AsteriskCTI Project

Authors:
Bruno Salzano

Todo:

- *Complete AsteriskCTI Client part*

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 make 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. These information should be made available to a client software for database queries and/or other usage.

Actually, the advent of Web 2.0 technologies have 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 logics 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 may slow down Asterisk PBX, multiplicate network traffic (don’t forget that on AMI flows a lot of PBX events) and generally decentralize on CTI Client the application logic.

For this reason, AsteriskCTI Server should provide a daemon that opens just one AMI connection and listen 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 runs 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 latter notify the client. These 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 have to be installed on all the PC of the call-center seats. The AsteriskCTI Client will be paired to the seat' telephone by the phone extension number: the Client will know by its configuration which phone the server should monitor for incoming calls. When the call is answered by that phone, the paired Client will perform the actions preconfigured on the Server.

This allows another reflection: not all the calls have to be notified from the server to the client: only the ones that arrive on a predetermined telephone service (this means a specified call queue).

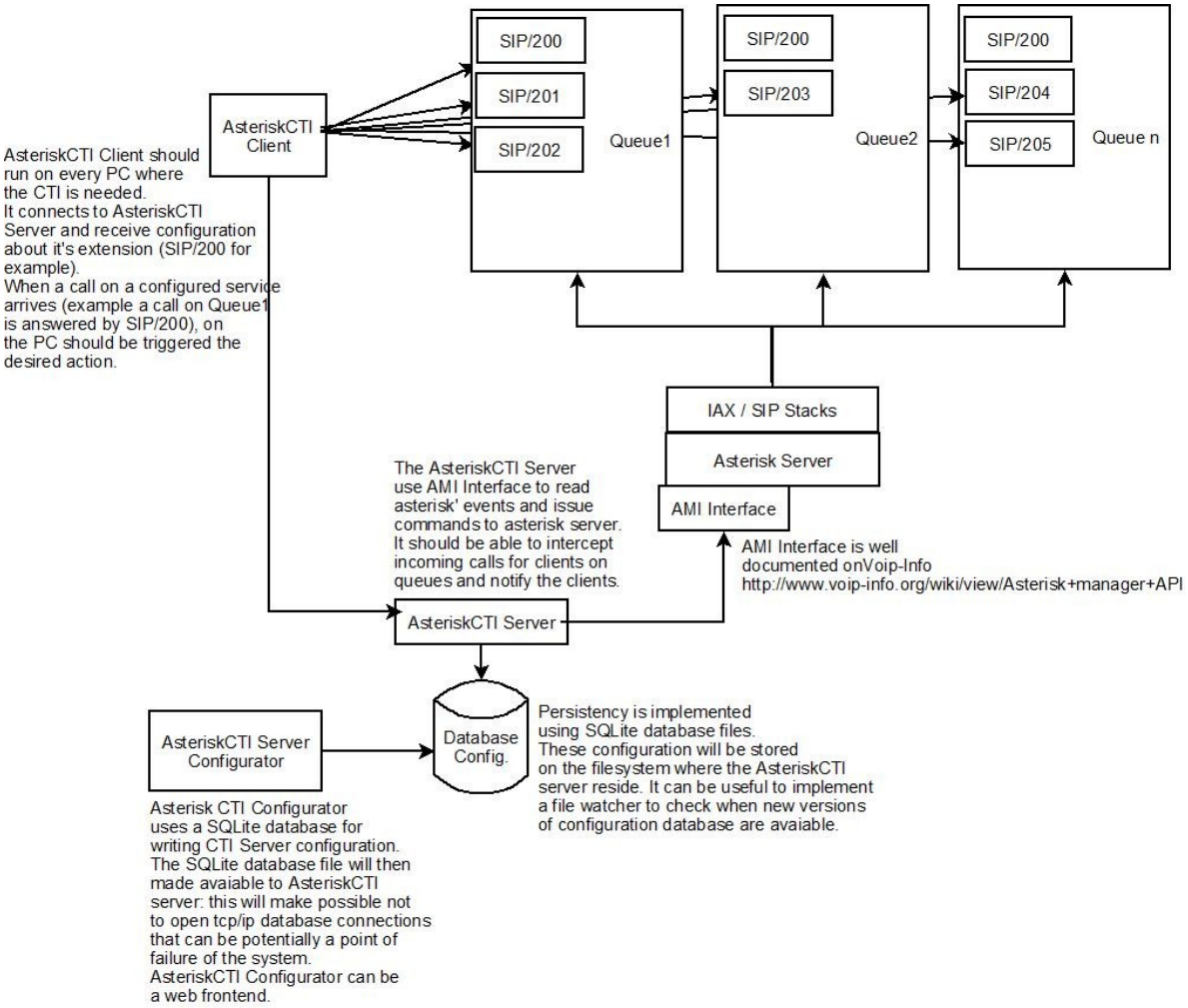

Figure 1 – Diagram

The last part of our software is a AsteriskCTI Configurator that can be a web application that reads and store CTI configuration in a database and persists them in a SQLite database file. This persistency will be used then by the AsteriskCTI Server to avoid that the a database disconnection can cause crashes.

Resume:

1) Asterisk PBX – core telephony.
2) AsteriskCTI Server – use AMI to read PBX events.

3) AsteriskCTI Client – Connects to AsteriskCTI Server and receive 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 on 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 reach 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 with a 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 sounds file and wait for 5 dtmf. The dtmf received are stored in a "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 sounds file and wait for 5 dtmf. The dtmf received are stored in a "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 exit from queue anyway).

For both kind of services, when an incoming call arrive, 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 case the Hangup is the final event we read to remove the call from AsteriskCTI Server' stack.

What is different between the two case above, is the way the call flow after the Set application: in the first case (service without queue) we read a "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 new 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 slight different. Infact, the call in our service context will always join a queue and left it (because some extension answered or because the caller hangup).

```
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, relative action is notified to the called extension.

```
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 Callerid variables are set on Dial event and on Join event. We need to read from there.

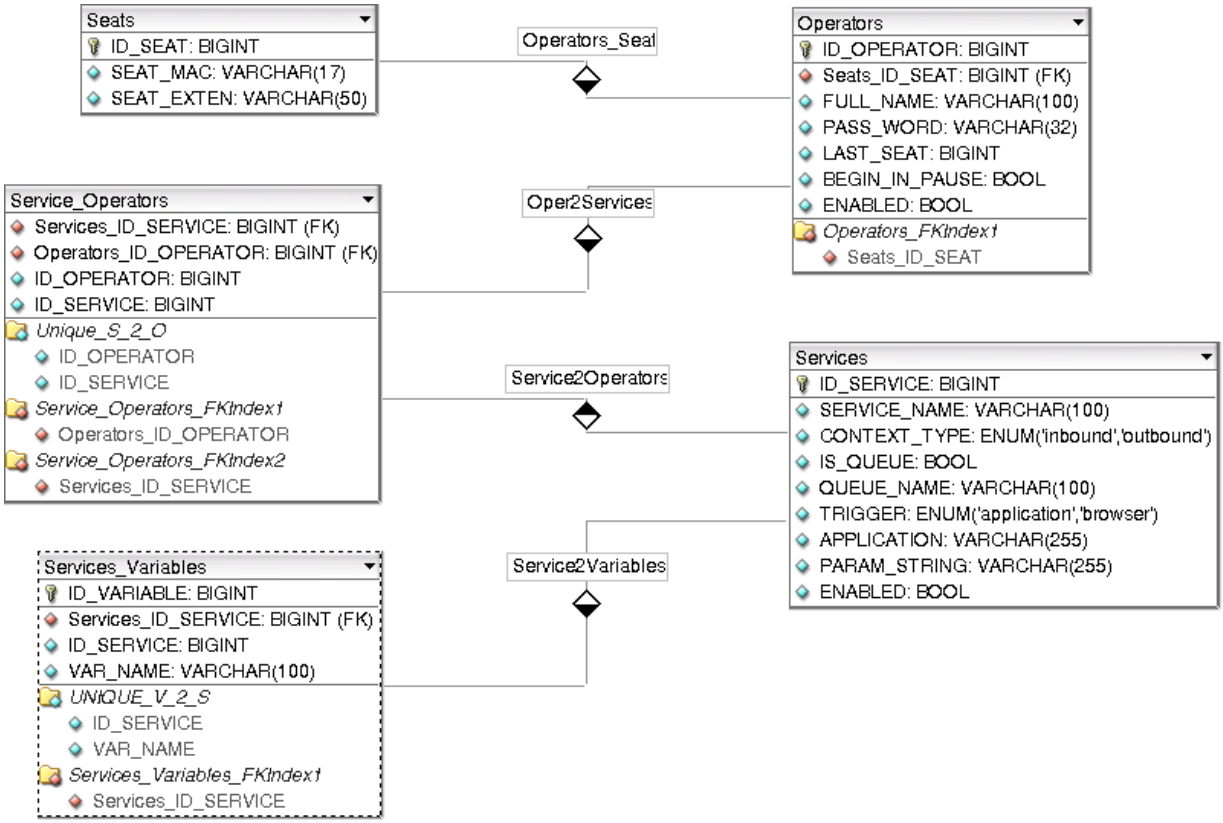

Figure 2 – Database configuration schema

For each single service, you can define:

- 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 selected an inbound context);
- the context name – that should match the one, configured on asterisk dialplan, where the call will reach the operator's extension;
- "is queue" – if this service have a queue
- "queue_name" – the name of the queue associated with this context. If "is_queue" is active and "queue_name" have a value, the AsteriskCTI Server will add dynamically operators' extensions to the Queue named in "queue_name" after their login, and remove them after their logout.
- "Trigger" is the type of application to popup when a call for this context is matched: "Application" for external applications; "Internal Browser" to popup an AsteriskCTI Browser window;
- "application" and "param_string" – the name or the full path of the application to run (when type is application) and the parameters to pass. Parameters can contains variables defined in "services_variables".
- "enabled" – if this context should be monitored by the Server

Each service can have zero or more variables. These are defined in "services_variables". This table contains only the variables' names because the value is read from the server from the dialplan!

Call-center seats are defined in "Seat" table: each seat is identified by the Mac Address of the PC the Client application will run. On startup, the AsteriskCTI Client will send the HW MAC Address so the AsteriskCTI Server will associate that connection with the relative extension in the table.

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 on zero or more services. The "penalty" field will be used for services of type "queue-inbound": when adding dynamically operators to queue, 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 an XML file. This file will persist the configuration and, as stated before, this will preserve the AsteriskCTI Server from opening database connection.

The file will have to be transferred some-way to the AsteriskCTI Server. I imagine a file copy service to the AsteriskCTI server configuration directory.

A file system watcher will then notify the server if the configuration file has changed. Once read the new configuration, and in case of no errors, the AsteriskCTI server will use the new configuration immediately.

2.2.2 AsteriskCTI Server

2.2.2.1 Goals of AsteriskCTI Server

The server have 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, 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 significative 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 details.

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 store all the call information when a new Uniqueid is generated (Newchannel) and destroy that object when the call is destroyed (Hangup).

All the AsteriskCall 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 notified to the right AsteriskCTI Client.

Newchannel - This event always happens when a new channel is created , so it is generated everytime a call begins. When this event is intercepted by the AsteriskCTI Server, a new AsteriskCall object have 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, 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 reflect a 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 ever save the new AsteriskCall State.

OriginateResponse – This Event is generated when AsteriskCTI Client ask 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 everytime a new step in the dialplan is done: a new extension of type Set may contains our calldata. The AsteriskCTI Server should verify if the Application event is “Set” and if the “Channel” field match a registered extension. In this case another check is requested: AsteriskCTI Server should read AppData and split key=value pair. If “key” match a variable registered for the “Context” specified on the event, we can save AppData on the AsteriskCall Object. So when the call is notified to the AsteriskCTI Client.

Link – This event is important, because two calls are bound from the Asterisk PBX. Here we should notify the client if the call match registered extensions. On the Link event we receive two Uniqueid: once for the caller’ call and once of the called call.

Hangup - When an hangup event is fired, we've to remove the call with the uniqueid matched.

Events related to Queues needs to be commented watching relative Actions:

Action	Dir	Response/Event	Description
`Action: QueueAdd` `Queue: astcti` `Interface: SIP/201` `Penalty: 1` `Paused: false`	→		*We ask to dinamically add an interface (phone, extension, other) on a specified queue. AsteriskCTI Server should automatically add the extension related to a user on all the queues (services) configured (see the services_operators table)*
	←	`Response: Success | Error` `Message: [describe success or error status]`	*Asterisk response is 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 these events and then notify the AsteriskCTI Client that he is on that queue*
`Action: QueuePause` `Queue: astcti` `Interface: SIP/201` `Paused: true`	→		*When AsteriskCTI Client asks to go in 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 received QueueMemberPaused event.
	←	`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. Eample:

```
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 infact we don't need to specify for our purpose: when omitted, the operator will be paused in all queue he belongs to. Example:

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

Originate – TODO

2.2.3 Client management and Server protocol

2.2.3.1 Overview

Command sents 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] [INFORMATIONS/PARAMETERS]
```

Each command will be terminated by a '\n' newline. When the server have to notify client of 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
	←	`USER [username]`	The Client sends the Username and wait for a response
`100 OK`	→		If the Command has completed successfully on server
`101 KO [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 successfully the client. The AsteriskCTI Server should dynamically add the extension associated with the client on all services configured
101 KO [reason]	→		If the Command has not successfully completed on server.
Mac Address			
	←	MAC [00:00:00:00:00:00]	The client notify to AsteriskCTI server the macadress of the seat.
100 OK	→		If the Command has completed successfully on server
101 KO [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 close the TCP connection
Keepalive			
	←	NOOP	The Client should send this command to keep alive connection avoiding socket timeout. If the server will not receive a NOOP command at predefined (configurable) time, the socket will be closed
100 OK	→		This is sent before the AsteriskCTI Server close the connection to the client.
Originate (let the client originate a call from asterisk server)			
	←	ORIG exten|cntxt|prio	The client will ask the AsteriskCTI server to generate an Originate command on the Asterisk Server. The Originate command need to know Extension (exten=number to call) on the specified context (cntxt) with the specified priority (usually 1)
100 OK	→		If the AsteriskCTI server successfully send data to asterisk server.
101 KO [reason]	→		If some parameter is missing or unsuccessful communication with asterisk server.
Avaiable on queues			
200 [queuename]	→		AsteriskCTI Server notify the client that he is available on a specified queue (service).
Pause requests / responses			
	←	PAUSE IN	AsteriskCTI Client request a pause
301 PAUSE PENDING	→		AsteriskCTI Server have submitted pause request to AMI.
300 OK	→		Pause accorded
302 KO [reason]	→		There was an error
	←	PAUSE OUT	

300 OK	→		Out of pause
302 KO [reason]	→		There was an error

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

[to fill]

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 SQLite database file. When the configuration is applied, this database file will be copied into the AsteriskCTI Server configurations folder with a unique filename. The server will provide to manage the new configuration.

The application need to be Web 2.0, developed in PHP5, XHTML, AJAX. Database backend will be SQLite 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 do a login before. After logged 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 already present entries 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 shouldn't admitted in numeric fields, date fields should contains date and shouldn't be editable by hand etc. etc.

Should be provided a menu 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 will be displayed login error if any and should be present a “password lost” recovery link.

2.2.5.3 Manager Users

Manager users' informations should be stored in a separate SQLite database file. These informations 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 contains the list of call center operators. Each operator

2.2.5.5 Variables

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

```
[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 to manage AsteriskCTI Services. From here, managers can add / modify or delete services.