Received: 11 March 2022 | Revised: 2 December 2022 | Accepted: 8 December 2022

DOI: 10.1002/spe.3182

SHORT COMMUNICATION

WILEY

Implementing scripted conversations by means of smart assistants

Moisés Pacheco-Lorenzo[1] | **Manuel J. Fernández-Iglesias**[1] | **Sonia Valladares-Rodriguez**[2] | **Luis E. Anido-Rifón**[1]

[1]atlanTTic, University of Vigo, Vigo, Spain

[2]Artificial Intelligence Department, UNED, Madrid, Spain

Correspondence
Manuel J. Fernández-Iglesias, atlanTTic, University of Vigo, Vigo, Spain.
Email: manolo@uvigo.gal

Funding information
Ministerio de Economía y Competitividad, Grant/Award Number: PID2020-115137RB-I00; Ministry of Science, Innovation and Universities, Grant/Award Number: FPU19/01981; University of Vigo/CISUG, Grant/Award Number: openaccesscharges

Abstract

Smart assistants are among the most popular technological devices at home. With a built-in voice-based user interface, they provide access to a broad portfolio of online services and information, and constitute the central element of state-of-the-art home automation systems. This work discusses the challenges addressed and the solutions adopted for the design and implementation of scripted conversations by means of off-the-shelf smart assistants. Scripted conversations play a fundamental role in many application fields, such as call center facilities, retail customer services, rapid prototyping, role-based training or the management of neuropsychiatric disorders. To illustrate this proposal, an actual implementation of the phone version of the Montreal cognitive assessment test as an Amazon's Alexa skill is described as a proof-of-concept.

KEYWORDS

cloud computing, conversational agent, human–computer interaction, Montreal cognitive assessment test, scripted conversation, smart assistant

1 | INTRODUCTION

Smart speakers[1] are voice-activated speakers that interact with human users by responding to voice commands. These devices interact with other elements in the digital home ecosystem and back-end services by means of Bluetooth, NFC, Wi-Fi and other wireless protocols. A typical smart speaker is programmed to perform different tasks on behalf of the user, such as accessing entertainment content and information sources, listing daily activities, providing assistance with home chores, managing communications and interacting with smart home devices, among many others. This rich portfolio of features promoted an increasing demand over other technological products and services at home.[2]

Smart speakers (cf. Table 1) are powered by an artificial intelligence-driven virtual assistant with speech recognition, natural language processing and speech synthesis capabilities. As such, virtual smart assistants in smart speakers can emulate human conversations and therefore act as conversational agents. The artificial intelligence-enabled computing capabilities of these agents are not hosted by the smart speakers themselves, but are provided by their vendors

Abbreviations: AI, artificial intelligence; AWS, Amazon Web Services; JSON, JavaScript Object Notation; MoCA, Montreal cognitive assessment; NFC, near-field communication; NLP, natural language processing; PARADISE, PARAdigm for dialogue system evaluation; SaaS, Software as a Service; T-MoCA, Montreal cognitive assessment, telephone version; TV, television.

TABLE 1 Most popular smart speakers and assistants

Vendor	Devices	Assistant	User SW	Dev. platform	Dev. cloud
Amazon	Echo	Alexa	Skill	Alexa console	Amazon Web Services
Apple	Homepod	Siri	App	Apple developers	Google Cloud
Google	Nest	Google	Actions	Google developers	Google Cloud

in a cloud-computing environment according to the Software as a Service model (SaaS[3]). In other words, when the user interacts with a smart speaker using voice commands or phrases, the captured audio information is sent through the network to the processing back-end, and in some cases an appropriate response is computed and sent back to the speaker to be uttered by it. Besides smart speakers, virtual assistants can be hosted by most smart off-the-shelf devices, such as computers, mobile phones, television sets, watches, or earphones.

As pointed out above, partnerships between manufacturers of smart appliances and those of smart speakers promoted the integration of their smart assistants with last-generation TV sets, streaming devices, connected cars, smart watches and many others. As a consequence, they became ubiquitous, and any piece of software developed for a smart assistant became immediately available everywhere, to everybody, anytime. Due to the supporting SaaS model of commercial off-the-shelf smart devices, the resources required to implement smart assistants are fairly modest.

The development of applications for smart devices is in most cases supported by their creators through cloud-based developers' platforms.[4,5] Typically, registered developers are offered a collection of tools to design, implement, and deploy their applications in a controlled environment. This approach dramatically simplifies and expedites application development, but also limits the features and functionalities available to those supported by the provided tools and, in some cases, in a very restricted way. For example, access is not granted to users' raw speech signals, but just to the already processed plain text resulting from the speech-to-text conversion process performed at the back-end. Moreover, recording spontaneous data in real-world situations requires protecting the privacy of those involved, which in turn involves ethical and legal considerations.[6]

No matter their limitations, off-the-shelf smart assistants are still well-suited to implement conversation-based interactions, and more specifically scripted conversations or dialogs. In their most simple form, scripting chatbots are provided with the responses to a collection of keywords, which in turn are extracted from text-converted users' utterances.[7] Whenever the assistant detects a keyword, the corresponding response is articulated, and the assistant waits for a new user's utterance according to the script. More advanced versions rely on conversational artificial intelligence techniques[8] to identify a wide collection of user requests without being explicitly trained on every possible variation.

Scripted conversations play a fundamental role in many application fields, such as call center facilities,[9] retail customer services,[10] rapid prototyping,[11] role-based training,[12] autism management,[13] or the management of other neuropsychiatric disorders.[14] As a general approach in these applications, a conversation tree is generated beforehand collecting all possible conversation sequences, which in turn are composed of a succession of cues detected from users' utterances and their responses from the assistant. Then, the user or the assistant triggers a conversation, which will proceed according to the script until an endpoint is reached. Note that conversation scripts may include cycles to handle errors, missing or unidentified keywords, or as a consequence of the specific characteristics of the application field (e.g., to handle a new user request for service once the previous one is completed).

Despite being conceptually simple, implementing scripted conversations by means of off-the-shelf smart assistants poses relevant challenges. As pointed out above, developers do not have access to raw speech signals, but to the text obtained after being processed by speech-to-text algorithms. This information, together with additional time indications (e.g., timeouts when no response is detected, time delays, etc.) is the only information available from speech interactions. This limitation becomes especially challenging when relevant information is encoded in non-textual data, such as non-textual responses (e.g., noises, onomatopoeias, etc.), prosodic features or simply when measuring reaction time is required.

The aim of this work is to discuss the challenges addressed and the solutions adopted for the design and implementation of scripted conversations using smart assistants. This work is illustrated with the development of a conversational agent aimed to assess the cognitive status of an individual by means of the telephone version of the Montreal cognitive assessment test (T-MoCA). This assessment test[15] generates a total score with a maximum of 22 points, eliminating the items from the original test requiring visual stimuli or the use of paper and pencil. It was successfully administered via

landline telephones and cellphones, without requiring more advanced technological equipment or proficiency, such as videoconferencing. The original MoCA test has a discrimination power similar to the Mini-Mental State Examination test,[16] presently the reference test to detect cognitive impairment. Therefore, T-MoCA is a promising approach to design a screening solution to detect mild cognitive impairment based on smart conversational agents.

Although the solutions adopted in this particular case are applicable to any major smart assistant platform, Amazon's Alexa smart assistant solution was selected as our development environment for this proof-of-concept implementation, because of its popularity and the availability of developers' tools and support. The next section introduces the basic details of the development of a conversational agent. First, the standard development workflow is described together with the common characteristics of smart assistant's back-end applications. Then, the most relevant aspects of the implementation of a scripted conversation are discussed in Section 3, and a standard evaluation of the implementation's performance is presented in Section 4. The solutions adopted to address the challenges encountered are discussed in Section 5 and finally, Section 6 summarizes the scripted conversation model introduced in this article and offers some concluding remarks.

2 | MATERIALS AND METHODS: BASICS OF A CONVERSATIONAL AGENT

The development of a smart assistant application to implement scripted conversations has some specific characteristics that have to be considered along the different software development phases.

2.1 | Conversational agent design guidelines

As pointed out above, tools available to create new applications are constrained to those offered by the original smart assistant's developers. Besides, applications have to meet some standards on availability, robustness and flexibility. As a consequence, there are some guidelines that should be adopted from the very early design stages.

First, your design must be adaptable. For this, different sentences or cues have to be defined for each intent (cf. 3). The possibility of repeating or correcting a previous interaction must also be considered to deal with incorrectly triggered actions due to speech recognition errors or to handle silence-triggered timeouts.

Another design aspect that must be addressed is personalization. In case different user profiles are foreseen, specific sentences should be identified for each profile. In the same way, interactions have to be designed taking into account existing localization information or restrictions. Note that scripted conversations may be specific to a given profile or locale, which has to be considered in your design.

Availability is another key aspect when designing a conversational agent. In other words, it should be guaranteed that application invocation is simple and can be achieved by means of a short utterance. Besides, tasks should be kept simple and specific, and lengthy interactions and unnecessary questions should be avoided.

Finally, note that a smart assistant's conversation has to be perceived as being as close as possible to an interaction with an actual person, that is, it has to be relatable. For that, conversations have to be scripted to sound clear, concise and natural, for example by introducing some variety in vocal interactions.

2.2 | Building a conversational agent

Applications for commercial-off-the-shelf smart assistants are implemented as cloud-based remote services. Therefore, building a conversational agent implementing a specific voice interaction model requires an Internet-accessible endpoint for hosting that cloud-based service.

As a general approach, conversational agents will be eventually deployed as web services or remote procedures, depending on the vendor. Smart speakers would invoke a specific application triggered by a specific voice command from a collection of equivalent voice commands. Off-the-shelf smart assistants react to particular voice keywords (e.g., *Hey Google* in the case of Google Home or *Alexa* in the case of Amazon Alexa) followed by an application invocation command. Only when triggered, data is sent to the cloud to be processed by the application's logic (cf. Figure 1).

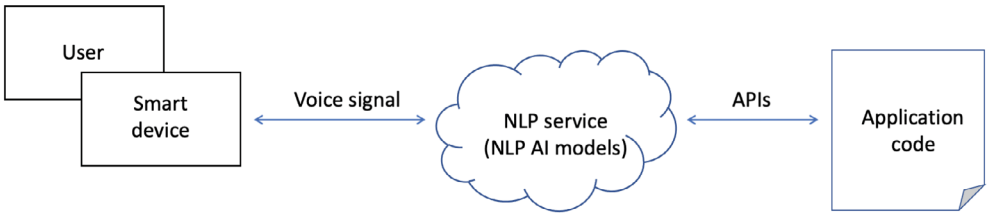

FIGURE 1 Conversational agent's deploy model for off-the-shelf smart assistants. The smart device/smart assistant deals exclusively with voice signals. Application code interacts exclusively with natural language processing (NLP) cloud services by means of character strings.

As a simpler option, Google and Amazon provide Google Assistant or Alexa-hosted options to build, store and host smart speaker applications and their resources in their respective cloud environments. This approach dramatically simplifies application development, particularly in the case of simple or popular interaction models, which can be accessed as pre-built modules in their respective development environments.

In general, these applications are event-driven and follow a synchronous invocation-response model. In other words, smart assistant applications can be seen as a collection of event handlers, each of them processing a specific user's invocation, represented as a text string, to generate an appropriate response, also as a text string.

Smart speaker applications can be developed in a wide variety of software engineering environments and programming languages. For example, interaction models may be implemented in Node.js, Java, Python, C# or Go in the case of Alexa, and Node.js, Go, C++ or Java in the case of Google Assistant, among others.

Note that smart assistants and other conversational agent-enabled solutions just collect user's voice utterances to be submitted to cloud-based natural language processing (NLP) services and play back voice signals generated by such services. NLP services are basically artificial intelligence (AI) models that are continuously learning to improve their efficacy. Application code receives just the plain text generated by speech recognition modules in the NLP service and has no access to the original audio signal. In the same way, application code may generate text streams as a response to events, which in turn would be converted to speech signals by the voice synthesis modules in the NLP service.

3 | RESULTS: STEP-BY-STEP IMPLEMENTATION OF A SCRIPTED CONVERSATION

The next paragraphs describe the most relevant details and the challenges addressed related to the implementation of scripted conversations. This discussion is illustrated with an actual case study, namely the implementation of T-MoCA as an Alexa skill, that is, as an application to be invoked by means of an Amazon smart speaker or any device implementing the Alexa smart assistant interface. Thus, the main requirement that the devised implementation has to satisfy is that it must keep the T-MoCA's discrimination capabilities. In other words, the scripted conversation implemented should be indistinguishable from the original telephone test insofar its ability to detect mild detect cognitive impairment is concerned. As pointed out above, the Alexa ecosystem was selected because of its popularity, its availability in a broad range of home appliances and computing equipment beyond Amazon devices, and the comprehensive collection of software development, testing and deployment resources provided by Amazon.

First, the skill's invocation name has to be defined. This is the word or phrase that the user needs to utter in order for Alexa to launch the skill implementing T-MoCA. It must be defined as a string of lowercase characters and it cannot contain any of the Alexa Skill's standard launch phrases (e.g., *launch*, *open*, *load*, *begin*, etc.), any of the predefined wake-up words (such as *Alexa*, *Amazon*, etc.) or prepositions. In our case, the selected invocation name was *digimoca*. Thus, an example invocation would be:

"*Alexa, open digimoca*"

The next step consists of the identification of all relevant intents and their mapping to spoken utterances. An intent represents a behavior that fulfills a user's voiced request. Behaviors may include, for example, interacting with other software modules, updating local or remote data elements or synthesizing a voiced response. Thus, in this phase, a set of

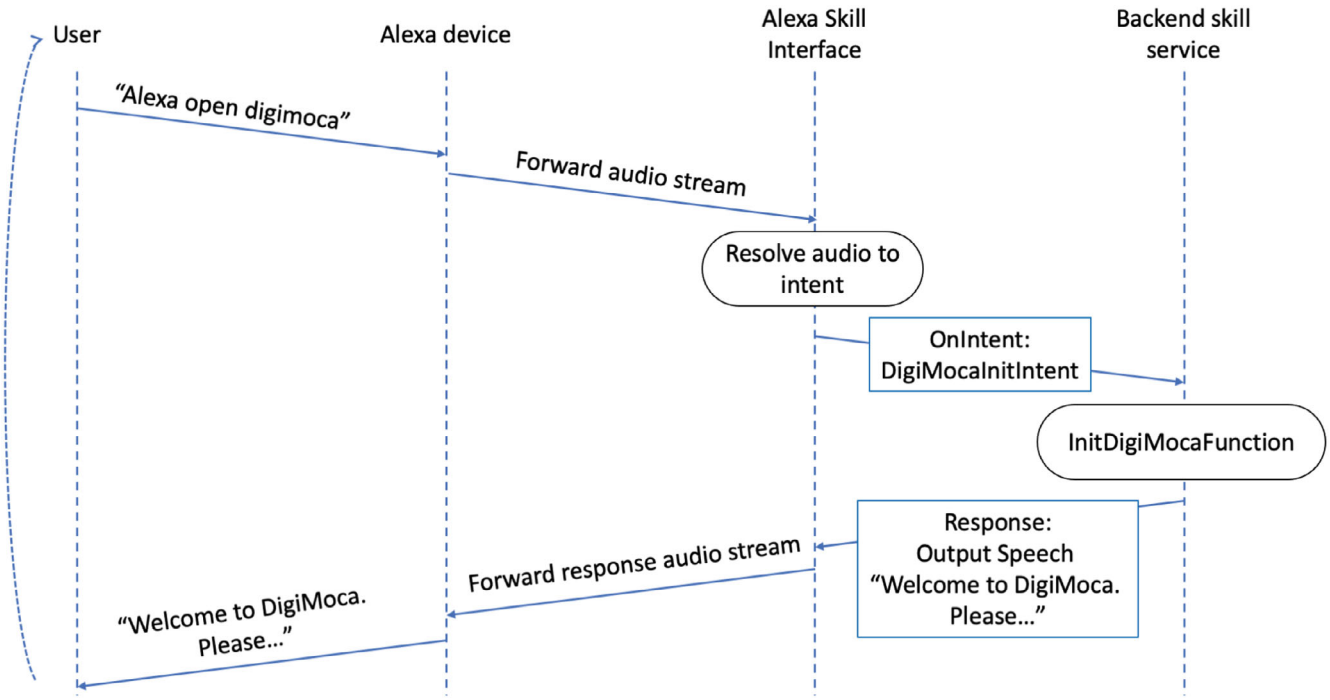

FIGURE 2 Interactive workflow of the DigiMoca Alexa skill.

sample utterances are specified that are mapped to specific intents. As depicted in Figure 2, every time the user produces one of the defined utterances, the spoken words are resolved to a specific intent, which in turn is implemented and handled by the Alexa cloud service.

Besides the intents specific to an application, all Alexa skills are equipped with four basic intents, namely `Cancel`, `Help`, `Stop`, and `NavigateHome`. The first three provide usage directions and support for quitting a skill, while `NavigateHome` is only used when developing complex skills that involve multiple steps. Developers may also add built-in (i.e., predefined) as well as custom intents.

To determine which intents are necessary to implement a scripted conversation, all the elements that serve as cues to trigger user responses have to be identified. In our particular case, all items in T-MoCA had to be mapped to the corresponding scripted conversation elements. For each question or sentence in the scripted conversation that the conversational agent will produce, all the likely responses from the user should be identified. These will correspond to the sample utterances discussed above to be handled by the back-end cloud service. For example, for each item in T-MoCA, all possible responses from the subject being tested for cognitive impairment should be identified in advance.

Once all utterances are identified, they are classified according to their meaning. Then, an intent is defined for each group of equivalent utterances from the application's perspective, T-MoCA in our case. For this, built-in intents may be exploited (e.g., `Yes`, `No`, `Repeat`, `Next`, etc.) and custom ones will be created from scratch when required. When implementing scripted conversations, the `FallBackIntent` built-in intent is especially relevant as it will address any unmapped utterances, that is, those occasions when the smart assistant will not understand the user.

Intents may include optional arguments, known as slots, to store additional information such as variable data. Slots are typed, that is, they may take a value from a finite set of values. In the same way as intents, both built-in and custom types may be used. Intents are collected in a JavaScript Object Notation (JSON) open data interchange format file.

Once intents are defined, the actual implementation of the skill is addressed. In a nutshell, a skill is an event-driven piece of code that handles each of the intents in the application. The Alexa Skills Kit, a software development framework that supports the implementation of skills in Node.js, Java, and Python, is used to construct Alexa skills. In this discussion, Python was selected as the programming language.

```python
import ask_sdk_core...      # Import Alexa Skills Kit's SDK classes
import ask_sdk_model...
import ...                  # Other necessary imports


class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch. When the Skill is firstly launched,
       the request type created is of LaunchRequest, and this is the
       handler for it. All implemented skills must have it."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        return handler_input.response_builder.speak("Skill launched!")

# Create other implemented Handler classes, inheriting from AbstractRequestHandler.
# Typically, each one of them handles one single possible response from the user.


# Create a custom Skill Builder object and add the handlers
sb = CustomSkillBuilder()

# Add all request handlers created
sb.add_request_handler(LaunchRequestHandler())

# The default name in the Lambda console is lambda_function.lambda_handler
lambda_handler = sb.lambda_handler()
```

FIGURE 3 Excerpt of the `lambda_function.py` collecting the intent handlers' logic.

The skill code will be hosted by the Amazon cloud and run by the AWS Lambda service, Amazon's serverless cloud computing service. The concept of serverless computing[17] refers to a cloud computing model that does not require maintaining specific servers to run a service (i.e., the DigiMoca skill in our example). AWS Lambda is a fully managed service that takes care of all the infrastructure needs.

Essentially, skill implementation proceeds in three steps:

1. Create a `lambda_function.py` file to collect all the handlers eventually run by the AWS Lambda service for each intent. This code has always the same structure (cf. Figure 3).
2. As pointed out above, skills are event-driven applications. In a nutshell, skills implement mandatory `can_handle()` and `handle()` methods. These methods receive as input data a JSON-encoded Python dictionary containing all the information from the user's request, as depicted in Figure 4. The `can_handle()` method is initially invoked for each handler defined, until one of them returns `True`. The `test_state` attribute, which is stored as a session attribute, provides the state information elements `item` and `step` determining the exact point inside a scripted conversation and therefore whether such handler is the appropriate one at this time. As for the `handle()` method, its purpose is to actually handle the user response, which is also stored in the input object along with the slots (i.e., relevant data elements) inside it. Finally, conversation advances to the next stage by invoking `NextIntentHandler`'s, `handle()` method to update the `item` and `step` values and send back a new cue (i.e., the next line in the scripted conversation) to the user. Figure 5 illustrates the user state elements in the case of T-MoCA.
3. Create a `CustomSkillBuilder` object to collect all the created intent and exception handler classes. Then, the skill will run as a `lambda_handler` object created by invoking the homonym method in the `CustomSkillBuilder` object.

```python
class MesIntentHandler(AbstractRequestHandler):
    """Handler for Mes Intent"""
    def can_handle(self, handler_input):
        state = handler_input.attributes_manager.session_attributes['test_state']
        full_state = str(state['item']) + str(state['step'])
        return (ask_utils.is_intent_name("MesIntent")(handler_input) and full_state in ['63'])

    def handle(self, handler_input):

        logger.info("In MesIntentHandler")
        state = handler_input.attributes_manager.session_attributes['test_state']
        state['month'] = handler_input.request_envelope.request.intent.slots['mes'].value
        return NextIntentHandler().handle(handler_input)
```

FIGURE 4 Excerpt of handlers' code for the *Mes* (month) intent. In this example, the user responds with today's month when asked. The user's response is appended to the state object, which contains all the responses to the test so far (cf. Figure 5).

4 | EVALUATION

In order to evaluate the performance of the DigiMoca implementation, the PARADISE (PARAdigm for DIalogue System Evaluation) framework[18] was utilized. PARADISE establishes two performance metrics, namely dialog cost, which should be minimized, and task completion or success, which should be maximized. The main contributions to dialog cost are the amount of dialog actions, particularly the number of turns required to complete it, and the time that the user takes to complete a conversation. In our specific case, however, dialog cost evaluation would not be applicable, since DigiMoCA was designed to perform cognitive assessment by means of a conversation or dialog that is intentionally conceived as a cognitive demanding task. Therefore the resulting measure would be more related to the user's level of cognitive impairment than anything else.

Task completion, on the other hand, measures how well the speaker is able to understand the user and it is extremely important for this application, since misunderstandings may lead to a contamination of the overall cognitive test result. Thus, according to the PARADISE framework, we used the Kappa coefficient in order to measure task completion.[19,20] The Kappa coefficient κ is computed from a confusion matrix, which shows the number of understandings and misunderstandings between user and speaker, for each possible scenario.

Table 2 depicts the DigiMoca's confusion matrix, created from 42 sessions with 21 different participants who completed the DigiMoCA test. Each row represents the expressed intent from the user (regardless of the particular utterance) and each column represents the detected intent by DigiMoCA. The first 7 intents (namely, "Cancel," "Stop," "Fallback," "Repeat," "Yes," "No," and "Next") are default built-in intents, which nevertheless are applicable to our purpose as they are relevant in the original T-MoCa test. Intents 8 to 18 (i.e., from the "Remember" intent onward) are DigiMoca-specific intents. The values on the main diagonal (bold values) represent the occasions when the skill understood the correct intent from the user, whereas numbers outside the main diagonal represent misunderstandings or mismatches between the user intent and the intent triggered by the speaker. We should stress the importance of the "Fallback" intent, which represents anything that the user says that does not match any of the other intents (e.g., a nonsensical sentence). As we can see, the most common types of misunderstanding occur when (1) the user says something that does not correspond to any intent (but the skill thinks otherwise) or (2) the user correctly utters an intent (e.g., says a number) but the "Fallback" intent is triggered instead. Type (1) misunderstandings fall correspond to the values on the third row, whereas type (2) misunderstandings appear on the third column.

Given the confusion matrix in Table 2, the Kappa coefficient measures the success of DigiMoCA at obtaining the information necessary to complete the whole cognitive task. It is computed as follows:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)},$$

where $P(A)$ is the percentage of times that the speaker understood the user correctly (i.e., triggered intents match expressed intents), and $P(E)$ is the percentage of times that the speaker is expected to match the proper intent *by chance*. Values of κ greater than 0.75 may be taken to represent excellent agreement beyond chance, values below 0.40 may be taken

```
class State:
 def __init__(self, item = ENDED, step = 0):
    self.item = item                # Item of the test in the current State
    self.step = step                # Step of the current test item
    self.forward_numbers = 0        # True if list of numbers recalled
    self.backward_numbers = 0       # True if reverse list of numbers recalled
    self.f_words = []               # List of valid words starting with F
    self.f_start_time = None        # Timestamp F words started (1 min max.)
    self.letters_last_time = None   # Timestamp of last letter uttered (reaction time)
    self.score = 0                  # Overall score of the current testing session
    self.calculations = []          # List of 7-subtractions starting from 100
    self.letter_mistakes = 29       # Number of letter mistakes
    self.first_sentence = 0         # 1 if user remembered first sentence perfectly
    self.second_sentence = 0        # 1 if user remembered second sentence perfectly
    self.transport = 0              # 1 if correct category for train and bike
    self.measure = 0                # 1 if correct category for ruler and clock
    self.unclued_recalls = []       # Words remembered without category clue
    self.clued_recalls = []         # Words remembered with category clue
    self.choice_recalls = []        # Words remembered with multiple choice clue
    self.day = 0                    # Day of the month stated by user
    self.month = ""                 # Current month stated by user
    self.week = ""                  # Day of the week stated by user
    self.year = 0                   # Current year stated by user
    self.afaga = 0                  # True if aware of premises
    self.vigo = 0                   # True if aware of being in Vigo
    self.date = ""                  # Test administration date
    self.study_years = 0            # Number of years of study
    self.user_id = 0                # User ID number

 def set(self, item = ENDED, step = 0):
    self.item = item
    self.step = step
```

FIGURE 5 State variables in the case of T-MoCA. The state object collects all the information relevant to score the test, to sequence its elements and to provide inter-session persistence.

to represent poor agreement beyond chance, and values between 0.40 and 0.75 may be taken to represent fair to good agreement beyond chance.[21] The fact that κ takes into account the complexity of the task by introducing the agreement beyond chance makes it a suitable metric for task completion.[18] In this case, the expected chance of agreement $P(E)$ is unknown, but can be estimated using the following formula:

$$P(E) = \sum_{i=1}^{n} \left(\frac{t_i}{T} \right)^2,$$

where t_i is the sum of elements in column i of M, and T is the sum of all the elements in M ($T = \sum_{i=1}^{n} t_i$). On the other hand, $P(A)$ can be directly computed as follows:

$$P(A) = \frac{\sum_{i=1}^{n} M_{i,i}}{T},$$

where $M_{i,i}$ is the ith element on the main diagonal.

Thus, DigiMoCA's task success values obtained according to the PARADISE framework are $P(E) = 0.1676$, $P(A) = 0.9176$, which corresponds to a Kappa coefficient κ of 0.9010. Therefore, DigiMoca understands the user correctly 91.76% of times with an excellent agreement beyond chance. According to the nomenclature proposed by Landis and Koch[22] for the description of strength of agreement based on Kappa statistics, we can label DigiMoCA as "Almost Perfect" in terms

TABLE 2 Confusion matrix: Expressed intent versus triggered intent.

Intent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Cancel	**5**																	
Stop		**18**																
Fallback			**272**							3	4							25
Repeat				**3**														
Yes			52		**795**													
No			45			**854**												
Next			7				**63**											
Remember			33					**307**										
Numbers			20						**158**								7	
FirstSent										**36**								
SecondSent											**34**							
Transport			2									**40**						
Measure			5										**37**					
Week			3											**39**				
Month			2												**40**			
Place		5														**35**		
DigitList																	**76**	
WordList			27															**163**

of task completion. This is a very positive result, but also a expected one considering that DigiMoCA's implementation relies on Alexa's speech recognition services.

5 | DISCUSSION ON THE MAIN ISSUES AND CHALLENGES ADDRESSED

In the following paragraphs we discuss the issues identified when implementing a scripted conversation according to the model in Section 3. In general, these issues are not specific to our proof-of-concept T-MoCA implementation, but can be extrapolated to other scripted conversation-based situations. Table 3 summarizes the challenges addressed.

As pointed out above, the dialog between user and smart assistant is triggered by a user's utterance, which the agent's artificial intelligence NLP model maps to an intent. Then, the intent generates a response that is voiced by the smart device, which in turn triggers a utterance from the user that triggers a new intent. This process is iterated until the conversation is abandoned (e.g., by invoking one of the `Cancel/Exit` intents) or completed.

TABLE 3 Conversation-related challenges addressed.

Challenge	Solution
Keeping track of user responses	Persistent user state information management (cf. Figure 5)
Dealing with limited attention span	Measure response time to timed cues
Lack of detailed timing information	Statistical estimation of roundtrip times
Speech rate adaptation	Prosodic annotations
Users' verbal fluency	Frequency analysis-based word lists
Orientation, location awareness	Location services, system's datetime services

Keeping track of user's responses and user's state is a key element when implementing a scripted conversation in general, and T-MoCA in particular. In our case, in order to compute the final score of the test, we need to save the responses to each test item. From a more general perspective, the sequence of user responses serves as an annotated log of the complete conversation, which may be utilized for quality assessment, as evidence in performance evaluation processes, as a profiling mechanism, to route a request for service and so forth. This can be done both intra-session and spanning different sessions, utilizing session and persistent attributes respectively. For this, a `state` object is created as discussed above to encapsulate all attributes containing the information about user responses. Then, an `attributes_manager` object is used to manage its values (cf. Figure 4).

Typically, a session ends when the smart assistant does not provide a cue to the user (i.e., when the underlying scripted conversation reaches one of the defined final sentences) or when the user fails to provide a response to the smart assistant in time (i.e., before a predefined timeout, which may be specific to each cue, is triggered). In the latter case, persistence is required to recover an ongoing conversation (i.e., a testing session in our particular case) that was abruptly terminated. Amazon skills rely on DynamoDB[23] for persistent storage. In our context, this is utilized to store the `state` object in the form of a JSON-like dictionary. When a session expires, attributes are stored in the persistent layer. Accordingly, every time a new conversation starts, the previous session's attributes are retrieved from the persistence layer. If no persistent attributes exist (e.g., when a scripted conversation is initiated anew), new default values are created.

Attention span is a relevant indicator of cognitive issues and measuring attention is another central element of cognitive tests. From a wider perspective, attention span is also relevant in the case of long and tiring conversations or distracted or unmotivated users. For example, one of the T-MoCA items requires the user to tap their hand on the table at specific moments, which in turn is used to estimate their ability to focus attention. However, this is not possible when interacting with a smart assistant, as any sound other than speech is not transcribed and therefore not available to the cloud AI model. After consulting with a psychologist, we implemented a workaround that serves the same purpose. Instead of tapping on the table, the user has to immediately respond "yes" or "no" to a specific voiced clue uttered by the smart assistant. This is reflected in the script as prosodic annotations. In general, attention-gathering cues can be defined to be uttered at specific times along the scripted conversation. Then, lack of response or response time are considered to estimate attention span.

Incidentally, another limitation from off-the-shelf voice assistants with a direct impact in software development is the lack of detailed timing information. For example, access to the actual reaction times is not provided. In order to adequately estimate reaction time (e.g., when estimating attention span), the overall roundtrip time (e.g., time between the smart agent sending an utterance and a response being received from the cloud AI model) has to be estimated. For this, the average value of minimum round trip times for all response time-monitored interactions is subtracted to obtain an estimation of users' reaction time. In our case, scores are computed taking as a reference the established T-MoCA threshold for table-tapping reactions, namely one second.

An additional common issue when modeling scripted conversation is the decrease in speech comprehension with increasing speech rates. This is especially relevant in the case of users with limited fluency in the assistant's language or in the case of senior users. In our specific case, typical users are senior adults being tested to confirm or discard cognitive impairment. A most relevant trait of this user profile is the decline of speech comprehension with increasing speech rates, as a consequence of a slowing of brain function with age.[24] As a consequence, speech adaptation is required to avoid this confounding factor, as poorer performance may be a consequence of auditory decline rather than cognitive impairment. To address speech rate adaptation, prosodic annotations are introduced in order to adapt the conversational agent's speech cadence. In our case, speech rates are adapted to that typical of an elderly person (cf. Figure 6).

Interacting users' verbal fluency is another relevant aspect to consider when implementing scripted conversations. As the smart assistant is triggered by specific verbal cues (i.e., the ones mapped to intents), its selection is instrumental to guarantee one of the basic design guidelines of a conversational agent in Section 2.1, namely being perceived as close as possible to an interaction with an actual person. In our case, fluency is a most relevant trait addressed by T-MoCA as it is an indicator of cognitive decline. This requires the subject saying words of a certain type and starting with a specific letter. In order to check if the word uttered by the user is a valid one in DigiMoca, a list of valid words including the most common words in the user's mother tongue is generated (approx. 2000 words, no proper nouns, numbers or verb inflections). There are essentially two ways to achieve this, namely by assigning the user's utterance to the `QueryText` property and checking it against the word list or using a new slot type with the list of valid words as possible values.

```
3  {"speak" : '''<prosody rate="slow"><p><break time="1s"/>
4    I am going to read you a sentence.
5    Repeat it after me, exactly as I say it.
6    </p></prosody>''', "reprompt" : "Do you understand?"},
7
8  {"speak" : '''<prosody rate="slow"><p>
9    I only know that John is the one to help today
10   </p></prosody>''', "reprompt" : "Please repeat that sentence exactly as I said it."},
11
12 {"speak" : '''<prosody rate="slow"><p>
13    Now I am going to read you another sentence.
14    Repeat it after me, exactly as I say it.
15    </p></prosody>''', "reprompt" : "Do you understand?"},
16
17 {"speak" : '''<prosody rate="slow"><p>
18    The cat always hid under the couch when dogs were in the room.
19    </p></prosody>''', "reprompt" : "Please repeat that sentence exactly as I said it."},
```

FIGURE 6 Example of prosodic annotations. In this case, the smart assistant is requested to utter these sentences slowly. In line 3, a 1-second timeout is defined to confirm user's awareness about the task to be carried out.

Finally, the place and time when a scripted conversation is taking place was also addressed. For this, the smart assistant's location details, together with date, month, year and day of the week values can be fetched from existing location services. To obtain time information, standard system services can be queried through standard libraries (e.g., Python's `datetime.now()`) while the assistant's location can be obtained by means of its own location services (e.g., Location Services for Alexa Skills). In the case of T-MoCA, orientation is another relevant trait to assess cognitive decline, that is, individuals under evaluation being aware of their actual location and time of the day. To assess orientation in DigiMoka, the smart assistant's time and location details are obtained as indicated when the user is queried for the time of the day and location. Then, results are compared to generate the corresponding score. Besides, this information is stored in the user's state object (cf. Figure 5).

6 | CONCLUSION

This article discusses the implementation of intelligent conversational agents utilizing off-the-shelf smart assistants. This approach is illustrated with the implementation of a standardized conversation-based test for the detection of cognitive disorders, which are typically diagnosed using pen-and-paper tests.

From a software engineering standpoint, smart assistants are basically speech-based interfaces to event-driven software applications. Cloud based NLP services convert users' speech interactions into text and back. At the back-end, event-based software applications react to specific cues or triggering sentences by performing business logic-defined activities, such as playing music, updating a calendar, launching other applications, or doing some task on behalf ft the user.

In some cases, the reaction to triggering sentences may include the synthesis of a voiced response to be played back to the user. In turn, this sentence may inspire a voiced response from the user, which may serve to trigger a new event at the back-end application. This process may be iterated to model a scripted conversation or dialog.

The event handling logic at the back-end may be as simple as registering the (text version of) user's voiced intervention, but it may also involve complex decision-making to guide the scripted conversation to evolve in different ways. Using back-end storage, information about previous interactions may be utilized to drive the conversation along different paths and also to provide persistence to allow interrupted conversations to be resumed.

On the other hand, from a neuropsychological point of view, this work lays the foundations for digitizing and adapting to a conversational agent classical pencil and paper tests affected by confounding variables. In particular, challenges such as keeping track of user responses; speech rate adaptation to asses attentional span by prosodic annotations, or users'

verbal fluency, among others, were addressed. Thus, this adaptation represents a promising step in the development of easy-to-use and easy-to-administer smart digital solutions for the elderly.

Besides cognitive assessment, the use of questionnaires as an evaluation, screening or diagnosis tool is common practice in other fields like neuropsychiatry, social sciences, or psychology. As a consequence, a successful approach to the introduction of conversational agents in these fields may suppose a relevant advance in a broad range of scenarios. As demonstrated in this article, the state of the art is mature enough to support the development of complex conversational agents at a reasonable cost.

ACKNOWLEDGMENTS

This research was funded by the Spanish Ministry of Economy, Industry and Competitiveness Grant PID2020-115137RB-I00: Servicios y Aplicaciones para un Envejecimiento Saludable (SAPIENS), and by the Ministry of Science, Innovation and Universities under the Grant reference FPU19/01981 (Formación de Profesorado Universitario). University of Vigo/CISUG: openaccesscharges.

AUTHOR CONTRIBUTIONS

Moisés Pacheco-Lorenzo: Conceptualization; methodology; investigation; original draft preparation. **Manuel J. Fernández-Iglesias**: original draft preparation; review and editing. **Sonia Valladares-Rodriguez**: investigation; PARADISE validation. **Luis E. Anido-Rifón**: supervision; resources; funding and acquisition; project administration. All authors read and agreed to the published version of the manuscript.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Manuel J. Fernández-Iglesias https://orcid.org/0000-0003-4462-8724
Sonia Valladares-Rodriguez https://orcid.org/0000-0003-1195-9949

REFERENCES

1. Bentley F, Luvogt C, Silverman M, Wirasinghe R, White B, Lottridge D. Understanding the long-term use of smart speaker assistants. *Proc ACM Interact Mob Wearable Ubiquitous Technol*. 2018;2(3):1-24. doi:10.1145/3264901
2. Kexel CA, Osterloh J, Hanel C. Smart home – A new marketing era. In: Thorhauer Y, Kexel CA, eds. *Facetten der Digitalisierung: Chancen und Herausforderungen für Mensch und Management*. Springer Fachmedien; 2020:31-53.
3. Loukis E, Janssen M, Mintchev I. Determinants of software-as-a-service benefits and impact on firm performance. *Decis Support Syst*. 2019;117:38-47. doi:10.1016/j.dss.2018.12.005
4. Jimenez C, Saavedra E, de Campo G, Santamaria A. Alexa-based voice assistant for smart home applications. *IEEE Potent*. 2021;40:31-38. doi:10.1109/MPOT.2020.3002526
5. Noda K. Google home: smart speaker as environmental control unit. *Disab Rehabilit Assist Technol*. 2018;13(7):674-675. doi:10.1080/17483107.2017.1369589
6. Wyatt D, Choudhury T, Bilmes J. Conversation detection and speaker segmentation in privacy sensitive situated speech data. In: van Hamme H, van Son R, eds. *Proceedings of Interspeech 2007*. ISCA; 2007:586-589.
7. Adamopoulou E, Moussiades L. An overview of chatbot technology. In: Maglogiannis I, Iliadis L, Pimenidis E, eds. *Artificial Intelligence Applications and Innovations*. IFIP Advances in Information and Communication Technology WG 12.5. Springer; 2020:373-383.
8. Albayrak N, Özdemir A, Zeydan E. An overview of artificial intelligence based chatbots and an example chatbot application. Proceedings of the 26th Signal Processing and Communications Applications Conference (SIU); 2018:1130-1134; IEEE, Curran Associates Proceedings.
9. Szymanski MH, Wall P, Watts-Englert J. Creating interactional alignment in call center customer care. In: de Waal Malefyt T, McCabe M, eds. *Women, Consumption and Paradox*. Routledge; 2020:78-102.
10. Sheehan B, Jin HS, Gottlieb U. Customer service chatbots: anthropomorphism and adoption. *J Bus Res*. 2020;115:14-24. doi:10.1016/j.jbusres.2020.04.030
11. Choi Y, Monserrat TJKP, Park J, Shin H, Lee N, Kim J. ProtoChat: supporting the conversation design process with crowd feedback. *Proc ACM Human-Comput Interact*. 2021;4(CSCW3). doi:10.1145/3432924
12. Daubman BR, Bernacki R, Stoltenberg M, Wilson E, Jacobsen J. Best practices for teaching clinicians to use a serious illness conversation guide. *Palliat Med Rep*. 2020;1(1):135-142. doi:10.1089/pmr.2020.0066
13. Grosberg D, Charlop MH. Teaching conversational speech to children with autism spectrum disorder using text-message prompting. *J Appl Behav Anal*. 2017;50(4):789-804. doi:10.1002/jaba.403

14. Pacheco-Lorenzo MR, Valladares-Rodríguez SM, Anido-Rifón LE, Fernóndez-Iglesias MJ. Smart conversational agents for the detection of neuropsychiatric disorders: a systematic review. *J Biomed Inform*. 2021;113:103632. doi:10.1016/j.jbi.2020.103632
15. Katz MJ, Wang C, Nester CO, et al. T-MoCA: a valid phone screen for cognitive impairment in diverse community samples. *Alzheimer's Dement Diagnos Assessment Disease Monitor*. 2021;13(1):e12144. doi:10.1002/dad2.12144
16. Pinto TCC, Machado L, Bulgacov TM, et al. Is the montreal cognitive assessment (MoCA) screening superior to the mini-mental state examination (MMSE) in the detection of mild cognitive impairment (MCI) and Alzheimer's disease (AD) in the elderly? *Int Psychog*. 2019;31(4):491-504. doi:10.1017/S1041610218001370
17. Baldini I, Castro P, Chang K, et al. Serverless computing: current trends and open problems. In: Chaudhary S, Somani G, Buyya R, eds. *Research Advances in Cloud Computing*. Springer; 2017:1-20.
18. Walker MA, Litman DJ, Kamm CA, Abella A. PARADISE: a framework for evaluating spoken dialogue agents. arXiv preprint cmp-lg/9704004; 1997.
19. Siegel S, Castellan N. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill International Editions Statistics Series. McGraw-Hill; 1988.
20. Rehman UU, Chang D, Jung Y, Akhtar U, Razzaq M, Lee S. Medical instructed real-time assistant for patient with Glaucoma and Diabetic conditions. *Appl Sci*. 2020;10:2216. doi:10.3390/app10072216
21. Fleiss JL, Levin B, Paik MC. *Statistical Methods for Rates and Proportions*. Wiley; 2003.
22. Landis JR, Koch GG. The measurement of observer agreement for categorical data. *Biometrics*. 1977;33(1):159-174.
23. Sivasubramanian S. Amazon DynamoDB: a seamlessly scalable non-relational database service. In: Candan KS, Chen Y, Fuxman A, Gravano L, eds. *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. Association of Computer Machinery; 2012:729-730.
24. Schneider BA, Daneman M, Murphy DR. Speech comprehension difficulties in older adults: cognitive slowing or age-related changes in hearing? *Psychol Aging*. 2005;20:261-271. doi:10.1037/0882-7974.20.2.261

How to cite this article: Pacheco-Lorenzo M, Fernández-Iglesias MJ, Valladares-Rodriguez S, Anido-Rifón LE. Implementing scripted conversations by means of smart assistants. *Softw Pract Exper*. 2022;1-13. doi: 10.1002/spe.3182