Platform for the creation of cross-platform voice applications

Rita Canavarro and António Nestor Ribeiro

HASLab, INESC TEC and University of Minho
Largo do Paço, Braga, Portugal
`a74484@uminho.pt, anr@di.uminho.pt`

Abstract. Due to the growth in the usage of digital assistants there was a need to expand their functionalities in order to cater to a wider range of users and their necessities. This need led to the creation of voice applications. Voice applications are still relatively new and as such there are still not that many tools, established architectural patterns or even a standard methodology that can be used in the development process. This problem is even bigger if we are addressing cross-platform applications, given that there is nowadays a plethora of different vendors of integrated digital assistants. The lack of a standardized methodology means that the developers will end up using the methodology(ies) that seems the most adequate concerning the purpose of obtaining a stable product. With this in mind, a research about the different application models for voice applications was conducted. This research led to the creation of a platform, with an incorporated visual editor, that promotes a platform-independent development process of cross-platform voice applications and its automatization. This platform allows a developer to create a language model template that will later be used to generate platform-specific models to define the frontend and boilerplate code to develop the backend functionality. By using this platform, the developers will be able to create and deploy voice applications for Amazon Alexa and Google Assistant from a single source of information despite the differences in their application models and, most important, resorting primarily to the intended requirements and not only to technological aspects.

Keywords: Voice applications · Digital assistants · Software engineering

1 Introduction

Digital assistants have as main objective to help people on their daily tasks such as sending a text message [6] or through voice commands or autonomously due to the gathering and processing of a particular type of data. In order to extend their functionalities, making them more appealing to the consumers market, companies such as Amazon and Google decided to provide SDKs for the development of voice applications. Voice applications can act as an interface for mobile/web applications through voice commands [6] or be independent applications that exist only for voice-first platforms.

Nowadays, what is happening in the field of voice applications is similar to what happened when mobile applications first appeared. When developers try to develop a voice application they feel tempted to apply, for instance, the methodologies that they use when developing web/mobile applications, which might not be a good idea due to the transient and invisible nature of voice applications. In this field, there are already some development tools and methodologies available yet a standard methodology, that defines a set of rules and the models that can be developed to specify certain components of the application such as the conversation model, still hasn't been defined. The focus on platform-specific tools/methodologies and the absence of a standard methodology makes the development of cross-platform voice applications more complex and time-consuming and might make the developer more focused on the different technological aspects rather than on the application requirements.

A platform that aims to promote the development process of cross-platform voice applications and its automatization is proposed in this paper. The objective is to provide a platform, with an incorporated visual editor, for the creation of voice applications, where from a generic specification it should be possible to generate and deploy them to various systems. This platform will be built towards the digital assistants Amazon Alexa and Google Assistant, given that the differences on their application models don't prevent the use of the proposed platform-independent development process. The frameworks Jovo and Violet will also be supported in order to allow the use of this development process and one of those frameworks together.

The platform core component is a cross-platform language model template, whose primary goal is to allow the specification of the application. The internal structure of the template is going to be transparent to the developer, which means that he can be focused on what he wants to specify in each component, via the visual editor, and not on how the whole template should be structured in order to be cross-platform. The template allows the definition of the frontend and also the generation of boilerplate code for the initial development of the backend. High-level UML activity diagrams, which allow the definition of the conversational model and the outlining of the application functionalities, were also developed to help in the definition of the components of the language model.

This paper describes the steps that were taken towards the development of the platform. The remnant of the paper is organized as follows. Section 2 will cover the necessary background regarding the aforementioned digital assistants and frameworks and the components of a voice application. Section 3 will present the general steps for developing a single-platform voice application, the high-level activity diagrams and the platform. Lastly, section 4 will conclude the paper.

2 Background

This section presents a brief review of the background relevant to the work present in this paper. The background will be focused on the digital assistants and frameworks chosen and on the structural components of a voice application.

2.1 Digital Assistants and Frameworks

Amazon Alexa This assistant was officially released in November 2014 [3], which makes it one of the oldest digital assistants next to Siri and Cortana. Amazon was the first company to provide open-source tools to developers in order for them to be able to develop voice applications, which Amazon named Skills [7]. By making this decision, Amazon allowed Alexa to grow rapidly in terms of functionalities available to its users in comparison with the rest of the digital assistants present in the market [15].

Google Assistant This digital assistant was officially released in May 18, 2016 [9]. The Assistant is the successor of Google Now, being that it inherited all of its research capabilities and still added some new functionalities [11] due to Google advances in the area of Artificial Intelligence. Moreover, the Assistant also improved the interaction with the user by being able to talk with him [10] and at the same time learn the user preferences [11]. Google, in order to be competitive in the digital assistants market, also allows the development of voice applications, to which it refers as Actions [13].

Jovo Jovo is a cross-platform framework that was created to allow a developer to develop voice applications, from the same codebase, for Amazon Alexa and Google Assistant. Given that these two assistants share some structural similarities, this framework makes use of that characteristic and allows developers to develop generic JavaScript (Node.JS) code without losing the ability to use platform-specific backend functionalities[8]. However, even though Jovo provides a proprietary language model for the development of platform-specific language models, the developer will have to specify it entirely by hand in a JSON file, which can become quite cumbersome the more complex the application becomes. Additionally, if there is the need for platform-specific functionalities, the developer will have to add to the Jovo model the proprietary language models of each assistant. This implies that, if in the future, the developer wishes to make a single-platform functionality present in both assistants, he might have to refactor a considerable portion of its language model by hand.

Violet Violet is a cross-platform framework that was created in order to ease the development of high-end voice user experiences by helping developers in the definition and use of conversational flows. This framework offers support to the development of conversational bots and voice applications [14]. Violet uses JavaScript (Node.JS) and a HTML-inspired language to develop the conversational flow between the user and the application. Furthermore, it also allows the use of plugins to ease the call to external APIs, such as a plugin to make a connection to a PostgreSQL database. It should also be mentioned that the language models generated by Violet for Google Assistant are still not well structured. The generated model for Google Assistant lacks the arguments and their types, that are defined in the backend code, which means that the developer will have to specify them by hand on the DialogFlow console and also correct the phrases where they were supposed to appear. Lastly, Violet is still in the beta development phase and it's currently in version 0.16.

2.2 Components of a voice application

In order to be able to develop the proposed platform, it was conducted a study about the necessary components in a voice application for Alexa and Google Assistant. The frontend is developed by defining a language model, that describes how the application can be invoked, what it can do and how the user can ask for the functionalities available. By defining this model, the developer will provide the assistant with a wide range of important information along with a mapping tool that allows the assistant to map each one of the user requests to its corresponding function on the backend. The information can be the phrases that can be used to request a functionality or the data that is necessary, from the user side, to fulfill its request. The conducted study led to the conclusion that both assistants have the same components in their language models, even though they have different structures and some components have different denominations. Figure 1 presents a UML domain model diagram, that specifies the language model components and the relations that they maintain among them.

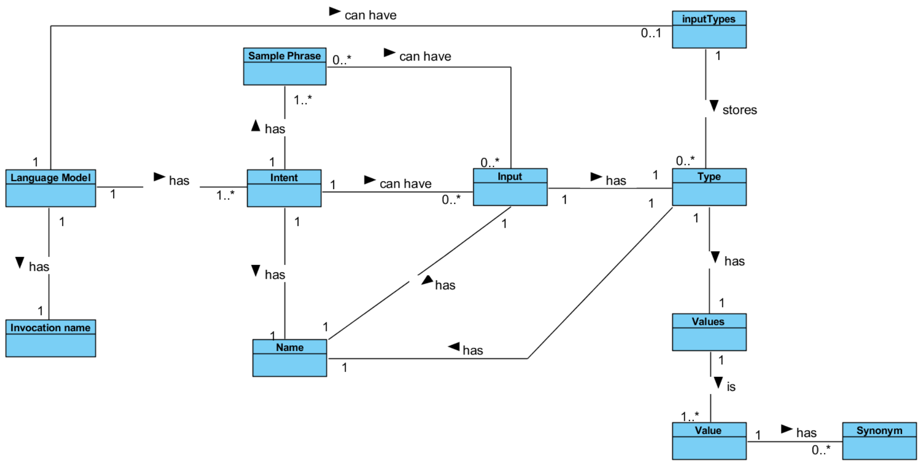

Fig. 1. Components of the language model

- Invocation name, that corresponds to the name that the user will use to invoke the voice application;
- Intent, that represents a functionality of the application. Each one of the Intents will be composed by a name and by a set of sample phrases, which Amazon denominates of utterances and Google of User expressions, that represent what the user will normally say to invoke the functionality;

– Inputs, which Amazon denominates slots and Google entities, that represent the data that the user must provide in order for the application to be able to fulfill its request. The Inputs are associated with the sample phrases as they must be placed in the phrase in the place that is most likely for the user to say them. Each one of the Inputs is composed by a name and a type;
– Input Types, that allow the definition of developer defined types;
– A type that is composed by a name and by a set of values that the Input might take. Furthermore, these values can have synonyms, that are defined in order to increase the number of ways that the user has of saying a certain word and therefore provide a certain field [1]. This makes the dialogue more flexible.

The backend will be constituted by handler functions that will support all the functionalities of the application and by a routing handler that will handle the incoming HTTP POST requests. There might also be the need to persist information in a database that will, for instance, improve the future use of the application or store the current conversational state that the application is in, so that the application may know which handler function to execute. Additionally, the backend might also have to use external APIs in order to have access to more resources and/or extra functionalities.

3 Methodology

In this section, the development steps of the platform and of the high-level activity diagrams, will be described alongside with a general explanation about the development of a voice application for the digital assistants Alexa and Assistant.

3.1 Single-platform development

In order to expose the similarities and differences that exist among the application models of Alexa and Assistant and to enhance how the developed platform took said similarities and used them to promote a platform-independent development process, a tabular explanation (Table 1, Table 2), that focus on the general development steps of a single-platform voice application, was elaborated.

3.2 High-level activity diagrams

Typically, when developers begin the definition of the conversation model of their voice application, they are faced with several approaches and one of them is the definition of high-level flow diagrams [4][5]. Regarding the high-level flow diagrams, not all developers feel confident in developing them due to the high probability of committing the mistake of iterating too much the definition process. This mistake happens when the developer tries to describe every possible next step in the conversation between the user and the application. Such task can end up being a potential infinite exercise given that the users have freedom of speech when they are using the application. If this mistake isn't revised, the voice application will be very similar to an interactive voice response and not to a natural dialogue with another person [2]. A different approach to the definition

Table 1. Development of the Frontend of voice applications

Digital assistants	Language Model	Application responses
Amazon Alexa	Definition through a visual editor or a single JSON file.	Define the responses on the backend code and send them on the response output. Can also define a dialog model to handle confirmation of an Intent and gathering and confirmation of an Input value.
Google Assistant	Definition through a visual editor or multiple JSON files (two for each Intent and Input).	Define the responses on the backend code and send them on the response output or define them on the model schema.

Table 2. Development of the Backend of voice applications

Digital assistants	Similarities	Differences
Amazon Alexa	Start with importing all the necessary modules and performing the needed configurations. Develop a handler for each functionality.	Each handler is going to be composed by two different functions. A function denominated **canHandle**, that allows the definition of the activation rules, and a function denominated **handle**, that has the code that will fulfill the user's request.
Google Assistant	End with the definition of a routing handler to establish the entry point for HTTP POST requests.	The handler is composed by only one function, that fulfills the user's request, given that there is no need for activation rules.

of the conversation model is proposed in this paper. This approach consists in the development of high-level UML activity diagrams given that they provide a well-defined notation and development rules, help in the standardization of the development process and are a platform-independent tool. However, the development of activity diagrams follows a certain set of rules that might not give the developer enough freedom to express what its voice application is going to do. Additionally, if the developer has never used UML he will have to spend some time learning the notation and rules inherent to the development of activity diagrams. Nonetheless, it's considered that, despite the possible disadvantages of using UML activity diagrams, they are still an acceptable approach when compared to high-level flow diagrams. The activity diagrams will be used to specify the application functionalities, an example of a phrase that can be used to invoke the functionality, and the main flow of conversation between the user and the application. These diagrams will help in the development of the language model components (section 2.2) because the developer will already have a solid notion of the functionalities (Intents) and their sample phrases and if they will need to receive arguments (Inputs) from the user in order to fulfill its request.

Furthermore, the activity diagrams can also be used to demonstrate the interactions between the user and the application during the fulfillment of a request (a more detailed flow of conversation), so that the developer can specify what he wants its application to respond back to the user when a certain functionality is being performed. The activity diagram (Fig.2) presents the main menu of a

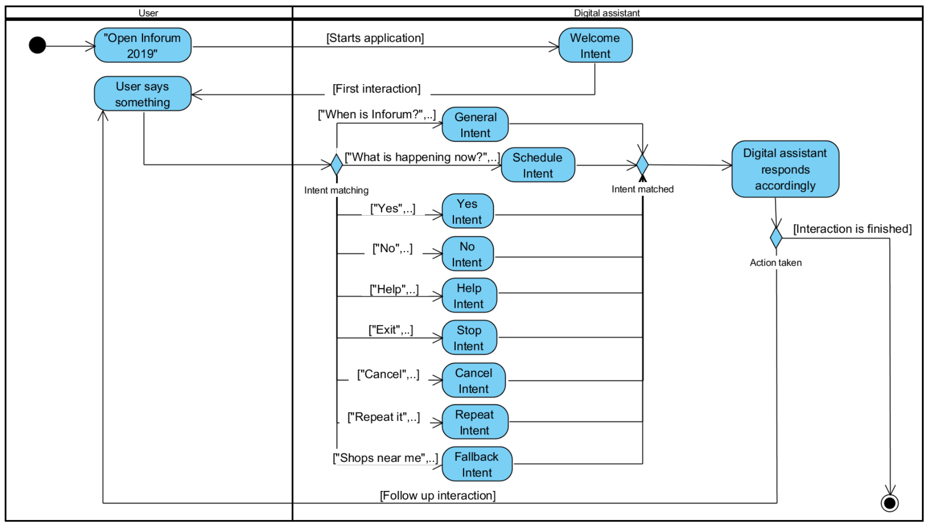

Fig. 2. Conversation flow between user and application

voice application that tells the user the general information about INForum and its schedule. By developing this type of diagram, the developer can specify what the user can do in the application without having to detail all its possible steps. For instance, the user can go to the Schedule Intent after being welcomed into the application and to demonstrate it, it wasn't necessary to specify that before he might, or not, have asked for another Intent. To specify the inner path of a functionality, that corresponds to the flow of conversation between the user and the application during its fulfillment, as it was aforementioned the developer should specify a more complete activity diagram. Therefore, this tool allows the developer to specify the main flow of conversation of the application and the functionalities that it will offer, without having to make prior commitments of what paths must necessarily be followed in order for a request to be fulfilled.

3.3 Platform

In the current approaches, the development of a cross-platform voice application implies the definition of the same language model for each digital assistant that the application will be in. Given that those models will most likely have different structural rules, the developer will have to spend part of its time performing duplicate work. The platform proposed in this paper (Fig.3) aims to remove the duplicate work and to streamline the definition process by allowing the definition of a cross-platform language model template (e.g listing.1.1) that can be used to generate platform-specific language models and boilerplate code for the inital development of the backend functionality. Furthermore, the platform also allows the deployment of the language model to the Alexa Developer Console and DialogFlow and the source code to AWS Lambda, as well as the deployment of both the language model and the source code through Jovo. The deployment process consists in gathering the necessary credentials from the user and in generating the corresponding language model and source code files. These files will then be deployed to the chosen digital assistant(s) and cloud-server platform.

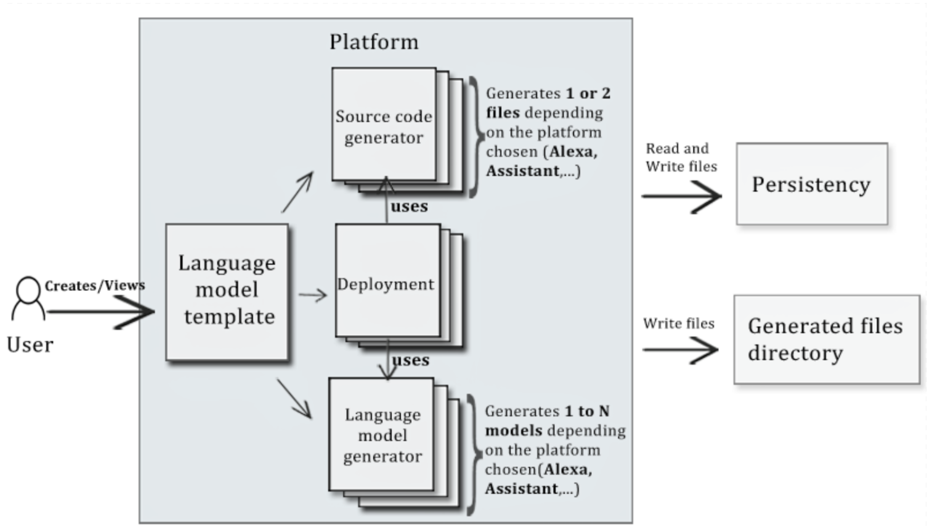
Platform
Source code generator
Generates 1 or 2 files depending on the platform chosen (Alexa, Assistant,...)
uses
Read and Write files
Persistency
Creates/Views
User
Language model template
Deployment
uses
Write files
Generated files directory
Language model generator
Generates 1 to N models depending on the platform chosen(Alexa, Assistant,...)

Fig. 3. Platform Layout

Cross-platform language model The cross-platform language model template, proposed in this paper, is based on the models of Amazon Alexa and Google Assistant. Regarding the frameworks, Jovo's language model wasn't considered as it's inspired in the model of Alexa, and given that Violet doesn't have a proprietary language model and uses the models of Alexa and Assistant, it wasn't considered as well. It was decided to name the language models, that the platform uses and that the developers will specify, language model templates because they will serve as a starting point to the generation of the platform-specific language models and boilerplate code. The idea for the cross-platform template originated from the study that was conducted upon the language models of both digital assistants (Subsection 2.2). As it was aforementioned, this study led to the conclusion that the language models of both assistants share the same structural components. This characteristic confirmed that it was indeed possible to define and generate a cross-platform template of the language models. However, even though they have the same components, the structural rules for the output of the model in JSON files differed among them (Table.1). In order to surpass this difference, it was decided that three different language model generators should be developed, so that the structural rules of both digital assistants and Jovo would be respected. Regarding the cross-platform template, it will have the common structural components of the language models and the necessary relationships among them. This template (Listing 1.1) will then be used to feed the generation process (Fig.3) in order for the language model generators to be able to produce their corresponding platform-specific models (Listing 1.2). The class diagram (Fig.4) depicts the components of the cross-platform language model template and their relationships. The platform has two types of templates, the

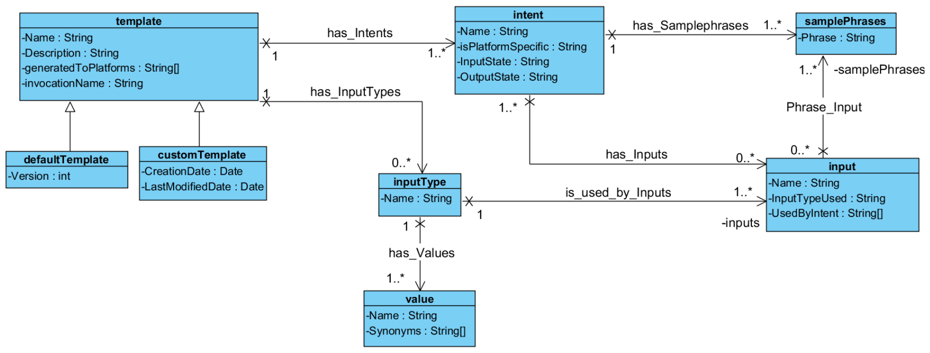

Fig. 4. Components of the cross-platform language model template

default templates and the custom templates. The default templates will come with the platform and will serve as examples to the developer. These templates are read-only, but it's possible to generate platform-specific language models and/or source code and also deploy them. The custom templates correspond to the new language model templates (e.g listing 1.1) that the developer can specify using the visual editor provided by the platform. In order to provide more options, the developer can also mark a functionality as platform-specific, which means that it will only appear in the language model of the intended assistant. Furthermore, the platform also makes available a list of built-in Intents and Input Types, that are provided by the digital assistants, so the developer can use them as if they where platform-independent.

```
{(...)
  "_Name": "INForum2019",
  "_Description": "Voice application for the Inforum
    symposium",
  "_GeneratedToPlatforms": ["All","Jovo"],
  "_InvocationName": "inforum guide",
  "_Has_Intents": [(...)
    {"_Name": "ScheduleIntent",
      "IsPlatformSpecific": "All",
      "InputState": "",
      "OutputState": "Schedule",
      "_Has_Samplephrases": [{"_Phrase": "what is the
    schedule for {location} on {date}"}, {"_Phrase": "where
    {paperName} is being presented"},(...)],
      "_Has_Inputs": [{"_Name": "topicName",
          "_InputTypeUsed": "TopicsINForum",
          "_UsedByIntent": ["ScheduleIntent"],(...)
    },(...)]},(...)
}
```

Listing 1.1. Language model template of the Inforum voice application

```
{"invocation":"inforum guide",
"intents":[{
        "name":"ScheduleIntent",
        "phrases":[
            "where {paperName} is being presented",
            "tell me what is going to be presented at
   {location} on the {date} at {time}",
            (...)],"inputs":[{
                "name":"topicName",
                "type":"TopicsINForum"},
            {
                "name":"date",
                "type":{"alexa":"AMAZON.DATE",
                "dialogflow":"@sys.date"}
            },(...)]}, (...)],
"inputTypes":[{"name":"TopicsINForum",
        "values":[{"value":"Mobile and Ubiquitous Computing",
                "synonyms":["Mobile
   Computing",(..)]},(...)},(...)],
(...)}
```

Listing 1.2. Platform generated language model example for the Inforum voice application

Generation of boilerplate code The information present in the cross-platform language model template allows the generation of boilerplate code for the initial development of the backend functionality of a voice application. In order to generate the boilerplate code, three code generators were implemented. There is the need for three different generators because, like in the output representation dilemma, each digital assistant and framework has a different approach to the codification of the backend. A code generator for Violet wasn't developed as this framework is still in beta (section 2.1). In this case, the template (Listing 1.1) will also be used to feed the generation process of boilerplate code (Fig.3). The boilerplate code will be generated for JavaScript (Node.JS) given that its supported by both assistants via a SDK and is the programming language used by Jovo. Regarding the code structure (Listing 1.3), it will have all the necessary configurations (e.g routing handler, etc) and the skeleton code of each one of the functionalites that the developer defined. Furthermore, the setup of a database and the definition of conversational states will also be supported. Conversational states are used to maintain the current context of the conversation between the user and the application in order to be able to know which functionality should be activated by the user's request.

```
const {App} = require('jovo-framework');
const {Alexa} = require('jovo-platform-alexa');
const {GoogleAssistant} =
   require('jovo-platform-googleassistant');
const {JovoDebugger} = require('jovo-plugin-debugger');
const app = new App();
app.use(new Alexa(),new GoogleAssistant(),new
   JovoDebugger());
app.setHandler({
    LAUNCH(){return this.toIntent('WelcomeIntent');},
    GeneralIntent(){ },
    ScheduleIntent(){ },
    (...)
    END(){ },
    Unhandled: function(){this.toIntent('FallbackIntent');}
});
module.exports.app = app;
```

Listing 1.3. Platform generated code example for the Inforum voice application

4 Conclusion

The first step towards the standardization of the development process of cross-platform voice applications is the creation of platform-independent tools and mechanisms that can increase the efficiency of the process and reduce the time that the developer has to spend performing duplicate work because of platform-specific rules. The absence of standardization makes the development of cross-platform voice applications more complex and time-consuming for the developer due to the current plethora of different digital assistants that have their own application models and development tools. Regarding the developers perspective, the insuccess and low retention rate of their voice applications [12] may demotivate them from developing more applications of this type. That may lead to a general disinterest for this field and thus to a small developer community and slower advances on this technology. However, the main issue with the absence of standardization is that the developers don't have a common language to use to share ideas or development steps with one another and platform-independent rules to guide them towards the development of voice applications that have a good user experience across digital assistants. The high-level activity diagrams are a tool that allows the developers to more easily model the frontend and communicate ideas with others (e.g client, etc). The proposed platform aims to promote a platform-independent definition of the frontend and to automatize the development of cross-platform voice applications, thus making it easier to develop and test them and also to maintain a consistent user experience throughout devices. The core component of the platform is a cross-platform language model template that was achieved by analyzing the common components of the digital assistants' language models. The platform allows the developers to only have to define one language model template in order to specify and generate the frontend and the boilerplate code for the backend functionality, for a cross-platform voice application. Namely, this will lead to less errors, that could appear due to having to define the same language model more than once because of digital assistants' rules, and to a more efficient development process. Lastly, the platform allows the developer to be more focused on the requirements and fundamental parts of the voice application (e.g functionalities, user experience, etc) rather than on platform-specific details.

References

1. A.Coates, D.: Voice Applications for Alexa and Google. Manning (2018)
2. Amazon: Script out. Online, accessed: 2018/11
3. Etherington, D.: Amazon Echo Is A $199 Connected Speaker Packing An Always-On Siri-Style Assistant. Online, accessed: 2018/12
4. Google: Write sample dialogs. Online, accessed: 2018/11
5. Goossens, F.: Designing a VUI Voice User Interface. Online, accessed: 2018/10
6. Hoy, M.B.: Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants **37**(1), 81–88 (2018)
7. Isbitski, D.: Introducing the Alexa Skills Kit, Enabling Developers to Create Entirely New Voice Driven Capabilities. Online, accessed: 2018/11
8. Konig, Jan., S.A.: Jovo Platform Specific Features. Online, accessed: 2018/11
9. Linley, M.: Google unveils Google Assistant, a virtual assistant thats a big upgrade to Google Now. Online, accessed: 2018/11
10. López, G., Quesada, Luís, G., A, L.: Alexa vs. Siri vs. Cortana vs. Google Assistant: A comparison of Speech-Based Natural User Interfaces (2017)
11. Maiolino, T.: Maximus - Automatizando Tarefas por Voz (2017)
12. Marchick, A.: The 2017 Voice Report by Alpine (fka VoiceLabs). Online, accessed: 2018/10
13. Miller, P.: Google Assistant will open up to developers in December with 'Actions on Google. Online, accessed: 2018/10
14. Sinha, V.: Open Sourcing Violet A Voice Application Framework. Online, accessed: 2018/10
15. VoiceLabs.co: The 2017 Voice Report - Executive Summary p. 12 (2017)

5 Acknowledgments

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project: UID/EEA/50014/2019.