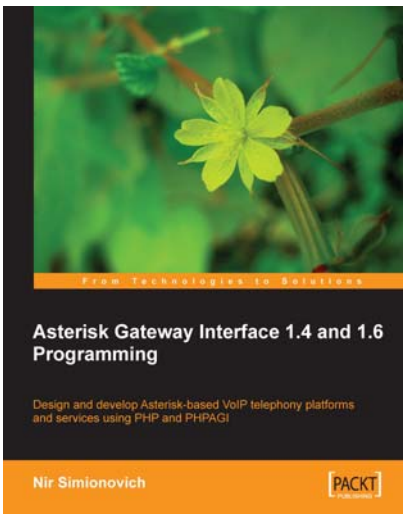
Asterisk Gateway Interface 1.4 and 1.6 Programming

Nir Simionovich



Chapter No. 4

"A Primer to AGI: Asterisk Gateway Interface"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.4 "A Primer to AGI: Asterisk Gateway Interface"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Nir Simionovich has been involved with the open source community in Israel since 1997. His involvement started back in 1997, when he was a student at Technion—Israel's Technology Institute—in Haifa. Nir quickly became involved in organizing open source venues and events, promoting the use of Linux and open source technologies in Israel.

In 1998, Nir started working for an IT consulting company (artNET experts Ltd.), where he began to introduce Linux-based solutions for enterprises and banks. By the year 2000, Nir had become a SAIR/GNU certified Linux trainer and administrator, slowly educating the future generations of Linux admins.

In 2001, Nir moved to the cellular content market, working for a mobile content delivery company (m-Wise Inc.—OTC.BB: MWIS.OB). During his commission at m-Wise, Nir successfully migrated a company—built purely on Windows 2000 and ColdFusion—to open source technologies, such as Mandrake Linux (today Mandriva), Apache Tomcat, and Kannel (an open source SMS/WAP gateway).

By 2006, Nir had co-founded Atelis (Atelis PLC—AIM: ATEL). Atelis is a Digium distributor and integrator. During the course of 2006, Nir developed an Asteriskbased international operator services platform for Bezeq International, which replaced a Nortel DMS-300 switch. This platform is currently in use by Bezeq International in Israel, serving over 4000 customer a day.

In mid 2007, Nir left Atelis to become a freelance Asterisk promoter and consultant. Nir is currently providing Asterisk consulting and development services for various companies, ranging from early-stage start-up companies, through VoIP service providers and VoIP equipment vendors. Nir is the founder of the Israeli Asterisk users group. In his spare time, he acts as the website maintainer of the group, and an Asterisk developer, dealing mainly with the localization aspects of Asterisk to Israel.

Coming to 2008, Nir's company (Greenfield Technologies Ltd) won the Digium Innovation award at AstriCon 2008, in the pioneer division—for its implementation of a phone-based prayer system, allowing people from around the world to pray at the western wall in Jerusalem.

Asterisk Gateway Interface 1.4 and 1.6 Programming

This is my second book, and I have to admit that I really enjoyed working on this book. While I enjoyed working on my previous book, (the *AsteriskNow* book from Packt Publishing), I couldn't help but feel that a portion of me has really slipped into the pages of this book.

This book is a developer's book, and it is written for developers by a developer. I see myself as an Asterisk application developer. After developing dozens of platforms over the course of the past six years, all based arround Asterisk, I can honestly say that I've seen mistakes that I made six years ago, still being made today by novice developers.

My role at Greenfield Technologies Ltd. (apart from being the CEO and Founder) is that of a development consultant, where I render various Asterisk consulting services to various companies in Israel and worldwide. Wherever I go, no matter what customer I cater, the mistakes and wrongful paradigms seem to persist. They persist due to a simple reason: there is no school for Asterisk developers. We have web developers, core developers, and database developers. But Asterisk developer is usually either a web developer or a core developer who is assigned a task, or in the worst case, a database developer entrusted with a task that he totally doesn't understand. The developers automatically do what they were taught to do: they superimpose their aggregated knowledge and experience on the Asterisk world, which usually ends up in disaster.

Asterisk is one of the most innovative pieces of open source software created in the past ten years (Asterisk just hit nine years old on December 05, 2008). While Asterisk provides one of the most extensive telephony toolkits available today, its utilization in a commercial application or platform construct isn't as straightforward as it would seem. This book de-mystifies some of the mystic characteristics associated with Asterisk, while at the same exposing some of the well-guarded secrets of professional Asterisk platform developers.

Asterisk requires a new skill set to be developed—one that web developers have no idea of and core developers completely disregard. My aim with this book is to enable you to learn the lessons and values that I've learned over a period of six years from a simple, shrink wrapped, to the point guide. I hope this book will remain on your table as a useful tool.

What This Book Covers

Chapter 1 introduces the various hardware elements required for installing your Asterisk PBX system, and guides you through the Asterisk installation procedure.

Chapter 2 introduces us to the dialplan—extension, context, and syntax. It then covers the main part—developing a basic **IVR** (**Interactive Voice Response**) application using Asterisk dialplan.

Chapter 3 takes us a bit deeper into IVR development, wherein we learn grabbing and processing input. The introduction of the Read application, and the conditional branching and execution, enable a new flexibility that was not available initially.

Chapter 4 is all about AGI—its working, its three types, and the different frameworks. Finally it covers the do's and don'ts that need to be followed for the AGI script to work and function properly.

Chapter 5 introduces you to your first AGI script, using the `Hello World` program. It also touches upon AGI debugging.

Chapter 6 covers a PHP based AGI class library—PHPAGI. The chapter starts with an explanation of the PHPAGI file structure, and then goes on to cover simple, and finally more complex, PHPAGI examples.

Chapter 7 introduces the basic elements of a FastAGI server, again using PHP and PHPAGI.

Chapter 8 helps understand the **Asterisk Manager Interface** (**AMI**)—an Asterisk proprietary **Computer Telephony Integration** (**CTI**) interface.

Chapter 9 takes you through the steps of developing a full click-2-call application, using all the concepts you've learned. Chapter 9 can be used as the basis for a large scale service, such as JaJah or RebTel.

Chapter 10 tries to deal with some of the more advanced topics of developing Asterisk applications—mainly scalability and performance issues. By the end of this chapter, the reader should be well-equipped with the information to build the next Verizon Killer application.

4

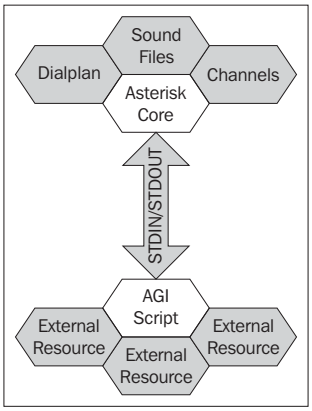
A Primer to AGI: Asterisk Gateway Interface

Explanation separates us from astonishment, which is the only gateway to the incomprehensible.–Eugene Ionesco

Eugene Ionesco, a Romanian/French playwright and dramatist is known mostly for his work on the "Theatre of the Absurd". Asterisk **AGI** (**Asterisk Gateway Interface**) enables an IVR developer to develop IVR structures that are sometimes, bordering on the absurd, as applications tend to become more and more complex by using AGI. However, there are some scenarios where common dialplan practices are no longer applicable, and the use of an external logic is a must. Enter AGI!

How does AGI work?

Let's examine the following diagram:



As the previous diagram illustrates, an AGI script communicates with Asterisk via two standard data streams—STDIN (Standard Input) and STDOUT (Standard Output). From the AGI script point-of-view, any input coming in from Asterisk would be considered STDIN, while output to Asterisk would be considered as STDOUT.

The idea of using STDIN/STDOUT data streams with applications isn't a new one, even if you're a junior level programmer. Think of it as regarding any input from Asterisk with a `read` directive and outputting to Asterisk with a `print` or `echo` directive. When thinking about it in such a simplistic manner, it is clear that AGI scripts can be written in any scripting or programming language, ranging from BASH scripting, through PERL/PHP scripting, to even writing C/C++ programs to perform the same task.

Let's now examine how an AGI script is invoked from within the Asterisk dialplan:

```
exten => _X.,1,AGI(some_script_name.agi,param1,param2,param3)
```

As you can see, the invocation is similar to the invocation of any other Asterisk dialplan application. However, there is one major difference between a regular dialplan application and an AGI script—the resources an AGI script consumes. While an internal application consumes a well-known set of resources from Asterisk, an AGI script simply hands over the control to an external process. Thus, the resources required to execute the external AGI script are now unknown, while at the same time, Asterisk consumes the resources for managing the execution of the AGI script. Now, imagine that your script is written in BASH. This means that every time you run an AGI script, a full BASH shell is invoked for the script. Ok, so BASH isn't much of a resource hog, but what about Java? This means that the choice of programming language for your AGI scripts is important. Choosing the wrong programming language can often lead to slow systems and in most cases, non-operational systems.

While one may argue that the underlying programming language has a direct impact on the performance of your AGI application, it is imperative to learn the impact of each. To be more exact, it's not the language itself, but more the technology of the programming language runtime that is important. The following table tries to distinguish between three programming languages' families and their applicability to AGI development.

Language Family	Member Languages	Details
Binary Compiled	C, C++, Pascal	The executable code generated can be highly optimized; thus, its general system footprint is fairly light; although these are the perfect choice for AGI development, the development process is long and tedious
Virtual Machine	Java, C#, Mono	Virtual machine executables incur a hefty toll, with the virtual machine itself usually consuming much memory; while languages like Java enable rapid development, their main use should be limited to FastAGI (described later in this book)
Interpreted	PERL, PHP, Python, Ruby	Interpreted languages have a slightly higher toll than binary compiled executables; however, their general footprint is much smaller than that of the Virtual Machine based languages; Interpreted languages, such as PHP, make up for about 80% of the AGI development in the world, and easily fit both AGI and FastAGI development

EAGI, DeadAGI and FastAGI

AGI has three cousins—EAGI, DeadAGI, and FastAGI. We shall now explain the use of each of these variants, and their proper usage.

EAGI—Enhanced Asterisk Gateway Interface

EAGI is a slightly more advanced version of AGI, allowing the AGI script to interact with the inbound audio stream via file descriptor 3. Essentially, EAGI can be used to create applications that can tap into an inbound audio stream, analyze it, and perform tasks in accordance with that stream of data.

The utilization of EAGI is not covered in this book.

DeadAGI—execution on hangup

Essentially speaking, AGI requires that an active channel be available for the AGI script to run. The main idea behind this is that an AGI script is supposed to interact with the user, or make the dialplan access various aspects outside the Asterisk environment.

A question that can be asked is: "In many scenarios we would like to execute commands upon the finalization of the call, or to be more exact, upon hangup or error. How can we run an AGI script upon hangup or error?" Well, the answer is: "By means of the utilization of the DeadAGI."

DeadAGI enables the execution of an AGI script on a hung-up channel, or a channel that has not been fully established yet (in general, a non-answering channel).

While the above behaviour is true for versions 1.0.X and 1.2.X of Asterisk, version 1.4.X generates a warning upon the execution of a DeadAGI on a channel that has just been established, even if not answered. Asterisk 1.6.X is supposed to include a facility that will enable it to decide what type of AGI operation to utilize, making the DeadAGI application obsolete.

Let's now examine how a DeadAGI script is invoked from within the Asterisk dialplan:

```
exten => h,1,DeadAGI(some_script_name.agi,param1,param2,param3)
```

The invocation is similar to that of a regular AGI script. However, DeadAGI scripts are supposed to be executed by the `h` extension only, or via the 'failed' extension mentioned in Chapter 2.

FastAGI—AGI execution via a TCP socket

Technically speaking, FastAGI is different in the following context: when Asterisk executes an AGI script via FastAGI, the resources required for the AGI script to run are consumed by a completely different process, and not Asterisk. In addition, the communications that were previously based on internal STDIN/STDOUT communications are now based on a TCP socket. This means that your AGI script, now actually an AGI server, can be operated and maintained on a completely different server, enabling you to separate the application logic from the Asterisk dialplan logic.

Bear in mind the following that if your FastAGI server has executed an internal Asterisk application (for example, playback), you will consume the resources of both the Asterisk application and the AGI execution client.

Let's now examine how a FastAGI script is invoked from within the Asterisk dialplan:

```
exten => _X.,1,AGI(agi://IP_NUMBER:PORT/some_script_name.agi)
```

Please note that passing arguments to the FastAGI servers is possible. However, it varies depending on the Asterisk version you are using.

Asterisk 1.2.X and 1.4.X

Passing arguments to a FastAGI server from either Asterisk 1.2.X or Asterisk 1.4.X is performed by using an HTTP GET type request:

```
exten => _X.,1,AGI(agi://192.168.2.1:1048/TestAGI?exten=${EXTEN})
```

In this case, the FastAGI server is responsible for handling the various arguments, parsing them, and handling each of them correctly.

Asterisk 1.6.X

Passing arguments to a FastAGI server from Asterisk 1.6.X is simpler and highly resembles the methodology used for a regular AGI script:

```
exten => _X.,1,AGI(agi://192.168.2.1:1048/TestAGI|${EXTEN}|{VAR2})
```

In this scenario, the arguments are available via the AGI variables named `agi_arg_1` and `agi_arg_2` respectively. The previous ones are also supported. However, if you are using Asterisk 1.6, try to use the new methodology, in order to be forward compatible.

FastAGI frameworks

As indicated above, FastAGI is a TCP socket based system, making it a client/server environment. As with any client/server environment that is based upon an open source technology, a multitude of frameworks exist in order to make our life easier in the development of FastAGI servers. The following is a short list of frameworks, available for various platforms that do just that:

Language	Framework	URL
.NET	NAsterisk	`http://www.codeplex.com/nasterisk`
ActiveX	AstOCX	`http://www.pcbest.net/astocx.htm`
Erlang	ErlAst	`http://tools.assembla.com/erlast`
Python	FATS StarPy	`http://fats.burus.org/` `http://www.vrplumber.com/programming/starpy/`
Java	Asterisk-Java	`http://www.voip-info.org/wiki/view/Asterisk-java`
Ruby	Adhearsion	`http://docs.adhearsion.com/display/adhearison/Home`

Others exist too; however, these are the most common ones for Asterisk FastAGI development.

AGI scripting frameworks

As with any other open source project, the number of frameworks built for the development of AGI scripts is amazing. Considering the fact that the AGI language consists of less than thirty different methods, the existence of over thirty different scripting frameworks is amazing.

The following list contains some of the more popular frameworks for AGI scripting:

Language	Framework	URL
PERL	Asterisk PERL Library	`http://asterisk.gnuinter.net/`
PHP	PHPAGI	`http://sourceforge.net/projects/phpagi/`
Python	py-Asterisk	`http://py-asterisk.berlios.de/py-asterisk.php`
C	libagiNow	`http://www.open-tk.de/libagiNow/`
.NET	MONO-TONE	`http://gundy.org/asterisk`

The AGI application

The following is the documentation of the AGI dialplan command, as it appears in the Asterisk documentation:

-= Info about application 'AGI' =-

```
[Synopsis]
Executes an AGI compliant application

[Description]
  [E|Dead]AGI(command|args): Executes an Asterisk Gateway Interface
compliant program on a channel. AGI allows Asterisk to launch external
programs written in any language to control a telephony channel, play
audio, read DTMF digits, etc. by communicating with the AGI protocol
on stdin and stdout.

  This channel will stop dialplan execution on hangup inside of this
application, except when using DeadAGI.  Otherwise, dialplan execution
will continue normally.
```

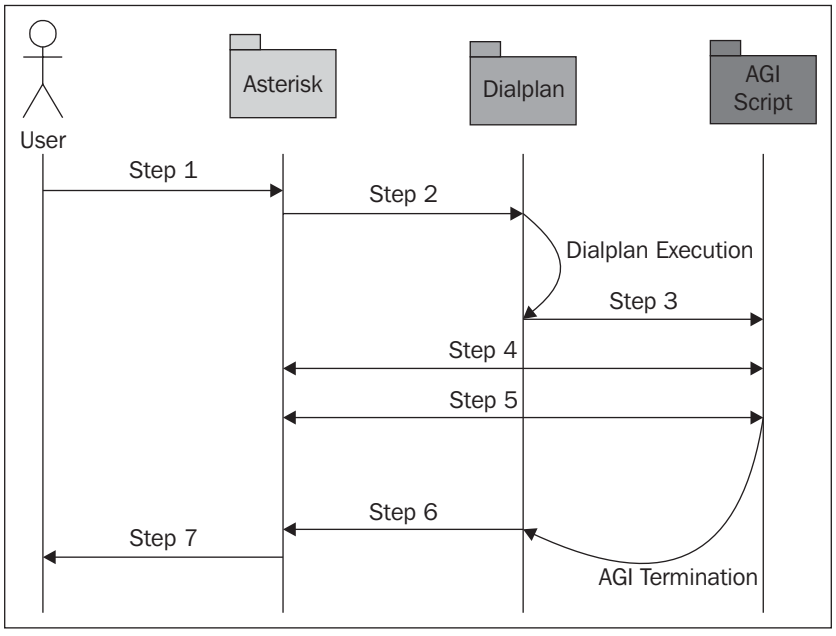
```
  A locally executed AGI script will receive SIGHUP on hangup from
the channel except when using DeadAGI. This can be disabled by setting
the AGISIGHUP channel variable to "no" before executing the AGI
application.

  Using 'EAGI' provides enhanced AGI, with incoming audio available
out of band on file descriptor 3

  Use the CLI command 'agi show' to list available agi commands
  This application sets the following channel variable upon
completion:
     AGISTATUS      The status of the attempt to the run the AGI
script text string, one of SUCCESS | FAILURE | HANGUP
```

Confusing? Well, for the first time you read this, it may actually be very confusing. Let's demystify AGI, shall we?

The AGI execution flow

Once an AGI script has been invoked, a preset information flow is performed between the AGI script and Asterisk. It is imperative to understand this information flow, as the structure of your AGI script depends on this flow.

The following diagram describes the steps that occur when an AGI script is executed from within the Asterisk dialplan:



If you are familiar with UML, the immediately preceding diagram may seem a little weird, as it doesn't follow the exact rules of the UML diagram. The diagram is meant for non-UML readers to be able to relate to the information.

As you can see, most of the interaction between Asterisk and our AGI script happens between the third and the fifth stages,. Let's examine what happens in these stages, using the following dialplan example:

```
exten => _X.,1,Answer
exten => _X.,n,Set(DID=${EXTEN})
exten => _X.,n,Set(CLID=${CALLERID(num)})
exten => _X.,n,AGI(SomeScript.php)
```

As our AGI script is being executed from the Asterisk dialplan, Asterisk will pass a preset number of variables, along with general AGI execution information to our AGI script, which requires initial processing, prior to the actual AGI script execution.

AGI Variable	Description
`agi_request`	Name of the agi script that is being called
`agi_channel`	Channel that the call is coming from
`agi_language`	Language that is configured on the server
`agi_type`	Call type; mainly the channel type
`agi_uniqueid`	A unique identifier for this session
`agi_callerid`	Caller ID number
`agi_calleridname`	Caller ID name, where available; not supported on all channel types
`agi_callingpres`	PRI Call ID presentation variable
`agi_callingani2`	Caller ANI2 (PRI channels), where applicable
`agi_callington`	Caller type of number (PRI channels)
`agi_callingtns`	Transit Network Selector (PRI channels)
`agi_dnid`	Dialed number identifier
`agi_rdnis`	Redirected Dial Number ID Service (RDNIS)
`agi_context`	Current context from which the AGI script was executed
`agi_extension`	Extension that was called
`agi_priority`	Current priority in the dialplan, that is, priority of the AGI script execution
`agi_enhanced`	0.0
`agi_accountcode`	Account code

As your script is being executed, all the information presented in the table, will be dumped into your script execution input, before you receive any other input from Asterisk.

At the time of writing this book, the set of variables presented in the table were found to be correct. However, it is highly probable that by the time you read this book, AGI execution will include some additional variables.

Most AGI scripts may regard the above as "noise", as most AGI scripts will obtain the information contained within these variables from an external source. When using a framework, you would notice that most frameworks simply disregard this information, and continue execution after simply skipping this portion of the execution.

Once our AGI script has finalized the information reading from Asterisk, our actual AGI script operations flow will begin, that is, our AGI script logic will begin its implementation. As we've already learned, AGI uses STDIN and STDOUT to communicate with Asterisk. In the next chapter, we shall start working with an actual AGI script. However, in the meantime knowledge of these streams is enough.

Once an AGI script has terminated its execution, it will return the control back to Asterisk for the continued execution of the Asterisk dialplan.

The AGI methods API

The following is a complete list of AGI methods, available to the developer via the AGI interface. This list was correct at the time of writing this book, although it may change slightly by the time you read this book. It is best to update yourself via the Asterisk documentation of the AGI command, or via the `agi show` command, available from your Asterisk CLI.

```
*CLI> agi show
                answer   Answer channel
        channel status   Returns status of the connected channel
          database del   Removes database key/value
      database deltree   Removes database keytree/value
          database get   Gets database value
          database put   Adds/updates database value
                  exec   Executes a given Application
              get data   Prompts for DTMF on a channel
     get full variable   Evaluates a channel expression
            get option   Stream file, prompt for DTMF, with timeout
          get variable   Gets a channel variable
```

```
              hangup   Hangup the current channel
                noop   Does nothing
        receive char   Receives one character from channels
                       supporting it
        receive text   Receives text from channels supporting it
         record file   Records to a given file
           say alpha   Says a given character string
          say digits   Says a given digit string
          say number   Says a given number
        say phonetic   Says a given character string with phonetics
            say date   Says a given date
            say time   Says a given time
        say datetime   Says a given time as specfied by the format
                       given
          send image   Sends images to channels supporting it
           send text   Sends text to channels supporting it
      set autohangup   Autohangup channel in some time
        set callerid   Sets callerid for the current channel
         set context   Sets channel context
       set extension   Changes channel extension
           set music   Enable/Disable Music on hold generator
        set priority   Set channel dialplan priority
        set variable   Sets a channel variable
         stream file   Sends audio file on channel
control stream file   Sends audio file on channel and allows the
                       listner to control the stream
            tdd mode   Toggles TDD mode (for the deaf)
             verbose   Logs a message to the asterisk verbose log
      wait for digit   Waits for a digit to be pressed
```

The ten rules of AGI development

Developers, who are given the task of developing an AGI script for the first time, tend to superimpose their traditional development techniques over the development of AGI scripts. By far, this is the most dangerous thing that can be done, as AGI scripting and traditional programming vary immensely. The following section will list the do's and don'ts that need to be followed, so that your AGI scripts work and function properly.

Rule #1: An AGI script should terminate as fast as possible

First-time AGI developers tend to develop their entire application within an AGI script. As you develop your entire application within an AGI script, you may gain the power of the scripting language, but will incur a cost of performance. Always make sure that the AGI scripts that you develop terminate their execution as fast as possible, returning to the dialplan as fast as possible. This concept dictates that each AGI script being run should behave quickly as an atomic unit—hence the name "Atomic AGI". We will learn the concepts of "Atomic AGI" development in Chapter 6.

Rule #2: Blocking applications have no place in AGI

As a direct continuation to rule #1, you should never execute a blocking application from within an AGI script. Initiating a blocking application from within an AGI script will make your Asterisk environment explode slowly. Why is that? Because for every blocking application that you run from within the AGI script, you will have both your AGI script and the blocking application running for the duration of the block. Imagine that you were to initiate the Dial application from within an AGI script, and the call created would last over thirty minutes. For those thirty minutes, your AGI script is still active. This isn't much of a problem when dealing with small-scale systems. But when trying to run 50 concurrent scripts, be prepared for failure.

Blocking applications include the following: Dial, MeetMe, MusicOnHold, Playback (when dealing with long playbacks), Monitor, ChanSpy, and other applications that have an unknown execution duration.

Rule #3: Asterisk channels are stateful—use them

An Asterisk channel, once operational, is like a big bucket of information. Channel variables can be used to carry information from your AGI script to the dialplan and back. The variables remain as part of the channel untill the channel is terminated, when memory is then freed.

Using this "bucket" enables you to carry variables and information obtained via an AGI script and then pass these to an application in the dialplan. For example, imagine that you are developing a pre-paid platform. A decision on the call timeout is taken via an AGI script. However, the actual dialling of the call is performed from the dialplan itself.

Rule #4: AGI scripts should manipulate data—no more

Most developers tend to think of AGI scripting as a functional unit, meaning that they enclose multiple functionalities into the AGI script. While AGI is fully capable of performing telephony functionality, it is best to leave this functionality to the dialplan.

Utilize your AGI script to set and reset channel variables and communicate with out-of-band information systems. The concept of allowing out-of-band information flow into Asterisk's operational flow of channel, enables new functionality and possibilities. Not all logic should be handled by your AGI script. Sometimes, it is better to use the AGI script as a data conduit, while letting an external information system handle the data manipulation.

Rule #5: VM based languages are bad for AGI scripting

Virtual machine based programming languages' are bad for AGI scripting. Putting aside the rules #1 and #2, imagine that your application is built using an AGI script using the Java programming language. If you are familiar with Java, you most probably know that for each program that you execute using Java, a full Java virtual machine is invoked.

While this practice may seem fairly normal for information systems, Asterisk and IVR development vary. Imagine that our system is required to handle a maximum number of 120 concurrent channels, which means that the maximum number of concurrent AGI scripts will be 120. According to this concept, our Java environment will be fully invoked for each of these 120 instances. In other words, too many resources are being used just for the VM.

If you really feel that you want to develop AGI scripts using Java, FastAGI is the way to go for you.

Rule #6: Binary-compiled AGI is not always the answer

While evaluating rules #1, #2 and #5, we can't but reach an almost immediate conclusion that we need to have our AGI script binary compiled. While in terms of efficiency and performance, a binary compiled AGI may provide better performance, the time required to develop it may be longer. In some cases, scripting languages such as PHP, PERL, and Python may provide near-similar performance at a lesser cost.

Also, when using binary compiled AGI scripts, you are always in charge of the memory allocation and cleanup. Even the most experienced developer can commit errors while dealing with memory allocation, so binary compiled AGI need not be the solution always.

If your system truly requires the performance edge of a binary compiled AGI, we encourage you to develop a prototype using a scripting language. Once you have your prototype working, start developing your binary version.

Rule #7: Balance your scripts with dialplan logic

While evaluating rules #1, #2 and #4, we must keep in mind that developing IVR systems with Asterisk resembles a high-wire balancing act. The fine art of balancing your dialplan with AGI scripts proves to be a powerful tool, especially when developing complex IVR systems.

Many developers tend to externalize functionality from the dialplan into AGI, while the same functionality can be achieved by writing dialplan macros or dialplan contexts. Using Asterisk's branching commands (`goto`, `gotoif`, `exec`, `execif`, `gosub` and `gosubif`) can easily remove redundant AGI code, passing the control from the AGI back to the dialplan.

When I developed my first system, I was amazed at the sheer magnitude of the impact that rule #7 can have on a system. A system that was developed entirely with AGI, and a system achieving the same functionality using a combination of AGI and dialplan, differed by a magnitude of eight (instead of being able to sustain fifteen calls, the system sustained 120 calls), in favour of the AGI and dialplan combination. Of course, your results may vary, according to your system.

Rule #8: Balance your scripts with web services

When evaluating rule #4, one may ask: "What is an out-of-band information system?" We shall explain now. Most Asterisk developers tend to develop their systems with the data information system—either embedded into their Asterisk server or communicating with an information system installed on the same server with the Asterisk server.

While, for small systems, this proves to be both efficient and economic, when developing a high-end system that requires scalability and redundancy, this methodology proves to be counter-productive. One of the methodologies (although many others exist) for interconnecting Asterisk with an out-of-band information system is web services. Communication to the web service is performed via AGI; the web-service protocol—you can use your favourite one.

The choice of protocol isn't that important, as almost any protocol type used for web services would do. Be it SOAP, WSDL, XML-RPC, WDDX or any other, take your pick, and the performance and scalability should be similar in any of these.

Rule #9: Syslog is your friend—use it

Every developer knows that using log files for debugging and monitoring purposes is a must. Be it for using a binary compiled AGI or a scripting language based AGI, make sure to utilize the logging facility. Trying to debug an AGI application from within the Asterisk console, though possible, can prove to be a tedious task. Sending log entries to a well-formatted log can save you much time and headache.

Scripting languages, such as PHP and PERL, do not offer a direct debugging facility, making the debugging of such AGI scripts even harder. Using log files as a debugging point for your AGI script will prove very useful when developing highly complex systems.

In order to make your syslog more readable, assign a self-created unique ID to each of your calls. When writing to your log, make sure that the unique ID appears in each log entry, so that you can trace a specific session flow through Asterisk. Remember, an Asterisk channel is stateful. The unique ID will remain as part of the channel untill it is removed from the system.

Rule #10: The Internet is for Asterisk

As bad as the following may sound, if you have a problem or an idea, remember that someone else had almost definitely come across it before you did. I don't want to discourage you, but actually, I want you to make use of the multitude of Asterisk resources available on the Internet.

The amount of information relating to Asterisk and platform development that has been accumulated by search engines is staggering. Over the course of the past two years, the amount of information available has multiplied two times (at least), making it the best source to find answers to your questions.

Asterisk user forums exist today in almost every country around the world; in some countries, there is more than one forum. These forums provide fast answers and professional guidance, allowing you to concentrate on your development, instead of concentrating on obtaining information.

When I first started developing AGI applications (almost six years ago), information was fairly scarce. While websites like `www.voip-info.org` and `www.asterisk.org` contained most of the information, much of the experience of the other developers was not documented. Today, most of these developers write personal blogs, updated almost daily, with answers and techniques for almost any Asterisk related issue. User forums have become more and more professional, thereby making these your best choice for information.
Other sources of information include the Asterisk IRC channel (#asterisk @ irc.freenode.net), the various Asterisk mailing lists available at the Asterisk community website, and of course the almighty Google.

A preface to what's coming ahead

Over the course of the forthcoming chapters, we shall begin our descent into AGI development. The choice of programming language for this book is PHP, due to its popularity and ease of development. If you feel uncomfortable with PHP, we are confident that you will be able to translate the code snippets into the programming/scripting language of your choice.

Chapter 5 will deal with your first AGI script; think of it as your "Hello-World" program from "Programming 101".

Chapter 6 will introduce a PHP based AGI class library, called PHPAGI. While PHPAGI is a fairly old library, and is compatible with all the versions of Asterisk, AGI hasn't changed dramatically from one the Asterisk version to the next. By using PHPAGI and the nine rules we just saw, we shall show that even an old, slightly outdated library, can do wonders.

Chapter 7 will introduce the basic elements of a FastAGI server, again using PHP and PHPAGI.

Chapter 8 will introduce the Asterisk Manager Interface (AMI), an Asterisk proprietary CTI (Computer Telephony Integration) interface.

Chapter 9 will take you through the steps of developing a full click-2-call application, using all the concepts you've learned. Chapter 9 can be used as a basis for a large scale service, such as JaJah or RebTel.

Summary

We have now completed our introduction to Asterisk's AGI technology. While AGI proves to be a fairly simplistic development API, the usage of AGI within your system requires you to be fully aware of your technological barriers. Be it Asterisk itself, your choice of programming/scripting language, your information systems, or the required user interaction, all these have to come into play while developing IVR systems with Asterisk and AGI.

Where to buy this book

You can buy Asterisk Gateway Interface 1.4 and 1.6 Programming from the Packt Publishing website: `http://www.packtpub.com/asterisk-gateway-interface-programming/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/asterisk-gateway-interface-programming/book