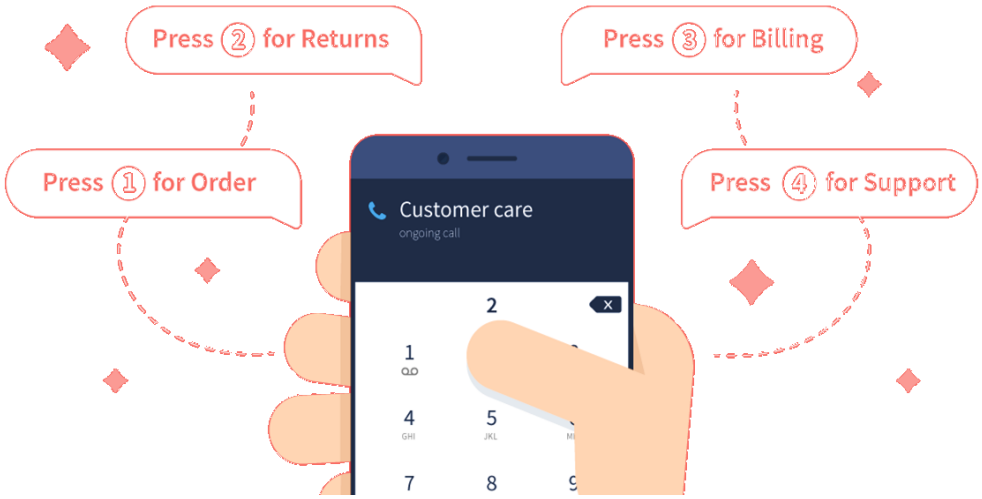
IVR

INTERACTIVE VOICE RESPONSE



Interactive Voice System for rating customer service

TELECOM TRACK 43 | ITI | JUNE, 22, 2023

Information Technology Institute

Telecom Application Development Track

Intake 43

IVR

Graduation Project 2023

Prepared by

Aisha Gamal

Manar Karam **Omar Ali**

Samir Hussein **Zyad El-Alfy**

Supervised by

Eng. Mohammed El-Sabbagh

Acknowledgments

To everyone who dedicated his life to help people. We would
just like to express our gratitude to our Parents and graduation project supervisor Eng. Mohammed El-Sabbagh

for his valuable advice, guidance and his enormous patience throughout the development of our project. Our appreciation to all
members of the.

Thank You

Abstract

Chapter 1: INTRODUCTION TO IVR

1.1. What is IVR?

Interactive voice response, or IVR, is an automated telephone system that combines pre-recorded messages or text-to-speech technology with a dual-tone multi-frequency (DTMF) interface to engage callers, allowing them to provide and access information without a live agent. If the IVR system cannot retrieve the information that the caller is looking for, the programmed menu options can help in routing callers to the appropriate representative for help. By integrating computer and telephony technologies, IVR software can improve call flow and reduce wait times, leading higher overall customer satisfaction.

IVR systems improve the customer experience by providing a self-service method for customers to access the information that they need without the assistance of customer support. It also reduces the call volume for contact centers, lowering wait times and operational costs for businesses.

1.2. Introduction to IVR

Today, IVR software is also evolving. The development of natural language processing technology expands the range of ways that callers can now interact with computers on the phone. Instead of using a touch tone system, more advanced IVR software enables callers to verbalize their needs on the phone. Then, through speech recognition, IVR system can understand and respond to their inquiries in real-time.

IVR systems improve the customer experience by providing a self-service method for customers to access the information that they need without the assistance of customer support. It also reduces the call volume for contact centers, lowering wait times and operational costs for businesses.

1.3. How interactive voice response works?

Interactive voice response phone system typically consists of the following components:

1. A TCP/IP network to provide internet and intranet connectivity.
2. Databases to supply IVR applications with relevant data

3. A web/application server where the IVR software applications will live. This server can host multiple applications, which are all written in Voice XML, for example, there could be applications for contact centers, outgoing sales calls and speech-to-text transcription.

From here, one of three types of IVR systems is typically constructed.

- **Touch-tone replacement:** This system prompts callers to use a touch-tone keypad selection to access information. For example, a pre-recorded message may say, "Press one for store hour information," and the caller would respond with "one."
- **Directed dialogue:** This type of IVR provides specific verbal prompts to callers depending on their inquiry. For example, the recording may ask, "Are you looking for store hours or location information?" The caller may respond with "store hours."
- **Natural language:** This advanced IVR system uses speech recognition to better understand user requests. For example, the system prompt may ask, "what information are you looking for today?" and the caller may reply with "I'm looking for store hour information" or other similar phrases.

1.4. Benefits of interactive voice response

IVR technology offers competitive advantages to businesses and advances their automation efforts. Some key benefits include:

- **Efficient call routing:** After obtaining relevant information from a given caller, IVR solutions route calls to the appropriate call-center agent, reducing wait times and increasing first contact resolutions.
- **Lower operational costs:** IVR systems are incredibly cost-effective. They not only reduce high call volumes for customer service representatives, but they can extend access to information during off-peak hours of the day, such as nights, weekends, and holidays.
- **Error Reduction:** When deployed effectively, IVR systems can reduce errors within the customer service process as it does not depend on a human customer service representative to take notes and route incoming calls appropriately.
- **Increased security:** Some IVR systems incorporate voice recognition technology to verify the identity of an individual, adding an extra layer of security. This can be helpful for highly sensitive personal information, like social security and phone numbers, checking and savings account information, and lab results from doctor's appointments.

1.5. Challenges of interactive voice response

While interactive voice response can offer benefits to businesses, the technology still has limitations that it needs to resolve and optimize for.

- **Over complex IVR menu options:** While IVR technology can streamline the call flow within call centers; it can also frustrate callers if the automated messaging system is too complex. Long pre-recorded messages may require callers to wait unnecessarily to select their intended option, resulting in lower customer satisfaction.
- **Long hold times:** Despite the advances in technology, long wait times remain a problem on many IVR systems. Callback functionality can alleviate frustration as callers can continue with other tasks in their day until a customer service representative is able to attend to their request.
- **Impersonal communication:** When customers are calling a support line, they may already be highly frustrated by a product or service issue. An automated messaging system may exacerbate frustration as a recording does not have the ability to empathize with their current problem.

Poorly deployed IVR systems can lead to high call abandonment rates and negative customer sentiment. Since low customer service satisfaction can harm a brand via negative reviews and public social media complaints, businesses should be thoughtful in their deployment of IVR solutions.

1.6. Applications of interactive voice response

IVR solutions have been utilized across a variety of industries, including banking, healthcare, education, and retail. Below we'll delve more deeply into these use cases:

Healthcare: IVR technology has a number of practical uses within healthcare, such as pre-treatment questionnaires, patient satisfaction surveys, lab and appointment scheduling, post-discharge follow-up, lab results and patient monitoring. This research (link resides outside of ibm.com) also highlights how it can increase overall patient satisfaction by reminding patients to adhere to their medication schedule.

Education: Research (link resides outside of ibm.com) has shown that educational institutions can implement IVR to assist parents in retrieving a status update on their child's performance and attendance in school. Parents can register with the system and then input a username and password to access key information on future calls.

Customer Service: Customer service call centers straddle across multiple industries. These centers are set up to handle a high volume of inbound calls using automated menus and pre-recorded to handle customer queries and complaints.

Finance: IVR can also be leveraged for a variety of tasks within banking and finance. They can provide account information, like account balances and loan application statuses, as well as enable changes to investment portfolios.

1.7. Key components of IVR systems

1. **Call Initiation**: A caller initiates a phone call to the IVR system. This can be through a designated phone number or a direct transfer from another system.

2. **Call Routing**: The IVR system routes the incoming call to the appropriate IVR application or menu based on predetermined criteria. This can include factors such as the time of day, the dialed number, or the caller's identification.

3. **Voice Prompts**: Once the call is connected, the IVR system plays pre-recorded voice prompts to guide the caller. These prompts provide instructions, options, and information about available services.

4. **Dual-Tone Multi-Frequency (DTMF) Input**: The caller can interact with the IVR system using their phone's keypad. This input method is known as Dual-Tone Multi-Frequency (DTMF) signaling. The caller can enter numerical data or select menu options by pressing the corresponding keys.

5. **Automatic Speech Recognition (ASR)**: IVR systems equipped with Automatic Speech Recognition technology can also accept voice input. Callers can speak their commands, answers, or other information, which the ASR system processes and interprets.

6. **Menu Navigation**: IVR systems present callers with a menu structure, typically in the form of a hierarchical menu. The caller selects options or branches by entering corresponding digits or speaking the desired choice.

7. **Database Integration**: IVR systems often integrate with backend databases or information systems. This allows them to retrieve caller-specific data, such as account balances, order statuses, or customer records.

8. **Transaction Processing**: IVR systems can handle various types of transactions initiated by callers. These can include tasks like making payments, scheduling appointments, or updating personal information. The IVR system guides callers through the necessary steps and processes the requested transactions.

9. **Call Transfer or Escalation**: If the IVR system cannot fulfill the caller's request or if the caller requests to speak with a live agent, the call may be transferred or escalated to a human operator or a different department within the organization.

10. **Call Completion**: Once the caller's interaction with the IVR system is complete, the system can provide a summary of the transaction or actions taken. The call is then either disconnected or transferred to the appropriate destination based on the caller's request.

The components and capabilities of IVR systems can vary depending on the specific implementation and requirements of the organization using them. Advanced IVR systems may incorporate features like natural language processing, advanced speech recognition, and integration with other communication channels such as catboats or web interfaces.

Chapter 2:
INTRODUCTION TO ASTERISK

2.1. A Brief History of the Asterisk Project

2.1.1. Linux Support Services

Way, way back in 1999 a young man named ***Mark Spencer*** was finishing his Computer Engineering degree at Auburn University when he hit on an interesting business concept. 1999 was the high point in the .com revolution (aka bubble), and thousands of businesses world-wide were discovering that they could save money by using the open source Linux operating system in place of proprietary operating systems.

The lure of a free operating system with open access to the source code was too much to pass up, unfortunately there was little in the way of commercial support available for Linux at that time, Mark decided to fill this gap by creating a company called "Linux Support Services". LSS offered a support hotline that IT professionals could (for a fee) call to get help with Linux.

The idea took off. Within a few months, Mark had a small office staffed with Linux experts. Within a few more months the growth of the business expanded demanded a "real" phone system that could distribute calls evenly across the support team, so Mark called up several local phone system vendors and asked for quotes, much to his surprise, the responses all came back well above $50,000 -- far more than Mark had budgeted for the project. Far more than LSS could afford.

2.1.2. Finding a Solution

Rather than give in and take out a small business loan, Mark made a pivotal decision. He decided to write his own phone system. Why not? A phone system is really just a computer running phone software, right? Fortunately for us, Mark had no idea how big a project he had taken on. If he had known what a massive undertaking it was to build a phone system from the ground up might have gritted his teeth, borrowed the money and spent the next

Decade doing Linux support, but he didn't know what he didn't know and so he started to code , Mark had done his engineering co-op at Adrian, a communications and networking device manufacturer in Huntsville, AL.

There he had cut his teeth on telecommunications system development, solving difficult problems generating a prodigious amount of complex code in a short time.

This experience proved invaluable as he began to frame out the system which grew into Asterisk.

In only ***a few months Mark crafted the original Asterisk core code***. As soon as he had a working prototype he published the source code on the Internet, making it available under the GPL license (the same license used for Linux).

Within a few months the idea of an "open source PBX" caught on. There had been a few other open source communications projects, but none had captured the imagination of the global population of communications geeks like Asterisk. As Mark labored on the core system, hundreds (now thousands)

Developers from all over the world began to submit new features and functions.

2.2. Asterisk in the Present

Asterisk is constantly evolving to meet the needs of the project's user-base. It's difficult to summarize the vast scope of everything that Asterisk can do as a communications toolkit; we'll list some resources that give you an idea of what is going on in the Asterisk project at present.

Asterisk Versions: Shows release timelines, support and EOL schedules

Roadmap section: Information from developer conferences and planning sessions

CHANGES: A document in Asterisk trunk, shows functionality changes between major versions

UPGRADE: A document in Asterisk trunk, shows breaking changes, deprecation of specific features and important info on upgrading.

Mailing lists: The dev. list is a great list to see what hot topics the developers are discussing in real-time.

2.2. What Is Asterisk?

People often tend to think of Asterisk as an "open source PBX" because that was the focus of the original development effort. But calling Asterisk a PBX is both selling it short (it is much more) and overstating it (it can be much less). It is true that Asterisk started out as a phone system for a small business but in the decade since it was originally released it has grown into a universal tool for building communications applications. Today Asterisk powers not only IP PBX systems but also VoIP gateways, call center systems, conference bridges, voicemail servers and all kinds of other applications that involve real-time communications.

Asterisk is not a PBX but is the engine that powers PBXs. Asterisk is not an IVR but is the engine that powers IVRs. Asterisk is not a call center ACD but is the engine that powers ACD/queuing systems.

Asterisk is to communications applications what the Apache web server is to web applications. Apache is a web server. Asterisk is a communication server. Apache handles all the low-level details of sending and receiving data using the HTTP protocol. Asterisk handles all the low level details of sending and receiving data using lots of different communication protocols. When you install Apache, you have a web server but it's up to you to create the web applications. When you install Asterisk, you have a communications server but it's up to you to create the communications applications.

Web applications are built out of HTML pages; CSS style sheets, server-side processing scripts, images, databases, web services, etc. Asterisk communications applications are built out of scripts, configuration files, audio recordings, databases, web services, etc., for a web application to work; you need the web server connected to the Internet. For a communications application to work, you need the communications server connected to communication services (VoIP or PSTN).

For people to be able to access your web site you need to register a domain name and set up DNS entries that point "www.yourdomain.com" to your server. For people to access your communications system you need phone numbers or VoIP URIs that send calls to your server.

In both cases the server is the plumbing that makes your application work, the server handles the low-level complexities and allows you, the application developer, to concentrate on the application logic and presentation.
You don't have to be an expert on HTTP to create powerful web applications, and you don't have to be an expert on ***SIP or Q.931*** to create powerful communications applications.

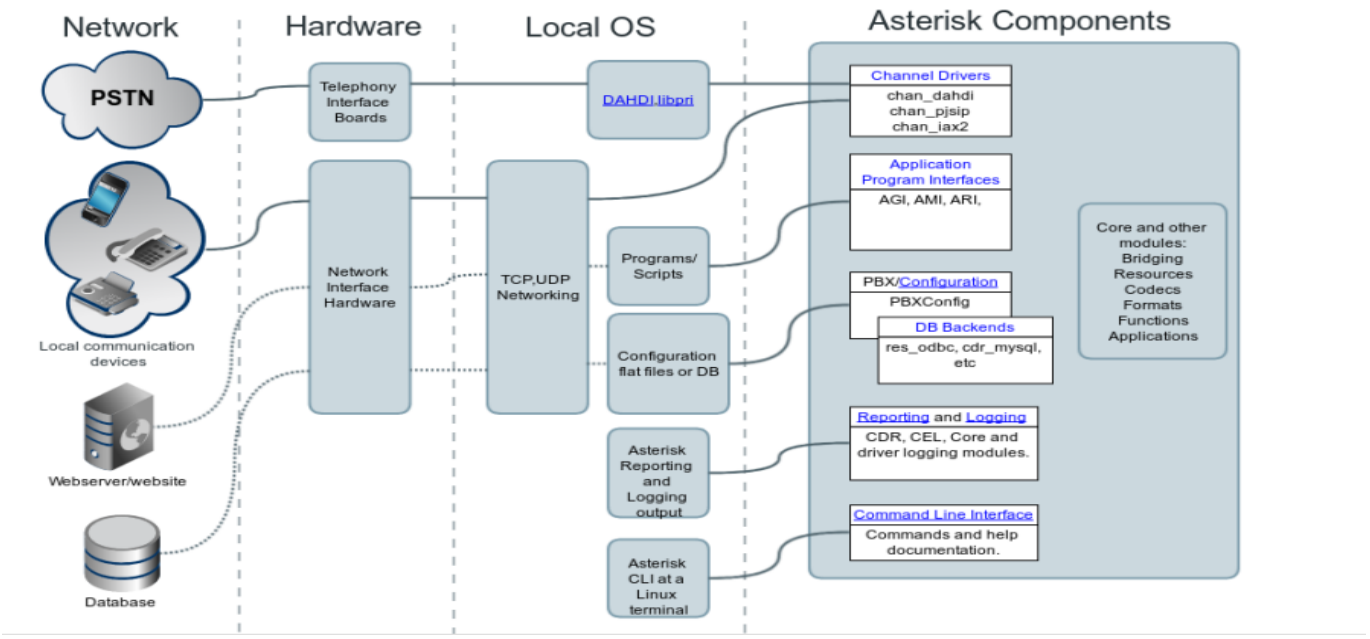
The following dial plan script answers the phone, waits for one second, plays back "hello world" then hangs up.

```
exten => 100,1,Answer()
exten => 100,n,Wait(1)
exten => 100,n,Playback(hello-world)
exten => 100,n,Hangup()
```

2.3. Asterisk Architecture

From an architectural standpoint, Asterisk is made up of many different modules. This modularity gives you an almost unlimited amount of flexibility in the design of an Asterisk-based system. As an Asterisk administrator, you have the choice on which modules to load and the configuration of each module.
Each module that you load provides different capabilities to the system. For example, one module might allow your Asterisk system to communicate with analog phone lines, while another might add call reporting capabilities.

Asterisk is a big program with many components, with complex relationships, to be able to use it, you don't have to know how everything relates in extreme detail, below is a simplified diagram intended to illustrate the relationships of some major components to each other and to entities outside Asterisk, it is useful to understand how a component may relate to things outside Asterisk as Asterisk is not typically operating without some connectivity or interaction with other network devices or files on the local system



(Figure1: An Asterisk System)

Asterisk has a **core** that can interact with many modules**. modules** called **channel drivers** provide channels that follow Asterisk **dial plan** to execute programmed behavior and facilitate communication between devices or programs outside Asterisk. Channels often use **bridging** infrastructure to interact with other channels, we'll describe some of these concepts briefly below.

2.4. Directory and File Structure Asterisk

The top level directories used by Asterisk can be configured in the asterisk.conf configuration file.

Here we'll describe what each directory is used for, and what sub-directories Asterisk will place in each by default. Below each heading you can also see the correlating configuration line in asterisk.conf.

2.4.1. Asterisk Configuration Files

```
astetcdir => /etc/asterisk
```

This location is used to store and read Asterisk configuration files. That is generally files with a .conf extension, but other configuration types as well, for example .lua and .ael.

2.4.2. Asterisk Modules

```
astmoddir => /usr/lib/asterisk/modules
```

Loadable modules in Shared Object format (.so) installed by Asterisk or the user should go here.

2.4.3. Various Libraries

```
astvarlibdir => /var/lib/asterisk
```

Additional library elements and files containing data used in runtime are put here.

2.4.4. Database Directory

```
astdbdir => /var/lib/asterisk
```

This location is used to store the data file for Asterisk's internal database. In Asterisk versions using the SQLite3 database, the file will be named astdb.sqlite3.

2.4.5. Encryption Keys

```
astkeydir => /var/lib/asterisk
```

When configuring key-based encryption, Asterisk will look in the keys subdirectory of this location for the necessary keys.

2.4.6. System Data Directory

```
astdatadir => /var/lib/asterisk
```

By default, Asterisk sounds are stored and read from the sounds subdirectory at this location.

2.4.7. AGI (Asterisk Gateway Interface) Directory

```
astagidir => /var/lib/asterisk/agi-bin
```

When using various AGI applications, Asterisk looks here for the AGI scripts by default.

2.4.8. Spool Directories

```
astspooldir => /var/spool/asterisk
```

This directory is used for storing spool files from various core and module-provided components of Asterisk.

2.4.9. Logging Output

```
astlogdir => /var/log/asterisk
```

When Asterisk is configured to provide log file output, it will be stored in this directory.

2.4.10. System Binary Directory

```
astsbindir => /usr/sbin
```

By default, Asterisk looks in this directory for any system binaries that it uses, if you move the Asterisk binary itself or any others that it uses, you'll need to change this location.

2.5. Asterisk Command Line Interface (CLI)

The Command Line Interface, or console for Asterisk, serves a variety of purposes for an Asterisk administrator.

- Obtaining information on Asterisk system components
- Affecting system configuration
- Seeing log output, errors and warnings in real-time

- Generating calls for testing
- Viewing embedded help documentation such as for APIs, applications, functions and module configuration

The sub-sections under this page will discuss how to access and use the CLI. That is, CLI syntax, command line help, and other CLI related topics.

For information on configuration of the CLI, see the Asterisk CLI Configuration section of the wiki.

2.5.1. Connecting to the Asterisk CLI

There are two ways to connect to an Asterisk console, either a **foreground console** when starting the Asterisk process or by connecting to a **remote console** after Asterisk is already running.

2.5.2. Connecting to a foreground console

This is as simple as passing the -c flag when starting Asterisk.

```
-c    Provide  a control console on the calling terminal. The console is similar to the remote console provided
      by -r. Specifying this option implies -f and will cause asterisk to no longer fork  or  detach  from  the
      controlling terminal. Equivalent to console = yes in asterisk.conf.
```

When working on a foreground console, you won't be able to exit, instead you'll have to stop Asterisk to return to the Linux shell. Most people will use a remote console when performing administration tasks.

After Asterisk has finished loading, you'll see the default CLI prompt. The text "server" will be replaced with your system's hostname.

```
server*CLI>
```

2.5.3. Connecting to a remote console

Connecting to a remote console using the -r or -R flags.

```
-r    Instead of running a new Asterisk process, attempt to connect to a running Asterisk process and provide a
      console interface for controlling it.
-R    Much like -r. Instead of running a new Asterisk process, attempt to connect to a running Asterisk process
      and provide a console interface for controlling it. Additionally, if connection to the  Asterisk  process
      is lost, attempt to reconnect for as long as 30 seconds.
```

To **exit a remote console**, simply type 'quit' or 'exit'. Please note that you can differentiate between a remote console and a foreground Asterisk console, as 'quit' or 'exit' will not

function on the main console, which prevents an accidental shutdown of the daemon. If you would like to shut down the Asterisk daemon from a remote console, there are various commands available.

2.6. Installation and Setup of Asterisk (Client and Server)

To install and set up Asterisk for your communication needs, follow these steps:

1. **Choose an Operating System**: Select a compatible operating system such as Centos, Ubuntu, for your Asterisk installation.

2. **Install Dependencies**: Install the necessary dependencies and libraries required by Asterisk, including development tools, audio and video codecs, and network components.

3. **Download and Compile Asterisk**: Obtain the Asterisk source code from the official website or community repository. Compile and install Asterisk using the provided installation scripts, ensuring you configure it with the desired options and modules.

4. **Configure Asterisk**: Customize the Asterisk configuration files to define your telephony settings, such as SIP/IAX trunks, extensions, dial plans, and other features. These configuration files include `sip.conf`, `extensions.conf`, and `iax.conf`.

5. **Start Asterisk**: Initiate the Asterisk service using the appropriate command for your operating system. Confirm that Asterisk is running smoothly and review the log files for any error messages or warnings.

6. Connect Clients: Configure and connect your client devices, such as softphones or IP phones, to the Asterisk server. Provide the necessary credentials (username, password, server IP) for authentication and registration with the Asterisk server.

7. **Test and Troubleshoot**: Perform test calls to verify call quality and test the functionality of your telephony applications. Should any issues arise, consult the Asterisk documentation, forums, or community resources for troubleshooting guidance.

Please note that the installation and setup process can vary depending on the specific operating system, Asterisk version, and any additional requirements or customizations. Therefore, it is advisable to refer to the official Asterisk documentation and community resources for detailed instructions tailored to your specific setup.

By following these steps, you'll be well on your way to harnessing the power of Asterisk and creating robust communication applications for your needs.

Chapter 3:
INTRODUCTION TO AGI

3.1. Introduction to AGI and its Role in Telephony Systems:

The Asterisk Gateway Interface (AGI) plays a vital role in telephony systems by serving as a bridge between the Asterisk PBX (Private Branch Exchange) platform and external scripts or applications. AGI enables seamless communication and interaction between the PBX and custom scripts, empowering businesses to create tailored telephony solutions. By acting as an interface, AGI opens up possibilities for extending the functionality of the PBX and implementing advanced call handling features, interactive voice response (IVR) systems, and more. With AGI, businesses can harness the power of customization and enhance their telephony systems to better suit their specific needs.

3.2. Emergence of AGI

AGI, which stands for Asterisk Gateway Interface, has a history closely tied to the development and evolution of the Asterisk open-source PBX (Private Branch Exchange) platform. Asterisk, created by Mark Spencer in 1999, revolutionized the telecommunications industry by offering a flexible and powerful software-based telephony solution. AGI was introduced as a means to extend the functionality of the Asterisk PBX and enable seamless integration with external scripts and applications.

AGI was designed to provide a standardized interface for communication between the Asterisk PBX and external programs. It allows developers to leverage their preferred programming languages to interact with the PBX, enabling the creation of custom telephony solutions. AGI scripts or applications can be written in various languages such as Perl, PHP, Python, Java and more, making it accessible to a wide range of developers.

3.3. Usage of AGI

Initially, AGI was primarily used to build Interactive Voice Response (IVR) systems, enabling businesses to create custom call handling and routing logic. IVR systems built with AGI could handle incoming calls, play voice prompts, collect caller input, and route calls based on predefined rules. This allowed businesses to create dynamic and interactive voice-based menus, automate processes, and enhance caller experiences.

Over time, the usage of AGI expanded beyond IVR systems. AGI became instrumental in integrating telephony systems with other business applications. By utilizing AGI, developers could bridge the gap between telephony and CRM systems, helpdesk

applications, databases, and more. This integration facilitated streamlined workflows, improved call logging, and enhanced efficiency in managing customer interactions.

3.4. Customization and Extensibility:

One of the key advantages of AGI is its ability to accommodate developers' preferred programming languages, allowing them to create custom telephony solutions. This flexibility and extensibility make AGI a powerful tool for enhancing the functionality of the Asterisk PBX. Developers can leverage their expertise in languages such as Python, Perl, or PHP to implement sophisticated call handling logic, integrate external databases, and enable seamless interaction with other applications. AGI provides a sandbox environment for developers to unleash their creativity and develop unique telephony features that align with their business requirements.

3.5. Real-time Interaction and Dynamic Call Control:

AGI facilitates real-time communication between the Asterisk PBX and external scripts, enabling dynamic call control. This means that businesses can implement intelligent call routing algorithms, call recording functionalities, and advanced call analytics. AGI empowers businesses to monitor ongoing calls, retrieve call details, and execute actions based on specific triggers or events. With AGI, telephony systems can dynamically adapt to changing conditions, optimize call handling processes, and provide personalized experiences for callers.

3.6. Integration of Speech Recognition and Synthesis:

AGI enables the integration of speech recognition and synthesis technologies, enhancing the capabilities of telephony systems. By leveraging AGI, businesses can develop voice-enabled applications and navigate through complex IVR systems using natural language. Speech recognition allows callers to interact with the system by speaking commands or providing input, while speech synthesis empowers the system to respond with pre-recorded or synthesized voice prompts. AGI makes it possible to create interactive and intuitive telephony experiences that rely on speech as a primary interface.

3.7. Administration and Call Monitoring:

AGI empowers administrators to create custom call handling logic, routing rules, and monitoring capabilities. With AGI, administrators have fine-grained control over ongoing calls, allowing them to monitor call status, retrieve call details, and manipulate call flow based on specific criteria. AGI also enables administrators to execute actions in real-time, such as transferring calls to different queues or IVR menus, initiating call recording, or activating call analytics modules. This level of control ensures efficient call management and empowers administrators to adapt telephony systems to changing business needs.

3.8. Enhancing Customer Experiences:

Through AGI, businesses can significantly enhance customer experiences by offering personalized call handling and intelligent call routing. AGI enables businesses to implement call prioritization based on customer profiles or previous interactions, ensuring high-value customers receive premium treatment. Additionally, intelligent call routing algorithms can consider factors such as agent availability, skills, or customer preferences to deliver efficient and satisfactory call outcomes. AGI empowers businesses to reduce call waiting times, improve first-call resolution rates, and ultimately enhance customer satisfaction.

3.9. Future Prospects and Innovations:

Looking ahead, AGI holds promising prospects for further advancements in telephony systems. Continued innovation in speech recognition and synthesis technologies will enable more sophisticated voice-enabled applications and natural language processing capabilities. Integration with emerging technologies like AI and machine learning can enhance call analytics, sentiment analysis, and automated call handling. AGI may also play a role in the evolution of virtual assistants, Chabot's, and smart communication systems. As technology progresses, AGI is poised to contribute to the development of more intelligent and intuitive telephony solutions.

3.10. Sample AGI script in java:

The AGI (Asterisk Gateway Interface) facility allows you to launch scripts, from the Asterisk dial plan. Traditionally communication between the scripts and Asterisk was via standard input and standard output and scripts had to run on the same machine as Asterisk.

```
import org.asteriskjava.fastagi.AgiChannel;
import org.asteriskjava.fastagi.AgiException;
import org.asteriskjava.fastagi.AgiRequest;
import org.asteriskjava.fastagi.BaseAgiScript;

public class HelloAgiScript extends BaseAgiScript
{
   public void service(AgiRequest request, AgiChannel channel)
         throws AgiException
   {
      // Answer the channel...
      answer();

      // ...say hello...
      streamFile("welcome");
```

```
        // ...and hangup.
        hangup();     }
}
```

This is a Java code that uses the Asterisk-Java package. The package consists of a set of Java classes that allow you to easily build Java applications that interact with an Asterisk PBX Server.

Asterisk-Java supports both interfaces that Asterisk provides for this scenario:

The FastAGI protocol and the Manager API.

The code imports four classes from the package:
org.asteriskjava.fastagi.AgiChannel
org.asteriskjava.fastagi.AgiException
org.asteriskjava.fastagi.AgiRequest
org.asteriskjava.fastagi.BaseAgiScript

The code extends the BaseAgiScript class and overrides its service method. The service method takes two parameters:

An instance of the AgiRequest class.

An instance of the AgiChannel class.

The method first answers the channel using the answer() method.

Then it says hello by streaming a file named “welcome” using the streamFile() method.

Finally, it hangs up using the hangup() method

Chapter 4:
INTRODUCTION TO VXML

4.1. Introduction to Voice Extensible Markup Language (VXML)

Voice Extensible Markup Language (VXML) is a markup language used for creating interactive voice response (IVR) systems. It enables developers to build applications that interact with users through voice commands and prompts. VXML combines elements of HTML and XML to define the structure and behavior of voice applications.

(VXML) is a digital document standard for specifying interactive media and voice dialogs between humans and computers. The Voice XML document format is based on Extensible Markup Language (XML), it is a standard developed by the World Wide Web Consortium (W3C).

4.2. Emergence of Voice Response Systems

In the 1990s, interactive voice response (IVR) systems started gaining popularity as an automated solution for telephone-based interactions. These systems allowed users to interact with computers using voice commands and touch-tone input. However, the development of IVR applications was complex and required specialized programming skills, to simplify the development of voice applications and promote interoperability, a group of industry leaders including AT&T, IBM, Lucent Technologies, and Motorola formed the Voice XML Forum in March 1999. The Voice XML Forum aimed to create a standard markup language that could be used for designing voice applications across different platforms and vendors.

In March 2000, the Voice XML Forum released Voice XML 1.0 as an open, vendor-neutral standard for creating voice applications, and More versions have been released in order to enhance and expand functionality.

4.3. Difference between VXML and XML

Voice XML (VXML) and XML (Extensible Markup Language) are both markup languages, but they serve different purposes and are used in different contexts:

4.3.1. XML (Extensible Markup Language):

- XML is a generic markup language used for storing and transmitting data in a structured format. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- XML is used to represent hierarchical data with tags that describe the structure and meaning of the data. These tags are user-defined, allowing for the creation of custom data formats suitable for various applications.
- XML is widely used for data exchange, configuration files, web services, and other applications where structured data needs to be stored or transmitted.
- XML documents have a root element that encapsulates all other elements, forming a tree-like structure.

Example of XML:

```
<book>
  <title>Introduction to XML</title>
  <author>John Doe</author>
  <year>2023</year>
</book>
```

4.3.2. Voice XML (VXML):

- Voice XML is a specialized markup language designed for creating interactive voice response (IVR) systems. It is used to create voice applications that can interact with users over the phone or other voice-based communication channels.
- Voice XML allows developers to define voice prompts, menu options, audio playback, voice recognition, and other voice-related interactions.
- Voice XML applications are typically used in telephony systems, call centers, and automated phone services, where users interact with the system by speaking and listening to prompts.
- Voice XML documents are specifically designed to handle voice-based interactions and can include audio elements, grammar tags for voice recognition, and other telephony-specific features.

Example of Voice XML:

```
<vxml version="2.1">
  <form>
    <field name="userInput">
      <prompt>Please say your name after the beep.</prompt>
      <grammar src="names.grxml" type="application/srgs+xml"/>
    </field>
  </form>
</vxml>
```

In summary, XML is a general-purpose markup language used for storing and exchanging structured data, while Voice XML is a specific markup language used to create voice applications for IVR systems and telephony-based interactions. VoiceXML extends the capabilities of XML to handle voice interactions and telephony-specific functionalities.

4.4. Usage of VXML

Voice XML applications are commonly used in many industries and segments of commerce. These applications include order inquiry, package tracking, driving directions, emergency notification, wake-up, flight tracking, voice access to email, customer relationship management, prescription refilling, audio news magazines, voice dialling, real-estate information and national directory assistance applications.

4.5. Key features of VXML

Structure of VXML:

VXML follows a document-centric structure, similar to HTML. It consists of elements and attributes that define the behavior of the voice application. These elements include <form>, <field>, <prompt>, <grammar>, and more. The structure of VXML allows developers to create dialogues and handle user input effectively.

Voice Prompts and Audio Playback:

VXML provides <prompt> elements to deliver pre-recorded voice prompts to the user. These prompts can be played using text-to-speech synthesis or by playing pre-recorded audio files. Developers can control the timing, speed, and voice characteristics of the prompts, providing a natural and interactive user experience.

User Input Handling:

With VXML, developers can define <field> elements to handle user input. These fields can specify grammars or define specific recognition criteria for voice input. VXML supports both speech recognition and touch-tone input, allowing users to interact with the application using their voice or a phone's keypad.

Call Control and Dialog Management:

VXML supports call control capabilities, allowing developers to perform actions such as transferring calls, conferencing, and recording. It also provides features for managing dialogues, including branching logic, conditional statements, and error handling. This flexibility enables the creation of dynamic and personalised voice applications.

Platform Independence:

VXML is designed to be platform-independent, meaning that voice applications written in VXML can run on various voice platforms and IVR systems. This portability allows developers to write once and deploy across different environments, reducing development time and effort.

Industry Standard:

VXML is an industry-standard markup language for developing voice applications. It is supported by major IVR platforms and has a large developer community. This standardization ensures compatibility, interoperability, and ongoing support for VXML-based applications.

4.6. VXML Code For IVR Project

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1">
 <form>
  <block>
   <prompt>
    <audio src="rate_prompt.wav" />
   </prompt>
   <filled>
    <if cond="rating == 1">
     <assign name="userRating" expr="1" />
```

```
        <prompt>
          <audio src="confusion_prompt.wav" />
        </prompt>
      <elseif cond="rating == 2"/>
        <assign name="userRating" expr="2" />
        <prompt>
          <audio src="feedback_prompt.wav" />
        </prompt>
      <elseif cond="rating == 3"/>
        <assign name="userRating" expr="3" />
        <prompt>
          <audio src="feedback_prompt.wav" />
        </prompt>
      <elseif cond="rating == 4"/>
        <assign name="userRating" expr="4" />
        <prompt>
          <audio src="feedback_prompt.wav" />
        </prompt>
      <elseif cond="rating == 5"/>
        <assign name="userRating" expr="5" />
        <prompt>
         <audio src="satisfied_prompt.wav" />
        </prompt>
      <else/>
        <prompt>
          <audio src="invalid_rating_prompt.wav" />
        </prompt>
```

```
        <assign name="attempts" expr="attempts + 1"/>
        <if cond="attempts >= 3">
          <prompt>
            <audio src="goodbye_prompt.wav" />
          </prompt>
          <disconnect/>
        </if>
      </if>
    </filled>
    <catch event="connection.disconnect.hangup">
      <exit />
    </catch>
    <catch event="connection.disconnect.error">
      <exit />
    </catch>
    <collect>
      <prompt>
        <audio src="rate_digits_prompt.wav" />
      </prompt>
      <grammar src="digits.grxml" />
      <noinput>
        <prompt>
          <audio src="noinput_prompt.wav" />
        </prompt>
        <reprompt />
      </noinput>
      <nomatch>
```

```
        <prompt>
          <audio src="nomatch_prompt.wav" />
        </prompt>
        <reprompt />
      </nomatch>
    </collect>
  </block>
 </form>
</vxml>
```

Let's go through each line of the provided code and explain its purpose:

- **<?xml version="1.0" encoding="UTF-8"?>**: This line declares the XML version and encoding used in the document.
- **<vxml version="2.1">**: This line specifies the version of VoiceXML used in the document.
- **<form>**: This tag defines a form that contains the interactive voice response (IVR) application.
- **<block>**: This tag represents a block of code or a sequence of actions within the form.
- **<prompt>**: This tag is used to play a voice prompt to the user.
- **<audio src="rate_prompt.wav" />**: This line specifies an audio file called "rate_prompt.wav" to be played as the initial prompt.
- **<filled>**: This tag indicates that this block should be executed when the form is filled with user input.
- **<if cond="rating == 1">**: This line checks if the variable "rating" is equal to 1. If the condition is true, it proceeds to the next line.
- **<assign name="userRating" expr="1" />**: This line assigns the value 1 to the variable "userRating".
- **<prompt>**: This tag plays an audio prompt called "confusion_prompt.wav" to the user.

- **<elseif cond="rating == 2"/>**: This line checks if the variable "rating" is equal to 2. If the condition is true, it proceeds to the next line.
- **<assign name="userRating" expr="2" />**: This line assigns the value 2 to the variable "userRating".
- **<prompt>**: This tag plays an audio prompt called "feedback_prompt.wav" to the user.
- **<elseif cond="rating == 3"/>**: This line checks if the variable "rating" is equal to 3. If the condition is true, it proceeds to the next line.
- **<assign name="userRating" expr="3" />**: This line assigns the value 3 to the variable "userRating".
- **<prompt>**: This tag plays an audio prompt called "feedback_prompt.wav" to the user.
- **<elseif cond="rating == 4"/>**: This line checks if the variable "rating" is equal to 4. If the condition is true, it proceeds to the next line.
- **<assign name="userRating" expr="4" />**: This line assigns the value 4 to the variable "userRating".
- **<prompt>**: This tag plays an audio prompt called "feedback_prompt.wav" to the user.
- **<elseif cond="rating == 5"/>**: This line checks if the variable "rating" is equal to 5. If the condition is true, it proceeds to the next line.
- **<assign name="userRating" expr="5" />**: This line assigns the value 5 to the variable "userRating".
- **<prompt>**: This tag plays an audio prompt called "satisfied_prompt.wav" to the user.
- **<else/>**: This line is executed if none of the previous conditions are true. It proceeds to the next line.
- **<prompt>**: This tag plays an audio prompt called "invalid_rating_prompt.wav" to the user.
- **<assign name="attempts" expr="attempts + 1"/>**: This line increments the value of the variable "attempts" by 1.

- **<if cond="attempts >= 3">**: This line checks if the value of the variable "attempts" is greater than or equal to 3. If the condition is true, it proceeds to the next line.
- **<prompt>**: This tag plays an audio prompt called "goodbye_prompt.wav" to the user.
- **<disconnect/>**: This tag disconnects the call.
- **</if>**: This line marks the end of the conditional statement.
- **</if>**: This line marks the end of the outer conditional statement.
- **<catch event="connection.disconnect.hangup">**: This line specifies a catch block to handle the event when the caller hangs up.
- **<exit />**: This tag exits the VXML document.
- **<catch event="connection.disconnect.error">**: This line specifies a catch block to handle the event when a connection error occurs.
- **<exit />**: This tag exits the VXML document.
- **<collect>**: This tag defines a section for collecting user input.
- **<prompt>**: This tag plays an audio prompt called "rate_digits_prompt.wav" to the user.
- **<grammar src="digits.grxml" />**: This line specifies a grammar file called "digits.grxml" to be used for speech recognition.
- **<noinput>**: This tag handles the event when no input is received from the user.
- **<prompt>**: This tag plays an audio prompt called "noinput_prompt.wav" to the user.
- **<reprompt />**: This tag requests the user to provide input again.
- **<nomatch>**: This tag handles the event when the user's input does not match the expected grammar.
- **<prompt>**: This tag plays an audio prompt called "nomatch_prompt.wav" to the user.
- **<reprompt />**: This tag requests the user to provide input again.
- **</collect>**: This line marks the end of the collection section.

- **</block>**: This line marks the end of the block.
- **</form>**: This line marks the end of the form.
- **</vxml>**: This line marks the end of the VXML document.

Each line of code in the VXML document serves a specific purpose in creating an IVR system that collects user ratings, prompts audio prompts accordingly.

Digitals.grxml

By using a grammar file like digits.grxml, voice applications can provide a more natural and interactive user experience. Users can speak their input instead of typing or selecting options, which can be especially useful in hands-free or voice-driven scenarios.

```
<grammar xmlns="http://www.w3.org/2001/06/grammar" version="1.0" root="digit">
  <rule id="digit">
    <one-of>
      <item>1</item>
      <item>2</item>
      <item>3</item>
      <item>4</item>
      <item>5</item>
    </one-of>
  </rule>
</grammar>
```

Let's break down the elements and their meanings:

- **xmlns="http://www.w3.org/2001/06/grammar"**: This attribute specifies the namespace of the grammar element. In this case, it refers to the W3C (World Wide Web Consortium) namespace for gramrs.
- **version="1.0"**: This attribute specifies the version of the grammar specification being used. In this case, it indicates version 1.0.
- **root="digit"**: This attribute specifies the root rule of the grammar, which serves as the starting point for the recognition process. In this example, the root rule is named "digit."

- **<rule id="digit">**: This element defines a rule within the grammar. The id attribute assigns a unique identifier to the rule, which can be referenced elsewhere in the grammar.
- **<one-of>**: This element represents a set of choices. It is used to define multiple alternative items within a rule.
- **<item>**: This element represents an individual item or option within a set of choices. In this example, the items are the numbers 1, 2, 3, 4, and 5.

So, in summary, the provided grammar defines a rule called "digit" that recognizes the spoken words "1," "2," "3," "4," or "5."