Creating an Agent for Alexa

Miguel Rufino Francisco
miguel.francisco@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2019

Abstract

FAtiMA-Toolkit is a powerful collection of assets and tools that aim to aid in the creation of characters and AI capable of emotional and social understanding. The toolkit allows for the creation of characters that act in a believable, human-like way. There are limitations on what developers can do with FAtiMA-Toolkit. One of such is the inability to create dialogues in speech form since FAtiMA-Toolkit does not have any speech recognition technology. With this in mind, the goal of this project is to create an agent using FAtiMA-Toolkit, that uses Amazon’s Alexa has a speech recognizer. Integrating FAtiMA-Toolkit with Amazon’s Alexa will enable future developers to create dialogues in the FAtiMA-Toolkit with the available option of speech recognition.

Keywords: FAtiMA-Toolkit, Amazon’s Alexa, speech recognition, believable characters.

1. Introduction

In the current days, conversational agents of all sorts are present in the market. Artificial Intelligence is said to be the future of technology. The need to create intelligent agents increases as technology evolves. There are several fields that have an increasing need to have these believable, human-like agents. Video-game characters and personal assistants are some of the several applications of these agents.

With the increasing need to create dialogues and agents with believable interactions, various toolkits were created to help developers generate their ideal agents and characters with ease. One of such is FAtiMA-Toolkit[8][5][6], this toolkit is designed to help create agents capable of social and emotional intelligence. It has a dialogue system with a mixed authorial approach, where an author specifies what are an agent’s options in terms of behaviour for a particular state, but then is up to the agent to decide what to do. The agent chooses what to say based on its emotional and social relationships with the player/user.

One of the current limitations of FAtiMA is the lack of options on user interaction. Interaction has to be done by selecting options on a traditional screen display. FAtiMA-Toolkit does not support speech or text recognition. We believe this to be a serious problem because when interacting with increasingly socially believable agents, users will expect a more advanced and natural type of interaction which tends to be speech.

1.1. Objectives

The goal of Creating an Agent for Amazon’s Alexa is to provide FAtiMA-Toolkit with support for speech recognition capabilities.

There are several options to speech recognition support. One of such is Amazon’s Alexa speech recognition technology. Alexa uses its own automatic speech recognition algorithm to convert spoken words into text. Using Alexa has a speech recognition device with an agent created in FAtiMA would solve the limitations on speech interaction. It would allow for the development of capable FAtiMA agents with speech recognition.

To use the two technologies together, a simple agent was created to interact with the user. Integration between FAtiMA-Toolkit and Alexa was necessary. This integration was made through Visual Studio and Amazon’s Web Service, which provided the much needed services for this thesis. Developers will have an API to guide them through the procedure of integrating Alexa with FAtiMA. This will allow for an easier configuration and provides developers of FAtiMA with the option of using Alexa as the device of choice for their agent.

2. Related Work

In this section we are going to talk about the research on the relevant technologies for this thesis. We research some types of development tools for the creation of agents. Since the speech recognition technology was already chosen, we opted to research agent creation toolkits to better understand the alternatives.
2.1. Development Tools and Toolkits

2.1.1 Twine

Twine is an open-source tool which facilitates the creation of story-telling experiences[10][7]. Twine provides developers with the tools to create a simple story that can be extended with variables, conditional logic, images, CSS and JavaScript[10]. Twine, at its core, is a tool for creating hypertext. With the complexity of decision trees in videogames, Twine helps developers visualize the hypertext structure through a story map. This story map is the main focus of Twine. It provides developers with the necessary visualization of their dialogues. There are tools which help to create a story with multiple branching paths. It gives developers the possibility of adding removing and editing hypertext links between text in an interactive way.

Twine is focused on helping people starting the development of a video game with simple dialogue. This comes from the fact that Twine is only a dialogue management tool and does not have any Natural Language technology [7] which makes all the dialogue require authoring to be created.

2.1.2 Azure Bot Service

Microsoft’s own chatbot framework [1]. This framework aims to help developers by providing tools, templates and AI services. It offers a software development kit which has bot builder tools. These consist in a collection of cross-platform command line tools designed to cover bot development work flow. The tools provided are extensive. They organize their tools in five categories (The five development cycles).

In planning, they use a tool capable of creating mock conversations. This tool is named Chatdown. It uses a .chat file to define the conversations between the user and the bot. These files are then converted in .transcript which can be viewed in a simulator. The chat files contain two parts: A header that defines the participants and the conversation between the user and the machine. These files are defined by the developer and contain the conversations with the bot.

In the building phase, a bootstrap language understanding tool is used as well as a question and answer maker and dispatch model creators. The bootstrap language understanding tool is called LUDown. It is a command line tool and serves as a simple way of describing utterances, entities and intents for the bot.

Question and answer maker, as the name implies, is a library made to provide classes suited for the creation of question and answer dialogues in the Azure Bot Service. The dispatcher allows developers to dispatch between the previously talked components (such as: question and answer Maker, language understanding tool and custom code). For testing, a support tool called bot framework emulator is available. This is a cross-platform emulator that serves to test and debug the created bots.

Azure Bot Service is a competitor to Alexa but it is more focused on business. Since it is focused more in business, it does not work so well with agents which are focused in role playing, because of this, Alexa is more suitable to our work.

2.2. FAtiMA-Toolkit

FAtiMA-Toolkit is a collection of useful tools designed with the creation of emotional and social intelligence in mind [8][5][6]. FAtiMA-Toolkit will allow us to create our agent. In this toolkit, it is possible to define dialogue layers using code alone or by giving the chance to create those dialogues in an Integrated Authoring Tool. This is a windows forms application where authors can design their scenarios and characters. The integrated authoring tool works as an independent user interface app, has a Simulator and Dialogue Tree Generator. All the components present in FAtiMA-Toolkit will be described in this section.

2.2.1 Integrated Authoring Tool

The integrated authoring tool is part of the FAtiMA-Toolkit and serves as the center piece of the toolkit. There are three main sections on the authoring tool: Role Play Character, Dialogue Editor and world model. There is also a simuator used for testing.

Scenarios are the program’s domain and serve as the main file for the Authoring tool. These files are of a specific type (.iat) but are essentially a JSON file with information on the authoring tool, characters created and their respective dialogues, the simulator and the world model. Scenarios have the characters created by the developer. There is a specific editor for these characters. The agents and users/players are represented by role playing characters.

Role Playing Characters are generated by the developer and serve as the conversational agents for FAtiMA. These agents have a set of given components defined by the developers. Role Playing characters have an emotional state, a knowledge base, an autobiographic memory, emotional appraisal, emotional decision making, social importance and comme il faut. These are tools that ease the work of developers in the creation of rich characters emotional and social behaviour. The first components we will explain are the knowledge base and autobiographic memory.
The knowledge base and autobiographic memory are the components that deal with the storage of the agents beliefs (knowledge base) and the memory of past events (autobiographic memory). Beliefs are described in Well-Formed Names (WFN)[8][9]. WFN's are made of three components: symbols, variables and composed names. Symbols are the constant entities like actions, objects and name of relations. Variables, as the name indicates, represent values that are not defined. Composed names are the properties and relations of the agent. On the composed names, the agent can store information about the relations it has, for example: Paul likes to play games. If Paul tells the agent this fact, the agent stores this in the knowledge base. The composed names are used as a representation of that knowledge. Events, stored in the autobiographic memory, are also composed names with specific syntax. There are three types of events: Action-Start, Action-End and Property-Change [8][9]. Action-Start defines the beginning of the action. Action-End is the same but for the Action ending. Property-Change, as the name implies, changes the agent beliefs through the experienced event. This allows the agent to change knowledge based on what users say or do.

Emotional Appraisal controls agents Emotions based on the Ortony, Clore and Collins (OCC) Model. This model is based on the human emotional structure. The OCC Model defines emotions as the result of a cognitive appraisal process that interprets that same emotions. Emotions also represent the type of reaction the agent will have to certain events and its perception of the world[8][9]. The emotional appraisal is defined by a set of appraisal variables, which define the type and intensity of emotions. The process of defining agent's emotions has two main phase, appraisal derivation and affect derivation. Appraisal derivation is defined by the developer through the definition of the appraisal rules. Rules are defined by associating the appraisal variables to the events. Appraisal derivation simply determines which appraisal variables will be affected by what events. Affect derivation is a process made in the FAAtiMA-Toolkit and associates the triggered appraisal variables of said event to their respective emotions, changing the agent emotional state.

As said before, Dialogue trees are extensive and require a lot of authoring for just a simple conversation. FAAtiMA-toolkit uses a hybrid approach to dialogue through the use of states. The Dialogue Manager has a list of utterances with a specific syntax. The utterances have a current state, a next state, meaning and style and the utterance text. The current and next state show both the current state of the agent and the next state for the agent if that utterance is chosen. The meaning shows the context of the conversation and the style serves as the agent personality trait. The utterance represents what the agent or the user will say. The dialogue manager lists the possible choices of utterances for the agent in the current state. It does not decide which utterance to choose, that is made through the Meta-Belief which will be explained further. The dialogue manager is defined in the dialogue editor tab on the Authoring tool. The editor gives the developers the possibility of defining the utterances for the dialogue manager and their respective properties.

World model editor serves as the tool to define the state where the agent goes if said action is executed. The World model editor gives the developer the tools to define the consequences of the agent's actions specified in the dialogue manager. These consequences can influence both the agent and the environment.

Meta-Beliefs control complex action conditions, they are used for actions that need specific events[8][9]. There are several Meta-Beliefs already implemented. They range from a condition to do math operations to another where it gets the current mood of an agent. This allows developers to not rely only in knowledge-base beliefs. Meta-Beliefs are used in the decision making where the actions of the agent can have conditions controlled by Meta-Beliefs. The use of these is up to the developer and can be completely ignored.

These are the main components of the authoring tool, the application that will help us create and define our agent for Alexa.

2.2.2 FAAtiMA-Toolkit Limitations

Speech recognition is an ever increasing technology which has several uses. It is the ultimate form of interaction with agents as it is the most common form of interaction for us. It is also a bit difficult to support since it needs specific software (speech recognition software). The best way to support speech interaction is to use the already made technologies available in the market. Alexa is one of such and will be the other piece in the creation of this thesis. The next section is dedicated to explain Alexa and all it's components.

2.3. Alexa Skills Kit

Alexa skills kit is the main development resource in the creation of Alexa related applications. This is a development kit for Alexa [2]. An Alexa skill is the name Amazon gives to programs that work with Alexa. Alexa skills kit provides a plethora of tools, some of those include: Voice user interface, understanding users intents and a back-end cloud service [2] which will be explained in detail further.
Alexa skills kit has automatic speech recognition which converts spoken words to text and deduces what the speaker means by using natural language understanding as explained in their main website [2]. The kit provides native and custom voice interfaces, supports many languages, offers multimodal input and response, more specifically, text-to-speech and audio streams. It is also possible to add visual elements and touch inputs on Alexa compatible devices.

2.3.1 Alexa's Available Skills

As explained earlier, the usage of Alexa's skills kit will be necessary since our work focuses on making an agent for Alexa. Alexa's skills kit provides a variety of templates to aid developers in specific creations. There are four templates for specific skills. There is also the possibility of making a custom skill which will be explained further. All these "pre-built" skills templates define the possible requests Alexa can receive and their corresponding user invocations. They range from smart home API's to Music API's. They will be explained furthermore:

- **Smart Home Skill API** - As the name indicates, this skill API offers a template for the creation of Smart Home programs to be integrated with Alexa. Examples of uses range from the simplest: Turning Lights on and off; To more complex ones like: Querying a lock to see if it is currently locked.

- **Video Skill API** - This API gives the ability to create programs based on video services. As all the API's tend to provide, Video Skill API defines the requests it can handle and the invocations users can use for this type of program. This API is well suited for developers who want to create services based on playing movies, finding TV shows and many other video service applications Alexa might have.

- **Music Skill API** - This API is one of the simplest. It eases the process of creating utterances and requests based on music services. From listening to certain music to creating playlists, this API is focused on building music applications for Alexa.

- **Flash Briefing Skill API** - Flash Briefing Skill API provides tools to create a flash briefing type of applications. Like the other API's, the Flash Briefing Skills give a structure to better develop flash briefings of all types. Flash briefings can range from overviews on news, comedy and sports to more personal ones like an overview of the tasks and plans users have on their calendar. This API has preset answers for the most common flash briefings. The responses are defined in code.

For our work, the provided skill API's do not have what we need to develop our integration. Alexa skills kit gives the possibility of creating a custom skill [2][3]. A custom skill gives the possibility of defining our own requests. Furthermore, it gives us the ability to also define users invocations of said requests. This will be important for our work since we aim to create specific dialogues with Role Playing Characters.

2.3.2 Custom Skills

As any other pre-made skill, Custom Skills are built in the Alexa developer console. To work properly, custom skills need to be created with a specific set of components [2][4] due to how Alexa works. These are:

- **Intents** - To work with Alexa using Custom Skills, a set of actions called *intents* is needed. Intents separate the user's input and serve as the main functionality of a custom skill. Intents are made in conjunction with utterances which will be explained further.

- **Invocation Name** - A custom skill needs to have an invocation name to be able to be launched. An invocation name is the launch utterance for a skill. It is defined by the developer in the starting process for the creation of a custom skill.

- **Utterances** - These are defined by the custom skill developer and convey the sentences that a user will say. The defined utterances work in conjunction with intents. Each utterance is associated with an intent. An interaction model is created with the intents and their associated utterances. There is a set of built-in intents made by Amazon that specify general commands. There are built-in intents to start the program, to end the program and to deal with unexpected utterances. A good example of how utterances and intents work can come from a simple skill like turning on the TV. If users ask Alexa to turn on the TV, the ASR interprets the speech and sends the request to the corresponding skill. The skill needs to check if the request is within the specified utterances. If it is, the skill generates the JSON file with the intent of the request; If it isn't, the skill sends a fallback intent.

- **Cloud-like Service** - A cloud service is needed to create a skill for Alexa. Amazon has their
own web service for Alexa. Lambda. There is also the possibility of using a specific cloud service with HTTPS. AWS Lambda is recommended due to the fact that it has built-in compatibility with Alexa skills kit. These cloud-like services are needed because of how Alexa's hardware works in general. The specific functions created by the developers are placed in these cloud services. The AWS Lambda service is free for personal or research purposes. There are limitations to AWS Lambda on free use. Mainly limitations regarding computation time. The free tier has 1 million solicitations for each month. Solicitations count every time Alexa communicates with AWS Lambda to ask for something.

- Alexa Developer Console - The Alexa Developer console is a configuration console made to aid in the integration of all the above components. The usage of the Alexa Developer Console is needed to the development of any kind of Alexa Skill. This console serves as the central point of all these interactions. It helps in the configuration of the specified components and enables Alexa to redirect the requests effectively.

From the moment the user talks to Alexa to the moment he receives a response, there are a lot of interactions happening between the several components that make the Alexa skill. There is a specific interaction flow for Alexa's speech recognition and consequent request and response to the skill. This will be explained in detail on the next subsection.

2.3.3 Alexa's Interaction Flow
Alexa's processes of interaction with the user and further communication with AWS can be summarized in three main phases:

1. Alexa's recognition of customer intent - These interactions start by a user command. The user commands Alexa to launch a specific skill or asks a specific question. Alexa, through its Automatic Speech Recognition, recognizes the skill you intend to launch (or use) and transforms that speech into a JSON file. After that, through the form of intents, Alexa sends that same JSON file to a function created by the developer.

2. Developed function - This is the developer's function. It is a program created by the developer to handle the intended requests/intents sent by Alexa. This function receives the .JSON file with the intent and returns another .JSON file with the response for the user.

3. Alexa's skill interpretation - Lastly, Alexa interprets the received JSON file and generates the response to the user.

2.4. Amazon Web Services
Between an Alexa skill and the developer function there needs to be a cloud-like service or a local server made by the developer. Since the cloud-like services provided by Amazon have support for Alexa skills, we will use them. These are: Amazon S3, lambda, IAM and CloudWatch.

2.4.1 AWS Lambda
AWS Lambda is arguably the most important Amazon service. Lambda is a service that lets developers upload and run their code without having to manage servers. This saves a lot of time for developers and the only cost is the computational time required. There is a free service for Lambda focused on helping research and personal use. Lambda has support for Alexa skills. There are specific triggers for the skills which execute the uploaded program, making the process of uploading and executing a specific program for a skill very easy and intuitive. There are some limitations to the Lambda service. It has a maximum space of 13 MB for the uploaded program. This can be a problem if the function created by the developer exceeds the amount. Another service is needed if extra space is required.

2.4.2 Amazon S3
Amazon's Simple Storage Service (S3) has the name indicates, is a simple bucket service. It's similar to github and it serves the purpose of storing files. This is very similar to all other bucket services. The difference between Amazon S3 and the others is the support for Lambda and all other Amazon related services. This makes a difference when developers are working with Alexa since they can easily use Amazon S3 to store extra files that are too big for Lambda.

2.4.3 AWS Identity and Access Management
IAM console serves the purpose of managing the access to AWS services through the management of users and groups. These users and groups can be created and managed in the IAM console as well as permissions to the rest of the services. It is also possible to create and manage roles for the user programs. These roles define what services are available. All the programs have a role assigned which determines the permissions for that programs.
2.4.4 Amazon CloudWatch
CloudWatch is a service that provides monitoring and debugging capabilities. It is used to monitor applications, collect data, logs and events and provides with a view of the resources, applications and services running on AWS services. CloudWatch will be used for debugging. With the built in logs and alarms the developers can set, it is a powerful tool needed to support debugging in our program/function since we are going to run it in lambda.

3. Implementation
3.1. Initial Concept
After an extensive discussion about all the tools and technologies available, we set on making the integration between FAtiMA-Toolkit and Alexa. The initial idea was to have a program running a testing scenario and displaying it on a traditional display. With the display, we would show the dialogue options available to the user. Alexa would talk to the user with the agent’s responses. The user could respond by answering Alexa with their voice.

3.1.1 Alexa
Starting by Alexa, the original idea was to have the user talk to Alexa through the echo device. Alexa would then use its speech recognition to interpret what the user said and convert it to an utterance. Alexa would then communicate with a program made by us which served as a bridge between FAtiMA and Alexa. Alexa would send the user response and wait for the bridge to return what Alexa would say.

3.1.2 FAtiMA-Toolkit
In the FAtiMA-Toolkit, our original idea was to have a scenario showing the integration capabilities between FAtiMA and Alexa. The scenario would be created in the authoring tool with two characters, one for the agent and one for the user. The scenario would be used to determine the possible utterances for the user, and the answers of the agent. We wanted to have as little dialogue duplication as possible but Alexa skills needs to have the dialogue for the user. Since the FAtiMA-Toolkit scenario also needed to have the user dialogue, we ended up having to duplicate it. A more detailed description of this problem will be explained in a later section. The scenario made with FAtiMA would then be used in the Alexa and FAtiMA Bridge.

3.1.3 Bridge between Alexa and FAtiMA
After receiving the user response from Alexa, this program would use the dialogue provided by the FAtiMA scenario and send a request for the response of the agent and the options of the user. After getting both from the scenario, the bridge would send the agent response to Alexa and send the user options to the display.

3.1.4 Traditional Display
This traditional display would serve as a guide to what the user could say. It would receive the user options from the bridge and display them to the user. Unfortunately, Alexa only supports display of information on devices created by Amazon, for example, the echo show. This made it very difficult to display any information and it was discarded. Further explanation about the problems of displaying information will be present in later sections.

3.2. Approach
Our original Problem can be summarized as “How can we make Alexa work with FAtiMA-Toolkit?” For our solution, we developed a program that can use FAtiMA-Toolkit as the decision maker for the interaction between Alexa and the user. There were several challenges in making this program before it could reach its current form. The next sections will provide insight on how the program was made and the challenges behind integrating Alexa and FAtiMA-Toolkit.

3.3. Architecture
Our final architecture has several components working together, we have the scenario and its files, the Amazon S3, the AWS lambda running our program, which communicates with Alexa to send a response to the user. The way the architecture works starts with he user communicating with Alexa. The user says something, Alexa interprets and converts it to an utterance which will then be converted in an intent and sent to our lambda function (the program serving as a bridge between FAtiMA and Alexa) which is running in AWS lambda. The lambda function gets the scenario from the files present in the Amazon S3 and the agent response using the current state of the conversation as context. The agent response is then voiced by Alexa to the user. After this, Alexa waits for the next user response. This process is repeated until the dialogue ends. The next sections will explain in detail each of these components individually.

3.4. Technical Specifications of the Scenario
The dialogue for our scenario was made in the Authoring Tool’s Dialogue Editor. All the dialogue between the client and the agent was made by authoring and has a simple, tree-like structure. For example, if the user asks for a drink, the dialogue moves to a new state but the user can also ask a question, which will move the dialogue to a different
state, splitting it’s path in two. The state of the conversation is determined by one action present on the world model. This action is “Speak(*, [ns], *, *)” with the subject “[x]" and the target “[t]”. Using the previous action of asking a drink as an example, the subject is the user and the target is the agent. This has an effect on the world model, that uses its action of “Speak(*, [ns], *, *)” to change the dialogue state from the current, to the next ([ns]), which is the state of the action “asking for a drink”.

There are two characters in our scenario, they do not have emotional appraisal. These two characters have similar configurations, both have the knowledge base using a basic belief for their initial dialogue state: “DialogueState(Bartender)” for the user and “DialogueState(Client) for the agent”. The value of this belief is the first state of each character (“Start” for the agent and “S1” for the user). For the decision making of both characters, we used the action rule of “Speak([cs], *, *, *)” with the target being the opposite character and having no priority. The condition for this action is that the current state ([cs]) has to be equal to that of the opposite character or else it will not occur.

3.5. FAtiMA-Toolkit Program Development

The FAtiMA program loads the local files of the scenario into the program, this is done by calling the function “LoadFromFile()” from FAtiMA’s libraries. This function can be called by several FAtiMA assets such as the integrated authoring asset, the role playing character assets and the world model asset. Depending on the asset calling the function, “LoadFromFile()” loads that asset by receiving the local path to the scenario file containing said asset (it starts with “BarScenario.iat”) as an attribute. This was done in the initialization of the program. After loading all the necessary assets from the scenario, we needed to update the states of the conversation, this was made with an update function tested in a traditional display.

3.6. Update Function

Our update function is called when the program is initialized. This update function calls an auxiliary function that executes two times, one for the user character and another for the agent character. This auxiliary function, called “RPCUpdater()”, receives a character asset and a boolean. The boolean serves to check if the character is the user or the agent. “RPCUpdater()” is responsible for getting the next state in the dialogue and updating the current state of the scenario accordingly. This is done differently for the two characters, if it is the user’s character, we have to display all the possible options and wait for the user to choose one.

3.6.1 Agent Character Update

For the agent character, the “RPCUpdater()” begins by getting the dialogue decision of the agent, and updates the display with the answer. After that, the scenario needs to be updated with the changes made by the character. To update the scenario we used “Speak([cs],[ns],*,*)” and sent it to the world model where we ran a simulation to see the “consequences” of the agent decision and then used the “Perceive()” function. This function receives the property changes from the simulation and updates the agent with the new state of the scenario. This process is the same every time the agent needs to be updated.

3.6.2 User Character Update

The User update is made differently because of the need to wait for a response. If “RPCUpdater()” receives a user character, it lists the options in our testing display and jumps the update process. After this, if any option is selected, “RPCUpdater()” is called to update the user and repeat the process. After creating this update function (which will be used in the main program) we proceeded to the Alexa integration and the lambda function development.

3.7. Lambda Function Development

This is the main program for the integration, it runs our Alexa skill and is called a lambda function. The development of the lambda function started with the installation of all references and dependencies needed for the project. We started by using a template for Alexa skills in c3. This facilitated the work required to install and include the Alexa libraries and dependencies.

3.8. Lambda Function and FAtiMA-Toolkit Program Merge

The previous code made in the FAtiMA program was used for the integration between FAtiMA and Alexa. This code already had the functions needed to update the characters and scenario. This facilitated the work a lot since we could focus more on the Alexa side of development. The first task was to have Alexa respond with the first utterance of the testing scenario. This was not simple to make, we ended up having to use Amazon S3 as our bucket server to store the scenario files. This is due to AWS lambda only uploading the main code and the dependencies of the lambda function.

3.8.1 Amazon S3 Function

After the installation of all the proper packages for the Amazon S3 (the explanation is in a later section), we used an S3 function that allowed us
to read the contents of files available on our S3 bucket. We used a function called “ReadObjectDataAsync()” to get the contents of the files. This function receives two strings, a key and a bucket string. Both strings are global variables defined by us in our program. The key is initially the path to the main scenario file inside the S3. The bucket string is the bucket name we defined for our S3 storage. With these strings, the “ReadObjectDataAsync()” function gets the first file from the scenario and reads the contents. After the first, the key string is changed to the paths of the other files, this way we can read the contents of the other files of the scenario.

After getting the contents of all the scenario files, we used an alternative function in the FAtiMA library called “LoadFromString()”. This function receives a string of content and returns the type of asset being called. These assets can be the role playing characters, the world model and the integrated authoring tool. With these assets loaded into the lambda function, we could finally use our FAtiMA scenario with Alexa.

### 3.9. Intent Handling

Our program starts with the “FunctionHandler()”. This is our lambda function, it is called whenever there is a trigger for the skill. The “FunctionHandler()” works as the main function and starts the execution. In the case of being the launch request for the skill, the “ProcessLaunchRequest()” function is called. If it is not, the “FunctionHandler()” checks whether the intent received has slot values or not. Intents with slot values are processed differently. If the request is for a normal intent, the function “ProcessIntentRequest()” is called, if it is a intent with slot values, the function “ProcessDialogRequest()” is called instead.

#### 3.9.1 Launch Request

This function, as the name implies, processes the launch of the skill. This is where we initialized all the scenario files through the use of the previously mentioned “LoadFromString()” function. After loading the files, the “ProcessLaunchRequest()” (which receives a response body as an attribute) updates the response with the respective launch message for the skill, which is specified as a global string on the lambda function.

#### 3.9.2 Intent Request without Slot Values

The “ProcessIntentRequest()” function handles the requests for the intents without slot values. The type of intent is received as an attribute and is handled by a “switch()” that chooses the correct case for that intent. The response is then calculated by calling the aforementioned “RPCUpdater()” function. After receiving the response from the “RPCUpdater()” function, the “ProcessIntentRequest()” returns that response to the “FunctionHandler()”.

#### 3.9.3 Intent Request with Slot Values

The “ProcessDialogRequest()” function processes the intent similarly to that of the “ProcessIntentRequest()” function, the only difference is that depending on the slot value, we could have different dialogue responses and because of that, we needed to have a condition. For example, if the user asks for milk, the agent has a special response. We need to check if the user said milk or another drink. After checking these conditions, the handling of the intent is done in the same way as the previous one.

#### 3.9.4 Updating the Dialogue States for the Characters

The “RPCUpdater()” function is being used to calculate the response to the user. It is also used to update both the user character and the agent character. There were modifications to this function for it to work with Alexa. The FAtiMA scenario expects a specific answer from the user, since we only receive the intent, an overload of the “RPCUpdater()” was needed where the function received a string with the expected scenario answer, specified by authoring. The main “RPCUpdater()” function updates the agent and returns its response.

#### 3.9.5 Conversion to .JSON

All of the responses returned by these intent handling functions already come in a speech synthesis markup language format (ssml). This format is supported by Alexa and is used to change how Alexa responds. We used the standard ssml synthesis (“speak”). The conversion to this format is done in the intent handling functions by another function called “SsmlDecorate()” which takes the agent response and converts it.

After the responses are in ssml format, they need to be in a .JSON file, this is done in the “FunctionHandler()” where an external function “JsonConvert()” is called. The result is the response in .JSON which is returned to Alexa by using AWS lambda.

### 3.10. Final Touches

After implementing the intent handling functions, the integration was successful. We launched our Alexa skill and our agent started the conversation with the utterance from our testing scenario. After
this, the process of implementing the rest of the dialogue was simple. We had to create an intent for each of the user character utterances in the developer console and add support for the new intents in our lambda function code, using the previously mentioned intent handling functions.

4. Scenario Description
In this chapter, we will talk about the scenario used to test the integration between Alexa and FAtiMA-Toolkit.

4.1. The Scenario Setting
Our scenario is located in a bar. There are two characters in the bar, the client and the bartender. The agent is the Bartender and the user is the client. The goal of this dialogue is to ask for a drink and receive it. There are a series of questions and answers between the bartender and the client before the order is complete.

4.2. Characters
There are two characters in our Scenario, these are the client and the bartender. The client is the user character and the bartender is the agent's character. The conversation context is of a client asking for a drink in a bar. There are around 21 states for the conversation. They alternate between the client and the bartender. The bartender starts the conversation by asking "Welcome to Alexa's Bar. How may I help you?". After this, a response from the client is expected, the client can ask for a drink or ask questions. The conversation then continues with the bartender influencing the flow of the conversation.

4.3. Dialogue defined in the Alexa Skill
For the client, all the dialogue needed to be defined on the Alexa developer console. As said previously, Alexa needs to have the possible answers for the user. Even though this has the con of not allowing us to have Alexa as a speech recognition device only, it allows us to have more flexibility on the responses of the user. For example, if done only with FAtiMA-Toolkit, the client would only have two options to answer the bartender question of "What do you do for a living?". The original dialogue only had the options of: "I study" and "I work". With Alexa, we can easily use the slot values to add several types of jobs. Now it is possible for the client to be more specific like saying they are engineers or painters. This is useful for other steps of the conversation too where similar questions are asked.

The problem with this approach is that for more complex dialogues, having to define them in Alexa as well as FAtiMA is time consuming. The dialogue that needs to be defined in the authoring tool is much less than on the Alexa side but there are repetitions. We were not able to make the dialogue work completely independently from Alexa.

4.4. Emotional Appraisal
Our dialogue does not have emotional appraisal but it is still possible to use it with Alexa. Since the emotional appraisal only influences what the agent says, it works completely independently from Alexa. Including emotional appraisal in an agent for Alexa is as simple as creating it on the FAtiMA-Toolkit as you would normally do.

5. Conclusions
5.1. Contributions
FAtiMA-Toolkit and Alexa Integration - The main contribution of this project is the integration itself, it can be used to create interesting agents in FAtiMA with the Alexa speech support. With this integration, developers finally have the possibility of using FAtiMA with voice recognition, it allows developers to speed up the process of having speech on their agent if they so choose. By having the integration already done, developers can focus on making a better agent instead of having to split the work between the support for voice recognition and the creation of the agent. Alexa also allows for more diversity in responses with the development of intents. Since Alexa works with voice, there is no need to use limited option selection for the user answers. Developers can easily specify user responses through Alexa’s development console. As said previously, the use of voice as the primary tool for communication makes the conversation between the user and the agent more engaging and realistic.

Bar Scenario - We also provide our bar scenario. Even though it is a simple scenario, it shows how the integration can be beneficial for future development. It can also be used separately from Alexa with the traditional options display.

5.2. Future Work
Future work is dependent on what developers want for their FAtiMA agents. Speech recognition is now supported and can expand the complexity of these agents.

Further research can be done on how to mitigate the dependency of Alexa’s skills on the intent definition. Research can also be done on how to display the dialogue options for the users and integrate that with the Alexa skills.

Our scenario can be improved and made more interesting and fun with emotional appraisal and more meaningful options. Our Alexa skill can also be built upon to have even more flexibility for the user responses. Slot types can be expanded to accommodate more options for the drinks and the
jobs.

References