure users, seat, operators, services and variables.

2.2 Architecture details

2.2.1 Configuration

The main configuration objects are “services”. With the term “Service” I mean a telephone service configured in Asterisk’ dial-plan, in the `extensions.conf` file.

You can learn more about Asterisk’ dial-plan on voip-info.org page: http://www.voip-info.org/tiki-index.php?page=Asterisk%20config%20extensions.conf

In Asterisk PBX, each managed call will flow through predefined contexts. The “context” where the call reaches an extension is what I define the “Service”.

AsteriskCTI should be able to manage two kind of “Service”:

a) Service without queue – this service, after calldata, will send the call to specified extensions using Dial application.

An example service without queue could be the following excerpt from `extensions.conf`:

```
[astcti-service]
exten => 101,1,Answer()
exten => 101,2,Read(cdata|/var/lib/asterisk/sounds/astctidemo/enter_five_digits|5|)
exten => 101,3,Set(calldata=${cdata})
exten => 101,4,Dial(SIP/201&SIP/202,30,m);
exten => 101,5,Hangup()
```

In this example the service is the context called “astcti-service”.

At priority1, we Answer the call.
At priority 2, we play a sound file and wait for 5 DTMF tones. The tones received are stored in “cdata” variable.
At priority 3, we initialize - through the Set dial-plan command - our variable called calldata to the value of “cdata”.
At priority 4, we Dial two internal extensions (SIP/201 and SIP/202) for 30 secs.
At priority 5, we finally hangup.

b) Service with queue – this service, after calldata, will send the call to a specified queue.

An example service with queue could be the following excerpt from `extensions.conf`:

```
[astcti-service]
exten => 101,1,Answer()
exten => 101,2,Read(cdata|/var/lib/asterisk/sounds/astctidemo/enter_five_digits|5|)
exten => 101,3,Set(calldata=${cdata})
exten => 101,4,Queue(astcti-queue|htw|||60);
exten => 101,5,Hangup()
```

In this example the service is the context called “astcti-service”.

At priority1, we Answer the call.
At priority 2, we play a sound file and wait for 5 DTMF tones. The tones received are stored in “cdata” variable.
At priority 3, we initialize - through the Set dial-plan command - our variable called calldata to the value of “cdata”.
At priority 4, we queue the call in a queue called “astcti-queue”.
At priority 5, we finally hangup (the call exits from queue anyway).

For both kind of services, when an incoming call arrives, the call context is always read on the “Newexten” event when Application field is set to “Answer”

```
Event: Newexten
Privilege: call,all
```

```
Channel: SIP/200-0895e728
Context: astctidemo
Extension: 101
Priority: 1
Application: Answer
AppData:
Uniqueid: asterisk-1231258004.4
```

And in both cases, the Hangup is the final event we read to remove the call from AsteriskCTI Server's stack.

What is different between the two cases above, is the way the call flows after the Set application. In the first case (service without queue) we read “Newexten” event with “Dial” application where the Uniqueid is set to the value of the Uniqueid of the service call context.

```
Event: Newexten
Privilege: call,all
Channel: SIP/200-08963918
Context: astctidemo
Extension: 100
Priority: 4
Application: Dial
AppData: SIP/201
Uniqueid: asterisk-1231258664.8
```

After that, a new channel is created in down state and a “Dial” event is generated where SrcUniqueID is the Uniqueid of the service call context and DestUniqueID contains the newly created channel:

```
Event: Dial
Privilege: call,all
Source: SIP/200-08963918
Destination: SIP/201-08961110
CallerID: 200
CallerIDName: Demo 1
SrcUniqueID: asterisk-1231258664.8
DestUniqueID: asterisk-1231258666.9
```

If the channel is answered we finally get a “Link” event where the two channels are bound.

```
Event: Link
Privilege: call,all
Channel1: SIP/200-08963918
Channel2: SIP/201-08961110
Uniqueid1: asterisk-1231258664.8
Uniqueid2: asterisk-1231258666.9
CallerID1: 200
CallerID2: 100
```

When call is on a service with queue, the call flow is slightly different. In fact, the call in our service context will always join a queue and leave it (because some extension answered or because the caller hanged up).

```
Event: Join
Privilege: call,all
Channel: SIP/200-0895e728
CallerID: 200
CallerIDName: Demo 1
Queue: astcti
Position: 1
Count: 1
Uniqueid: asterisk-1231258004.4
```

On the Join event, AsteriskCTI Server can check queue name for the active call and see if there's a corresponding configured service. At this point, when the call is finally answered and a Link event occurs, the called extension is notified about related action.

```
Event: Link
Privilege: call,all
Channel1: SIP/200-0895e728
Channel2: SIP/201-089639a0
Uniqueid1: asterisk-1231258004.4
Uniqueid2: asterisk-1231258006.5
CallerID1: 200
CallerID2: 200
```

CallerID and CallerIDName variables are set on Dial event and on Join event. We need to read them from there.

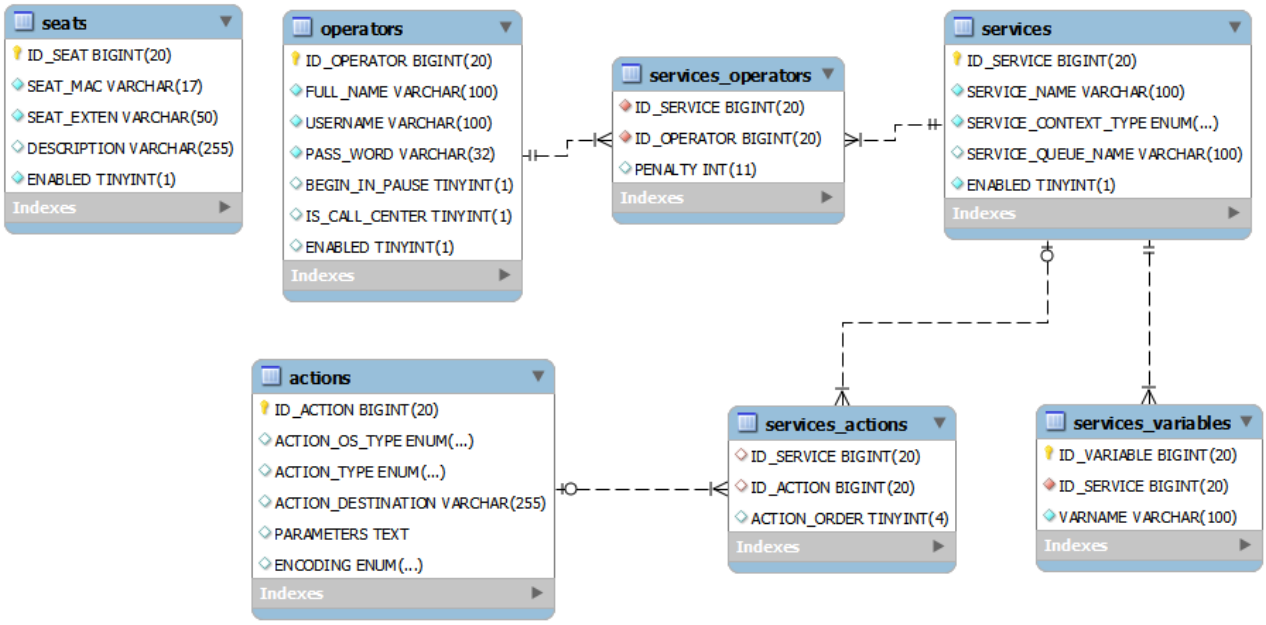

Figure 2 – Database configuration schema

For each single service, you can define:

- "SERVICE_NAME" reflects the context name – that should match the one, configured on asterisk dialplan, where the call will reach the operator's extension;
- "SERVICE_CONTEXT_TYPE" is the type of the context (inbound: for inbound calls; outbound, to let the user originate calls on outbound services and lock the outbound call feature when an inbound context is selected);
- "SERVICE_QUEUE_NAME" – the name of the queue associated with this context. If this field has a value, the AsteriskCTI Server will dynamically add operators' extensions to the Queue named "SERVICE_QUEUE_NAME" after their login, and remove them after their logout.
- "ENABLED" – if this context should be monitored by the AsteriskCTI Server.

Services are associated with actions via "services_actions" table. For each action, you can define:

- Path and parameters for different actions different operating systems. So "ACTION_OS_TYPE" will determine the Operating System(s) for which we want that specific action, identified by the "ACTION_DESTINATION", to be opened with "PARAMETERS" string.
- "ACTION_TYPE" - this parameter is used to determine if and what action to perform when a call for this context is matched: "Application" for external applications, "Internal Browser" to popup an

AsteriskCTI Browser window, “External browser” to open default browser, “TCP message” and “UDP message” to send a message to external server.

- In particular, “PARAMETERS” can contains variables that AsteriskCTI Server will read from Asterisk Server. The variables watched by the AsteriskCTI are defined in the “Service_Variables” table. You can use the following notation to expand variables into the parameters string: key={VAR_NAME} where “VAR_NAME” is the name of the variable defined in “Service_Variables”.
- “ENCODING” defines an encoding used for sending messages to an external server.

Call-center seats are defined in “Seats” table: each seat is identified by the MAC Address of the PC the Client application will run on. On startup, the AsteriskCTI Client will send the HW MAC Address so the AsteriskCTI Server will associate that connection with the related extension in the table. Non-callcenter operators can identify with an extension instead of MAC address.

Operators are defined in the “operators” table. An operator can be on a single seat. Each operator is bound to services through the “service_operators” table. One operator can be associated with zero or more services. The “penalty” field will be used for services of type “queue-inbound”: when adding operators to queue dynamically, AsteriskCTI server will use this value to assign queue penalty to operators.

This simple database schema should be managed from the AsteriskCTI Configurator. From the configurator, it should be possible to save the runtime configuration in a MySQL database. After that, the configurator will update the configuration timestamp field, which will notify the server about change.

2.2.2 AsteriskCTI Server

2.2.2.1 Goals of AsteriskCTI Server

The server has to be a multi-threaded application with at least:

- one thread to manage the server configuration watcher
- one thread to manage AMI TCP/IP socket connection and protocol parsing
- one thread to manage server socket for client connections
- zero or more threads to manage client connections

At the startup, the server will perform base checks (configuration for example) and if all is OK, it will try to connect to AMI. In case of failure, the server will retry to connect for 0 to infinite times (configurable) with pause of 5 or more seconds (configurable) between each try.

After successful AMI connection, the server will begin to accept client connections and spawn new threads for each client connection.

2.2.2.2 AMI Connection and Call Management

AMI Connection is built on AsteriskCTI startup. Once connected, AsteriskCTI will send a Login Action to the AMI console to authenticate as a manager and read all subsequent Asterisk events.

Not all Asterisk events are significant for CTI. For our purpose, we need to intercept and manage the following Asterisk Events:

Newchannel, Dial, Newstate, OriginateResponse, Newexten, Link, Hangup.

We should also implement the following AMI Commands:

QueueAdd, QueueRemove, QueuePause, Originate.

Let’s see this in detail.

2.2.2.3 Events

All Asterisk Events are related to one or more Uniqueid: this means that we can build an object (named AsteriskCall) that stores all the call information when a new Uniqueid is generated (Newchannel) and destroy that object when the call is destroyed (Hangup).

All AsteriskCall objects will be referenced with the Uniqueid and stored in a Hash for quick save and retrieve operations. We can parse Source and Destination of each Call and store only calls that match our registered extensions. When a CTI event is matched against a registered extension, that Event will be then sent to the right AsteriskCTI Client.

Newchannel - This event always happens when a new channel is created , so it is generated every time a call begins. When this event is intercepted by the AsteriskCTI Server, a new AsteriskCall object has to be built.

Dial – Someone is Dialing a call. This is the right place to intercept CallerId on calls from extensions. When a call is dialed, it means that the call doesn't come from a queue. So we shouldn't check callerid on "Newstate" event when State is "Ringing".

Newstate – This event reflects Asterisk's call status change. When we match the State "Ringing" and the call has not been dialed, we can grab the CallerId and generate a "Callerid" event to AsteriskCTI Client. We should always save the new AsteriskCall State.

OriginateResponse – This Event is generated when AsteriskCTI Client asks AsteriskCTI Server to send an "Originate" command to the Asterisk PBX. Asterisk PBX will parse the originate command and then will send a Response. When the Response field is "Success" we can generate a new AsteriskCall object with the Uniqueid and Context passed by OriginateResponse.

Newexten - This event is generated every time a new step in the dialplan is done: a new extension of type Set may contain our call data. The AsteriskCTI Server should verify if the Application event is "Set" and if the "Channel" field matches a registered extension. In this case another check is requested: AsteriskCTI Server should read AppData and split key=value pair. If "key" matches a variable registered for the "Context" specified in the event, we can save AppData in the AsteriskCall Object. So then the call is sent to the AsteriskCTI Client.

Link – This event is important, because two calls are bound in the Asterisk PBX. Here we should notify the client if the call matches registered extensions. On the Link event we receive two Uniqueids: one for the caller' call and one of the called call.

Hangup - When an hangup event is fired, we have to remove the call with the matching uniqueid .

Events related to Queues need to be commented watching related Actions:

Action	Dir	Response/Event	Description

`Action: QueueAdd`
`Queue: astcti`
`Interface: SIP/201`
`Penalty: 1`
`Paused: false`
	→		*We ask to dynamically add an interface (phone, extension, other) on a specified queue. AsteriskCTI Server should automatically add the extension related to an user on all the queues (services) configured (see the services_operators table)*

	←	`Response: Success | Error`	*Asterisk response is*

		Message: [describe success or error status]	*immediate: Success or Error*
		Event: QueueMemberAdded Privilege: agent,all Queue: astcti Location: SIP/201 MemberName: SIP/201 Membership: dynamic Penalty: 1 CallsTaken: 0 LastCall: 0 Status: 1 Paused: 0	*After the response, an Event is generated: QueueMemberAdded.* *We can wait for these events and then notify the AsteriskCTI Client that it is on that queue*
Action: QueuePause Queue: astcti Interface: SIP/201 Paused: true	→		*When AsteriskCTI Client asks to go on pause, AsteriskCTI Server will send a QueuePause command on AMI. On the Pause request, we send a "PENDING PAUSE REQUESTED" to AsteriskCTI Client. The Pause will be effective if Reponse is Success and after QueueMemberPaused event.is received*
	←	Response: Success | Error Message: [describe success or error status]	*Asterisk response is immediate: Success or Error*
	←	Event: QueueMemberPaused Privilege: agent,all Queue: astcti Location: SIP/201 MemberName: SIP/201 Paused: 1	*After the response, an Event is generated: QueueMemberPaused.*
		Event: QueueMemberStatus Privilege: agent,all Queue: astcti Location: SIP/201 MemberName: SIP/201 Membership: dynamic Penalty: 1 CallsTaken: 0 LastCall: 0 Status: 5 Paused: 1	*Status 5 = not available for calls, it's device related. Please check here* *Maybe we can skip this Event?*
Action: QueuePause Queue: astcti Interface: SIP/201 Paused: false	→		*Coming back from pause.*
	←	Response: Success | Error Message: [describe success or error status]	
	←	Event: QueueMemberPaused Privilege: agent,all Queue: astcti Location: SIP/201 MemberName: SIP/201 Paused: 0	
Action: QueueRemove Queue: astcti Interface: SIP/201	→		*When AsteriskCTI Client disconnects from AstCTI Server for any reason, we should dynamically remove the associated interface*

			from queues.
	←	Response: Success | Error Message: [describe success or error status]	
	←	Event: QueueMemberRemoved Privilege: agent,all Queue: astcti Location: SIP/201 MemberName: SIP/201	

2.2.2.4 Commands

QueueAdd – This command is used to dynamically add extensions on queues. Example:

```
Action: QueueAdd
Queue: [queue-name]
Interface: [technology/extension] (ex. SIP/200)
Penalty: [operator penalty]
Paused: [true/false] if the interface is initially paused
```

With this command AsteriskCTI Server can dynamically login call center operators on the configured queues.

QueueRemove – With this command AsteriskCTI should remove callcenter operators on AsteriskCTI request or disconnect. Example:

```
Action: QueueRemove
Queue: [queue-name]
Interface: [technology/extension] (ex. SIP/200)
```

QueuePause – This command will be used to pause / unpause an operator. Queue parameter is optional and in fact we don't need to specify it for our purpose: when omitted, the operator will be paused in all queue it belongs to. Example:

```
Action: QueuePause
Queue: [queue-name]
Interface: Member [technology/extension] (ex. SIP/200)
Paused: true / false
```

Originate – This command will be used to generate a call. [TO COMPLETE]

```
Action: Originate
Channel: Channel on which to originate the call (The same as you specify in the
Dial application command)
Exten: Extension to use on connect (must use Context & Priority with it)
Context: Context to use on connect (must use Exten & Priority with it)
Priority: Priority to use on connect (must use Context & Exten with it)
Application: Application to use on connect (use Data for parameters)
Data: Data if Application parameter is used
Timeout: Timeout (in milliseconds) for the originating connection to
happen(defaults to 30000 milliseconds)
CallerID: CallerID to use for the call
Variable: Channels variables to set (max 32). Variables will be set for both
channels (local and connected).
Async: true|false
```

Sequence of events: first the Channel is rung. Then, when that answers, the Extension is dialed within the Context to initiate the other end of the call. **Note** that the Timeout only applies to the initial connection to the Channel; any timeout for the other end can be specified, for instance in a Dial command in the definition of the Context.

Using ***Async*** leads to an **OriginateResponse** event which contains the failure ***reason*** if any. Reason may be

one of the following:

0 = no such extension or number
1 = no answer
4 = answered
8 = congested or not available

2.2.3 Client management and Server protocol

2.2.3.1 Overview

Command sent from Client to the server are built with this structure:

```
[COMMAND_TOKEN] [PARAMETERS]
```

Each command should be terminated by a '\n' newline.

The server will send it's response or notification to the client with the following structure:

```
[NUMERIC_RESPONSE] [COMMAND_TOKEN] [INFORMATION/PARAMETERS]
```

Each command will be terminated by a '\n' newline. When the server have to notify client about Asterisk Events, it will send an XML formatted object.

So, when the client receive a message that not begin with a [NUMERIC_RESPONSE] part, it should consider it like XML message (see below for example).

2.2.3.2 Client / Server Protocol

AstCTIServer	Dir	AstCTIClient	Description
Handshake and authentication			
`100 WELCOME`	→		This will begin the handshake process
	←	`CMPR`	The Client requests compression level for further communication
`103 [level]`	→		The server sends its compression level (0-9). Level 0 means no compression. The value -1 (representing zlib default level) is not allowed.
	←	`USER [username]`	The Client sends the Username and wait for a response
`100 OK`	→		If the Command has completed successfully on server
`101 [reason]`	→		If the Command has not successfully completed on server.
	←	`PASS [md5pass]`	The client sends an md5 encrypted password to the server
`102 OK`	→		The server authenticated the client successfully. The AsteriskCTI Server should dynamically add the extension associated with the client on all services configured
`101 [reason]`	→		If the Command has not successfully completed on server.
	←	`KEEP`	The Client requests the interval in which to send keep-alive messages
`104 [time]`	→		The server sends the maximum time (in

			milliseconds) it will wait in idle state before terminating the connection
	←	CHPW [md5oldpass] [md5newpass]	After a successful authentication, the client may send this command to update the password of the connected operator. The client is responsible for verifying the validity and quality of the new password.
100 OK	→		Password changed successfully
101 [reason]	→		The server sends an error if the operator is not authenticated or if the some other error occurred.
Mac Address			
	←	MAC [00:00:00:00:00:00]	The client notifies the AsteriskCTI server about the MAC address of the seat. Client must identify with MAC address or extension, but not both.
100 OK [extension]	→		If the Command has completed successfully on server, the seat extension is sent
101 [reason]	→		If the Command has not successfully completed on server.
Extension			
	←	EXTEN [extension]	The client notifies the AsteriskCTI server about the extension of the seat. Client must identify with MAC address or extension, but not both.
100 OK [extension]	→		If the Command has completed successfully on server, the seat extension is sent
101 [reason]	→		If the Command has not successfully completed on server.
Disconnection			
	←	QUIT	The Client sends a disconnect command.
900 OK	→		The Server sends this response and then closes the TCP connection
Keepalive			
	←	NOOP	The Client should send this command to keep the connection alive, avoiding socket timeout. If the server does not receive a NOOP command at predefined (configurable) time, the socket will be closed
100 OK	→		This is sent before the AsteriskCTI Server closes the connection to the client.
Originate (let the client originate a call from asterisk server)			
	←	ORIG [number to call]	The client asks the AsteriskCTI server to generate an Originate command on the Asterisk Server. The Originate command will
100 OK	→		If the AsteriskCTI server successfully sent data to asterisk server.
101 [reason]	→		If some parameter is missing or unsuccessful communication with asterisk server.
Services and queues			
	←	SERVICES	AsteriskCTI Client asks for it's available services
200 service-name service-context [inbound|	→		AsteriskCTI Server will send the list of services, one at a time. Each item received by the client will contain service name, context and the type of the context (inbound/outbound)

outbound]			
201 eof	→		This means that the services list is at an end
		QUEUES	
202 [queue-name]	→		AsteriskCTI Server notifies the client that it is available/unavailable on a specified queue (service).
203 eof	→		This means that the queues list is at the end
Pause requests / responses			
	←	PAUSE IN	AsteriskCTI Client requests a pause
301 PAUSE PENDING	→		AsteriskCTI Server have submitted pause request to AMI.
300 OK	→		Pause confirmed
302 [reason]	→		There was an error
	←	PAUSE OUT	
300 OK	→		Out of pause
302 [reason]	→		There was an error
Other commands			
	←	IDEN	Get the seat extension
100 OK [extension]	→		The seat extension is sent
101 [reason]			If the Command has not successfully completed on server.

2.2.3.3 *Structure of XML Event*

The XML Events are structured as following:

```
<event id="">
   <call>
          <uniqueid>[asterisk call uniqueid]</uniqueid>
          <channel>[channel of the event]</channel>
          <calleridnum>[callerid]</calleridnum>
          <calleridname>[callername]</calleridname>
          <context>[context of the call]</context>
          <appdata>
                 <data key="[key-of-appdata]">[value-of-appdata]</data>
          </appdata>
          <state>[channel state]</state>
   </call>
</event>
```

Example:

```
<event id="Newchannel">
   <call>
          <uniqueid>1094154427.10</uniqueid>
          <channel>SIP/101-3f3f </channel>
          <calleridnum>102</calleridnum>
          <calleridname>Demo 2</calleridname>
          <context>asteriskcti-demo</context>
          <appdata>
                 <data key=""></data>
          </appdata>
          <state>Ringing</state>
   </call>
</event>
```

2.2.4 AsteriskCTI Client

AsteriskCTI Client is a desktop application that will perform the actual CTI functions: it will connect to AsteriskCTI Server and will receive notifications about caller, service and application to start.
It's key features:
1) Multiplatform software - This application should be able to compile and run on Windows®, GNU/Linux, Apple MacOS X®. Using QT framework, it's theoretically also possible to have the CTI Client running on mobile devices.
2) Internationalization – The client should give support to various translations. The GUI should be able to render various international character sets and translations.
3) Easy To Use – The Client Graphical User Interface should show immediately the basic actions, like Login, Pause, Configuration. An Icon in the tray area will be the way to hide/show the application, and leave the desktop free for other applications.
4) Integrated web browser – The application will have it's own easy web browser. Where special features like Flash® plugin or Java® plugin are not needed , a simple web browser can be the right choice to show web enabled contents.
5) Multiple visual layouts in order to support different usage cases.

2.2.4.1 Call center

This should be a minimalistic layout in which user will, for the most part, be guided by predefined services.

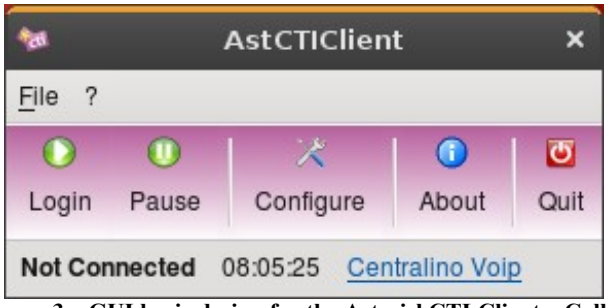

Figure 3 – GUI basic design for the AsteriskCTI Client – Call center

The MainScreen should display: Login/Logout, Pause, Dial and Quit buttons. Configure and About can be moved respectively in “File” menu and “?” menu. A status bar will show Connection Status, Operator' status and actual date and time.

The Dial button should be a dropdown menu button that will show a textbox field and a service combo box filled with outbound services on which the operator is active. The list of active outbound services is received at the startup from AsteriskCTI Server. Notifications about available outbound services can be received at runtime and the combo items should be updated as requested.

The MainScreen can be minimized with an Icon into the Tray Area. When in Pause, the Client should be not reachable by others and a dialog with a password request should appear. This dialog should be done as follows: a label that says “Pause is active. Please, when you return from Pause, enter your password and click the Unpause button.”. In the dialog, the “Unpause button” and an “Unlock” button should be avaiable. When pressed, the Unlock button should let another user to unlock the AstCTIClient providing a new set of username / password. The old operator should be logged out and the new operator logged in.

Client/Server communication, as shown in previous page of this document, should be used to communicate with AsteriskCTI Server and receive call notifications and other events.

2.2.4.2 Phone Manager

This layout should allow users to control their own telephone. Users should be able to place, answer, transfer, park and hold calls, listen to voice mail, record conversations, view call logs. All functions should be accessible by keyboard as well as mouse.

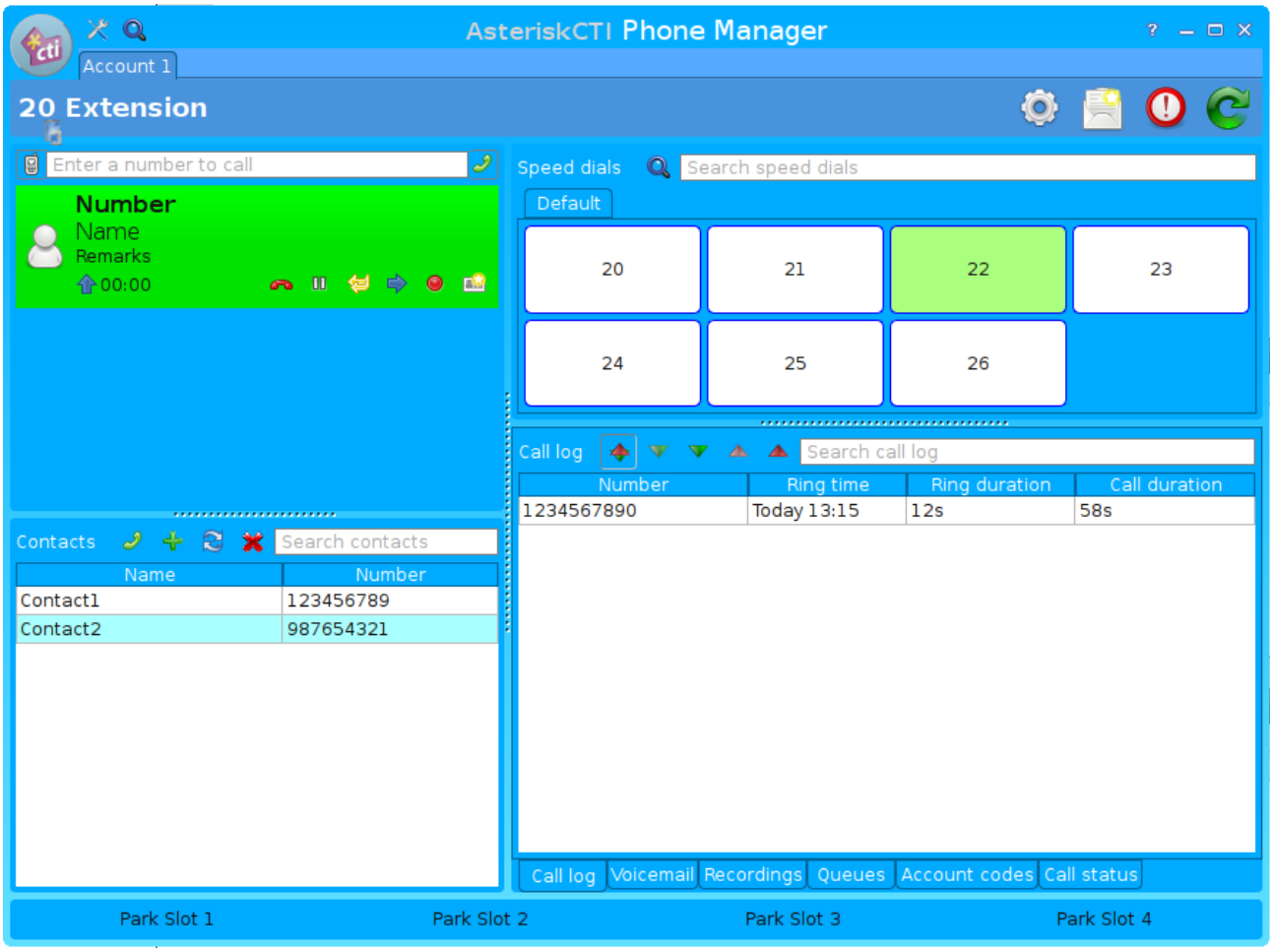

Figure 4 – GUI basic design for the AsteriskCTI Client – Phone Manager

The client should also be able to work in "telecommuter mode", meaning it can use another extension when answering and placing calls.

The client should support multiple phone accounts, with the ability to easily switch between them.

The main window should be split into several areas, with multiple features organized in tabs:

- Currently active calls
  User should be able to answer any call from the list and easily switch between calls
- Speed dial/Busy lamp field
  User should be able to see the status (free, busy, unavailable, do-not-disturb etc.) of particular extensions. This area can also be used for a quick call or transfer to desired extension. There should be support for multiple speed dial groups.
- Call log
  Call activity log (dialed, answered. missed calls).
- Voicemail
  Access to user's voicemail inbox, with the option to listen to voice files directly from PC.
- Recordings
  List of recorded conversations, with the option to listen to voice files directly from PC.
- Account codes
  List of account codes, with the ability to start the call with selected account code, or to apply the account code to existing call.
- Call status
  List of all currently active calls in the system

- Park area
  This area should display parking slots with ability to easily park and unpark calls.

Contact list of some kind should be provided, preferably via LDAP. Another desirable feature is integration with popular PIM software, such as Microsoft® Outlook® and Thunderbird®.

2.2.5 AsteriskCTI Configurator

2.2.5.1 Overview

AsteriskCTI Configurator should be a web-based software that will allow system configuration management.

Configuration parameters will be read and stored into a MySQL database. Before last configuration timestamp is updated, the Configurator should be able to provide a backup of old and working configuration. Once the configuration timestamp has been changed, the server will process the new configuration.

The application need to be Web 2.0, developed in PHP5, XHTML, AJAX. Database backend will be MySQL used through Phplens' ADOdb database classes.

Key features of this software should be: user-friendly GUI , full portability. I suggest the use jQuery as Javascript Library for AJAX and jQuery UI widgets to quickly build web forms and GUIs.

AsteriskCTI Configurator should be password restricted: to access configuration pages, system administrator should provide login credentials. After they log in, system administrators should be able to access the following four base modules:

- AsteriskCTI Manager Users
- Edit Operators
- Add Variables
- Services management

Main screen should have a Menubar to access main application modules.

Each management section should be composed of listviews to list entries that are already present and forms to manage entries fields. Toolbars and menus should be provided to access main functions such as “add”, “remove”, “edit”.

Double-click on entries in listview should trigger the “edit” action. Forms can be opened in modal dialogs. Each form is composed of required and non required fields: submitting a form will be checked server side and if required fields are empty, errors should be triggered and shown to the user.

Form's fields should be always checked for their type: alphanumeric data shouldn't be admitted in numeric fields, date fields should contains date and shouldn't be editable by hand etc. etc.

A menu should be provided to apply configuration from AsteriskCTI Configurator environment to production environment.

2.2.5.2 Login / Logout

As said before, we can have one or more users that may need to access AsteriskCTI Configurator.

The access to AsteriskCTI Configurator will be restricted by a login screen: the user will be prompted for username and password. After a login, the username will be kept in session and used for logging purposes. On the Login screen, login error (if any) will be displayed and a “password lost” recovery link should be present .

2.2.5.3 Manager Users

Manager users' information should be stored in a separate MySQL table. This information should include, at minimum, “full name”, “userid”, “password”, “creation date”, “enabled”.

User's login should be checked against userid and password from the manager users table.

2.2.5.4 Operators

This view should contain the list of call center operators. Each operator

2.2.5.5 Variables

This view should contain the list of AsteriskCTI Variables. We access the variable configuration from a "Service" detail. Each variable is defined as a name. For example, if a variable called "customer_id" is defined in Configurator, we need to have something like this in `extensions.conf`:

```
[astcti-service]
exten => 101,1,Answer()
exten => 101,2,Read(cdata|/var/lib/asterisk/sounds/astctidemo/enter_five_digits|5|)
exten => 101,3,Set(customer_id=${cdata})
exten => 101,4,Queue(astcti-queue|htw|||60);
exten => 101,5,Hangup()
```

2.2.5.6 Services

This view should let the manager user manage AsteriskCTI Services. From here, managers can add, modify or delete services.