A Domain Specific Modeling Language for Model-Based Design of Voice User Interfaces

Claudia Steinberger and Christian Kop
Universität Klagenfurt, Klagenfurt, AUSTRIA
{claudia.steinberger,christian.kop}@aau.at

Abstract. Designing a voice user interface (VUI) can become more challenging than designing a graphical user interface (GUI). Without visual interaction elements, a user can be less bound to predefined interaction regulations and restrictions. Concrete user requests and system responses in a dialog strongly depend on the initial intention of the user and the user's utterances during the dialog, to which the voice-based system has to respond. The aim of this paper is to present an intention-oriented approach to the design of VUIs. This is achieved in particular by defining and applying RIML, a domain specific modeling language, which enables VUI designers to create platform-independent intention models. A RIML model includes to which requests a VUI should respond to, what intentions are involved, how the system handles user requests and responses, and what to do in case of misunderstanding or failure. Based on a RIML model, a voice-based system is able to communicate flexibly via its VUI with the user. We show this using the example of AYUDO Voice, a language assistant for personal health management

Keywords: Domain Specific Modeling Language · Intention Modeling · Active Assistance · Voice User Interface · Voice-Based System

1 Introduction

In our days voice-based system technologies are advancing rapidly. Their voice user interfaces (VUIs) allow a user to interact with a system in the form of natural language inquiries. This enables a hands-free and eyes-free manner in which users can interact with the system in an intuitive way. However, a positive user experience is necessary for VUIs to be accepted. Thus, the design of usable VUIs plays a major role in the system development process.

In contrast to graphical user interfaces (GUIs), VUIs have no visible interface. Instead of clicking buttons and selecting options from dialog boxes, users make their voice requests and respond to questions by voice [20]. Their concrete requests and answers depend on their intentions [24]. By an *intention* we understand what purpose a user wants to achieve through the interaction with a voice-based system. From the point of view of a voice-based system, which wants to fulfill certain intentions, we call them *request intentions*.

Without visual interaction elements a user can be less bound to predefined interaction regulations and restrictions. The interaction must therefore be as natural as possible, not strictly sequential and flexible, because there are many ways in which a user can articulate his or her request intention. Existing interaction-element-oriented approaches

for designing GUIs (e.g. [1][2][3]) are not well suited for designing VUIs. We believe that VUIs must be designed intention-oriented to ensure a high level of usability.

The aim of this paper is to present an intention-oriented approach to the design of VUIs. Therefore, we present RIML (Request Intention Modeling Language), a domain specific modeling language, which is dedicated to creating intention models. A RIML model includes to which requests a voice-based system should respond, what intentions are involved, how the system deals with the users' requests and responses, and what has to happen in case of misunderstanding or failure. Thus, it conceptually represents those user intentions that the VUI should support - together with their corresponding features to be able to do so. With the help of RIML, an Intention Designer can specify the expected request intentions to a future voice-based system, independent of a specific platform or technology. In this paper, we also sketch RIML-Modeler, a tool to support the creation of RIML models. Based on a RIML model, a voice-based system with a model driven architecture [18] is able to communicate via its VUI with the user flexibly, to request required inputs from the user, to interpret answers, to recognize and to feedback voice inputs that are not understandable or incorrect. As a case study throughout the paper, we use the VUI that we are working on in the AYUDO project [19].

The paper is further structured as follows: Chapter 2 deals with challenges in designing VUIs and presents a simplified voice-based system architecture model. Chapter 3 sketches AYUDO, a voice-based system for personal health management, used as a case study in this paper. Chapter 4 presents our approach in the MOF metamodeling hierarchy and introduces the metamodel of RIML. Then it exemplifies an excerpt of the AYUDO RIML model as a use case. Chapter 5 summarizes the results of the paper and provides an outlook to further research work.

2 Challenges of Modeling Voice User Interfaces Systems

One of the main reasons VUIs are so fascinating is because verbal conversation is a natural form of communication for people [20]. Thus, VUIs can particularly reduce barriers for the elderlies or for impaired people. As a result, there exists a trend from systems where the screen is displayed first to voice-based systems [6][25].
The architecture of a voice-based system [21] contains several modules as presented in *Figure 1*: a *speech recognition module* interprets an oral utterance of the user and transforms it into a textual representation (STT - Speech to Text), which is handed over to the *interaction management*. There, first the *intention recognition module* analyzes this text to find a match with an appropriate request intention. Once the intention is recognized, the *dialog management module* together with the *action execution module* checks the actual context memory. The context memory contains the information the user has already communicated to the system. The intention model represents the knowledge that the interaction management has about the intentions of its users. If more information is needed from the user to fulfill the request intention, the system prompts the user to communicate it in a loop. The action execution module hands over the prompt to the *response generation module*, which again verbalizes it into a natural language

result-prompt. This text is then handed over to the *speech synthesis module*, which produces an acoustic output to the user. Finally, if all information to fulfill a request intention is available, the intended command and its associated parameters are sent via the API to a *web service* endpoint for processing and the result is communicated again to the user.

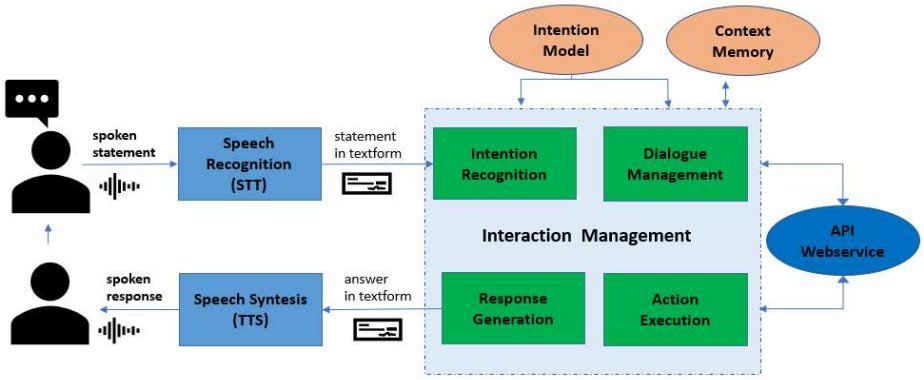

Fig. 1. Simplified architecture model of a voice-based system

While GUIs are tied to the screen and the keyboard of a device, like a desktop, a tablet or a smartphone and their visible interaction elements, voice however, is ubiquitous. VUI is surface independent and not tied to visible interaction elements. This requires a new design approach, that considers the situation and context in which the user is at that moment. From the perspective of a designer, there are subtle but strong differences between VUIs and GUIs. VUIs face the following four challenges [4][5][6]:

1. **Users do not know what they can or cannot ask the system:** A VUI user does not want to learn a load of different commands, s/he just wants to say whatever naturally comes to mind. VUIs need to understand different ways users might say the same thing. Language is diverse, complex and nuanced. When designing a successful voice experience, it is important to consider the many variations that we humans use to say the same thing. This is completely different to how one would handle this on a GUI.
2. **Once the machine says a response, it stays only in the short-term memory of the user and has no persistence:** For users, VUIs exist only in their minds. Hence, users need to concentrate more to listen to what the VUI is saying. Therefore, a voice experience must be designed to reduce cognitive load as much as possible [8]. VUIs are also not always better suited than GUIs: It is much quicker to use voice to ask a question, or input information, than it is to type a request on a keyboard while GUIs are more efficient for outputting information. Sometimes it makes perfect sense to combine both types.
3. **Unnecessary information in VUIs is much costlier, in terms of cognitive load, than it is in GUIs:** Designers have to be much more reckless in dealing with information they exclude from interaction. The dialogue with a voice-based system should please the user instead of frustrating him.

4. **VUIs enable a flat navigation:** An important difference designing VUIs is the need to rethink navigation and information architecture. In GUIs, users follow a click path to get through the individual screens. The voice enables "flat navigation" where users can go directly to where they want to go. This allows the user to find things quicker, providing a more efficient experience than with a GUI.

But how to design for this? When designing for GUIs, you often begin by mapping out the logical flow of the pages and steps a user can go through based on interactive interaction elements (i.e., a graph-based UI model [1][2][3]). Consequently, existing GUI modeling approaches are not well suited to model VUIs. Designing for voice is different from designing for web and mobile [20]. Voice lets users get right down to what they want. So, designers must abstract and think about the interactions and the user intentions as a whole.

Commercial voice technology platforms use their own cloud services for the interaction management of their VUIs. Amazon Alexa skills use *skill interaction models* that define the intents a skill can handle and the utterances that users should say to invoke them. For the design of Alexa skills, Amazon recommends to use a frame-based UI model [5], which helps to manage a dialogue in a way that users 'jump around' to get the information they need. However, this approach is integrated into the Alexa Skills Kit [9]. Alexa's skill interaction models are completely platform dependent and their voice services as well as their intention recognition services are available only via the Amazon Cloud. Thus, Alexa's compliance with data privacy is controversial.

The goal of this paper is to overcome this platform dependent approaches. We are going to present a domain-specific modeling language that enables the design of platform independent intention models, which can be used after a transformation for both, commercial platforms and private-by-design VUIs.

3 AYUDO Use Case

As a use case for our approach, we refer to a voice-based system we are currently working on. This chapter therefore introduces the AYUDO project as a use case and presents the particular challenges we faced when designing a VUI.

3.1 AYUDO Project

AYUDO aims to develop an active assisted living (AAL) system to help elderly or chronically ill people to improve their personal health and well-being and to support them in remaining independent in their familiar environment as long as possible. The AYUDO project is financed by the FFG[1] and runs from 2019 to 2022 [19]. AYUDO supports the user in documenting and monitoring his or her own state of health. *Figure 2* sketches the overall architecture of the AYUDO system: it consists of *AYUDO Voice*, a mobile application with graphical display functionality that the user can operate

[1] FFG: Österreichische Forschungsförderungsgesellschaft (https://www.ffg.at)

mainly by voice, *AYUDO Admin*, a GUI to set up user preferences and to configure the system, the AYUDO *Core System* (MCA architecture) and the *Integration Interface* to couple context-based middleware systems and external health records (ELGA[2]) to AYUDO. Altogether, the AYUDO system aims to motivate the target group to behave health consciously based on the captured and documented context data.

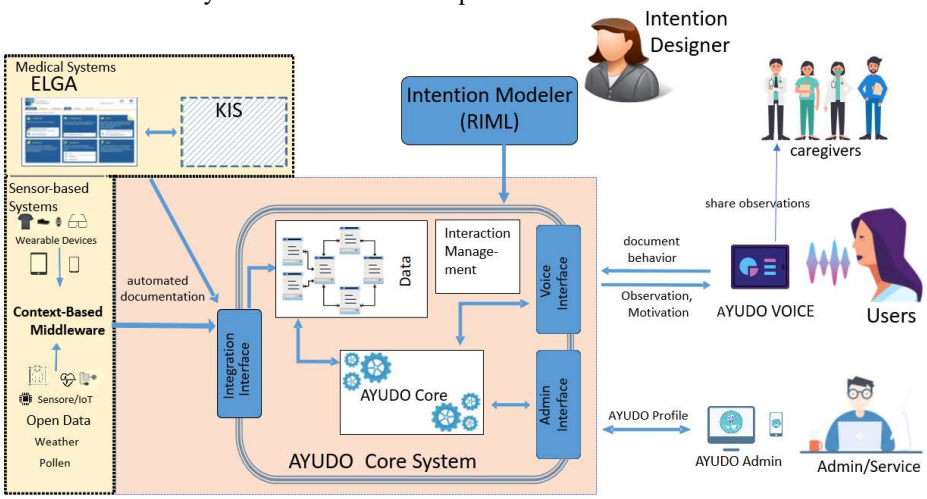

Fig. 2. AYUDO architecture

In the following, we focus on AYUDO Voice. The design of AYUDO Voice is used in a use case in Chapter 4. For example, users should be able to document measured vital parameters, their medication, their nutritional behavior and subjective vital parameters such as their daily mood on a voice-based basis. Analyses of the documented data and motivations for health-conscious behavior are also carried out verbally in dialogue form. An example of such a verbal interaction is: "*AYUDO, I have measured my blood sugar now"* with the intention to automatically document the measured value with all the data that goes with it. AYUDO Voice then starts a natural dialogue to fulfill the user's wishes, considering the challenges described in Chapter 2.

3.2 Challenges of AYUDO Voice Design

Documentation and monitoring of health data in this domain are itself challenging, since elderlies suffer from age-related limitations [10]. For instance, elderly people have issues in hearing, their visual abilities decrease, their touch perception diminishes and it becomes hard for elderlies to follow and process complex information due to issues with memory. In addition, this target group does not belong to the group of so-called "digital natives". So, a good mix of interaction possibilities with the AYUDO system is required so that these persons can easily document and retrieve their personal health data. Since elderly people also suffer under visual impairments, we also try to

[2] ELGA: Elektronische Gesundheitsakte (https://www.gesundheit.gv.at/elga/inhalt)

focus on this target group. However, this focus also has its limits. If the visual impairment is too progressed and for the target group of blind users, voice user interfaces can raise further challenges according to [14], since these users have additional requirements.

Health data is also very sensitive data with respect to Article 9 of GDPR [11]. Therefore, it has to be ensured that processing the data is based on informed and explicit consent, and the transmission is secured (authenticated and encrypted). Furthermore, both the natural language services and the AYUDO knowledge base should not be located somewhere in a cloud, but on the user clients locally or on a separate project server. Our initial analysis has shown that most providers of voice-based systems use a cloud solution where the user has little control over where the data is stored [23]. For that reason, big commercial players like Amazon Alexa or Google Assistant could not be considered as platforms for *AYUDO Voice*. We even had to discard our first choice "SNIPS", a software solution powering private-by-design voice assistants[3]. The Snips Console is no longer available due to acquisition by SONOS.

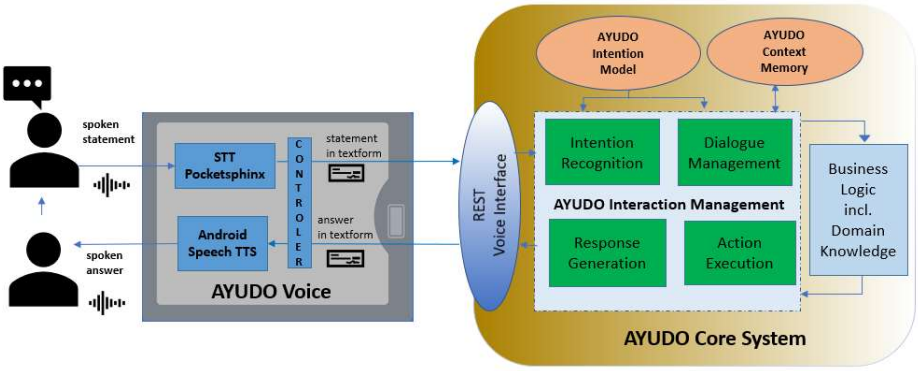

Fig. 3. AYUDO Voice Architecture

We therefore decided to develop our own private-by-design solution based on the architecture shown in *Figure 3*. To keep the user's privacy according to his or her utterances to *AYUDO Voice*, we keep the STT and TTS-synthesis locally on the user's client device (e.g. a tablet computer). We use Pocketsphynx [16] and Android Speech TTS [15] as open source toolkits for speech recognition/generation, but these components are modularly interchangeable. The interaction management is located on our own AYUDO server. The transmission between *AYUDO Voice* and the Voice Interface API of AYUDO server is secured. Business logic is kept separate from interaction management in a separate domain knowledge module. Currently, we are developing the interaction management module for AYUDO. However, in order to be able to stay flexible and extendable in future, we wanted to separate the design of the AYUDO intention

[3] Snips Homepage (https://snips.ai/, last accessed 2020/10/08).

model from the concrete realization of the interaction management. Therefore, we developed a domain specific modeling language (DSML) for the model-based design of a VUI with the goal to specify the intention model in descriptive form including the request intentions, the user utterances (i.e. what a user can say), existing constraints and corresponding system responses (i.e. what the system can return as output) for our domain. This DSML can be used beyond AYUDO to create intention models independently from (commercial) platforms.

4 Model based Design of Voice User Interfaces

A DSML is designed for exclusive use in a certain domain and for specific purposes [26]. Using the MOF 4-level metamodel hierarchy as a basis [7][17][18], a DSML is an extension of M3 and a metamodel for M1. That means that the DSML is defined on level M2 using a metamodeling language provided on M3. On level M1, the DSML is used to create concrete models that are instantiated on level M0. *Figure 4* shows the metamodel, which defines RIML at level M2 and the AYUDO intention model as an instantiation of RIML. This AYUDO intention model is used on level M0 for the detailed interaction management. In the next subchapters, we will explain RIML, as well as designing an RIML intention model and the intention model at runtime.

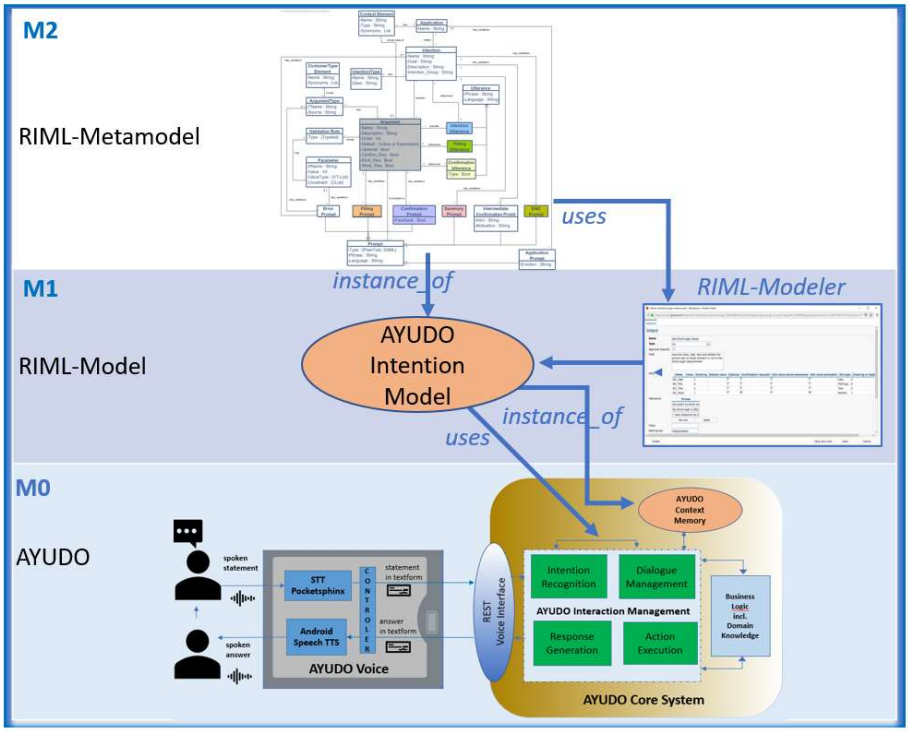

Fig. 4: RIML in the MOF metamodel hierarchy

4.1 The RIML-Metamodel

A RIML model has the purpose to conceptualize to which requests a voice-based system should respond, what intentions are involved, how the system deals with the users' requests and responses, and what has to happen in case of misunderstanding or failure. *Figure 5* presents the meta-model of RIML: The core classes in RIML (some of them are colored in Figure 5) are *Intention*, *Argument*, *Intention Utterance*, *Filling Utterance*, *Confirmation Utterance* and *Prompt.* The class *Intention* enables to model an expected request intention of a user (e.g., document the blood sugar measurement) and represents a specific purpose a user wants to achieve through the interaction with the voice-based system.

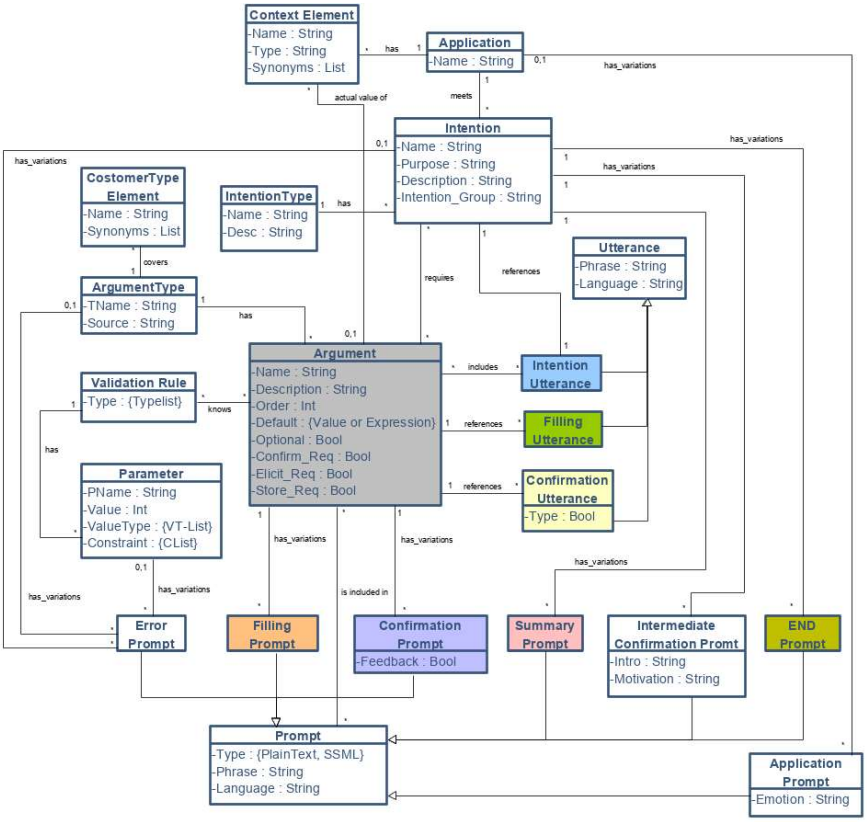

Fig. 5. RIML-Metamodel

Each Intention has an *Intention Type*, e.g., 'ask', to get an answer to a question or 'do', to execute a certain transaction. Each *Intention* can have several instances of the class *Argument*. Arguments are variables, with which the user specifies something. Instances of *Argument* itself can be related to several *Intentions*. For instance, the above-mentioned example intention "document my blood sugar" has the following arguments: blood sugar value, date of measuring, time of measuring, a flag if the user was on empty

stomach or not. An *Argument* can have additional features (e.g., the *Argument Type*, *Order* of an Argument, *Default* value, the *Validation Rules* of an Argument). The metamodel contains the features as attributes or related classes.

Utterances represent the expected statements of the user in a dialogue to fulfill an intention. The class *Utterance* has the following subclasses: *Intention Utterance*, *Filling Utterance* and *Confirmation Utterance*. *Intention Utterance* abstracts statements the user is expected to use to request an intention at all. Instances of *Filling Utterance* represent the expected user responses, which s/he is expected to answer to the voice-based system's request regarding a certain argument of an intention. Instances of *Confirmation Utterance* represent accepted user responses, which affirm or deny a certain previously given message of the voice-based system.

Prompts represent the expected statements of the voice-based system in a dialogue to fulfill an intention. It is possible to model variations of *Prompts* to make a dialogue more diverse later. *Prompt* has several Subclasses: First, there are prompts related to an argument (*Filling Prompt* and *Confirmation Prompt*). *Filling Prompts* are requests for information about a certain argument. Instances of *Confirmation Prompt* either confirm information received from the user or represent follow-up requests for information about an argument. In addition, there exist prompts related to an *Intention* instance itself (*Summary Prompt*, *Intermediate Confirmation Prompt* and *END Prompt*). The intention designer can model instances of these classes to design the overall dialog of a specific intention instance.

Careful error handling is important for the acceptance of voice-based systems. Thus, the class *Error Prompt* is of special nature. An error in a dialogue can e.g., be caused because of the wrong input for a requested argument, the violation of permitted value ranges or the request of an unsupported user intention. Instances of *Error Prompt* are related to instances of the classes *Argument Type* and instances of *Parameter* that belong to instances of *Validation Rule* respectively. Additionally, if a request of a user cannot be resolved at all, the designer can define prompts for that purpose too and relate an *Error Prompt* instance to an *Intention* instance.

4.2 The AYUDO Intention Model

One of the AYUDO project partners[4] has developed the form-based RIML-Modeler to create intention models with RIML. Figure 6 and Figure 7 show an excerpt from the AYUDO intention model which was modeled with the RIML-Modeler:

The instantiation of *Application* on level M1 is named "AYUDO". AYUDO includes several instances of *Intention* e.g., "document the blood sugar measurement" (see Figure 6). We will use this intention to show how to model it, based on RIML. This intention has the *Intention Type* "Do", because data has to be collected and stored permanently in the personal health record of the user. For the considered intention, the following instances of *Argument* exist: "BS_Date" (date of measurement), "BS_Time" (time of measurement), "BS_Value" (blood sugar value) and "BS_FBS" (the flag that indicates if the person was on empty-stomach). Figure 7 shows the specification for the

[4] Groiss Informatics (https://www.groiss.com/en/)

BS_Value. Instances of the classes *Intention Utterance*, *Filling Utterance* and *Confirmation Utterances* are sentence templates with references to *Argument* instances that have to be filled. These references are specified in form of curly brackets, e.g. “I have measured my blood sugar {BS_Time}” or “At {BS_Time}, I had a blood sugar value of {BS_Value}”.

The template style is also used for the instances of the different *Prompt* subclasses. Examples of *Confirmation Prompts* are “Thank you, I understood {BS_Value} as your blood sugar value” or variants like “Ok, your blood sugar value is {BS_Value}”. A *Confirmation Prompt* can also be expressed as a question for feedback (e.g., “You measured {BS_Value}?”). Typical instances of the class *Filling Prompt* are: “Ok, what value for your blood sugar did you measure”; “At which time did you measure the blood sugar value {BS_Value}”). It is even possible to model such instances of a *Prompt* class with SSML [22].

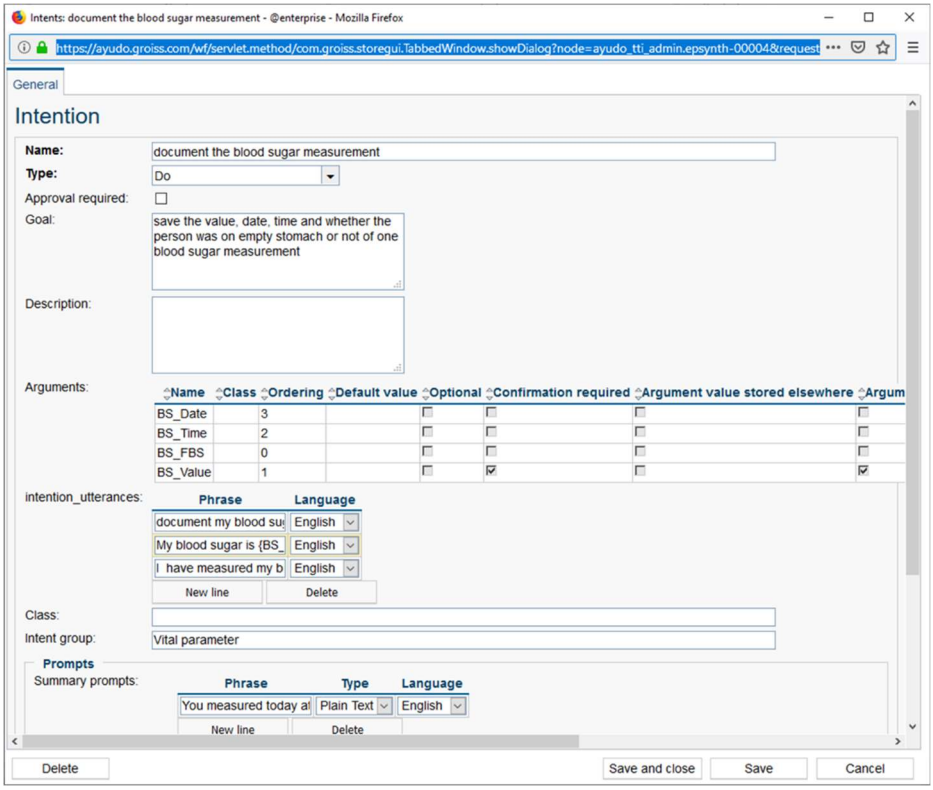

Fig. 6. The Intention “document the blood sugar measurement” modeled with the RIML-Modeler (excerpt)

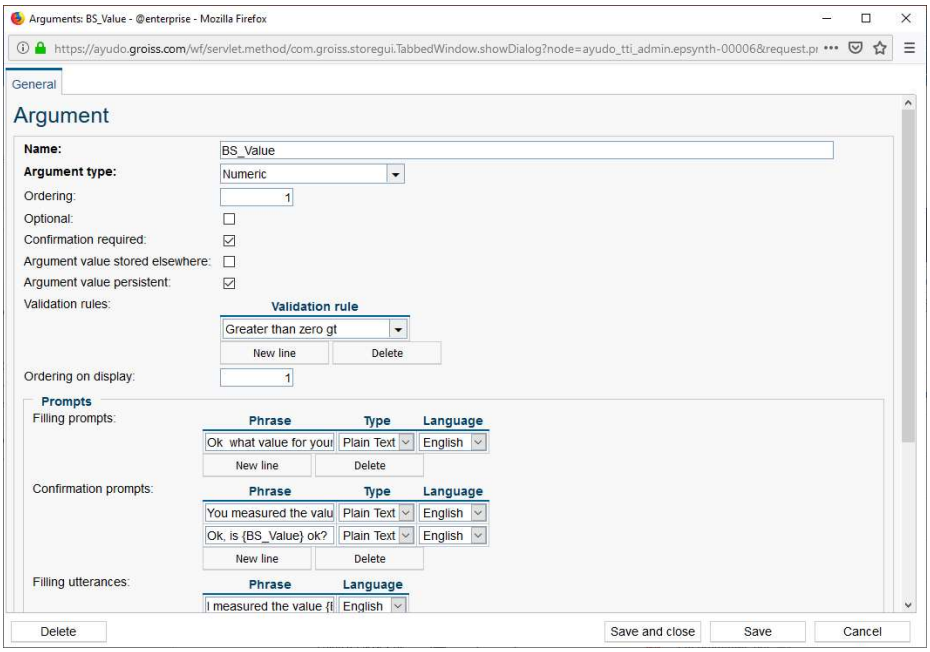

Fig. 7. The Argument "Blood Sugar Value (BS_Value)" modeled with the RIML-Modeler (excerpt)

4.3 AYUDO Voice and interaction management at runtime

Based on the AYUDO intention model presented in Chapter 4.2, a typical dialog between a user, in the following called Mary, and AYUDO Voice could look like sketched in Table 1. The colors of the rows in Table 1 refer to the corresponding classes of the RIML metamodel in Figure 5. The instances of these classes come from the AYUDO intention model (see Figure 6 and Figure 7) and are used by the AYUDO interaction management to keep the dialogue flexible.

Table 1. Dialogue Scenario between User Mary and AYUDO Voice

Role	Conversation
Mary	*AYUDO, I have measured my blood sugar now*
AYUDO	*Ok, what value for your blood sugar did you measure?*
Mary	*I measured the value 90*
AYUDO	*You measured 90?*
Mary	*Oh no, sorry, I measured 95!*

AYUDO	*Ok, is 95 ok?*
Mary	*Oh, yes. This is ok!*
AYUDO	*Did you measure before meals?*
Mary	*Yes, exactly!*
AYUDO	*You measured today at 10.30 am the blood sugar value 95 on empty stomach, ok?*
Mary	*Ok*
AYUDO	*Thanks, I stored this now. Let me know when I can help you again.*

Mary starts the dialogue and speaks to AYUDO Voice. The AYUDO Voice intention recognition component analyses her utterance based on its intention model with the result that it best matches the *Intention Utterance* instance "I have measured my blood sugar {BS_Time}" and concludes the corresponding intention "document the blood sugar measurement". From the word "now" the system concludes that Mary meant the current time. Therefore, the dialogue management tries to find out the missing argument values for this intention. Assuming, that the argument {BS_Value} (= blood sugar value) has the highest priority, it prompts Mary about it. Mary answers with the utterance "I measured the value 90". The interaction management tries to match this response with an instance of *Filling Utterance* and concludes the best match ("I measured the value {BS_Value}"). Now, the system knows that Mary meant the blood sugar value. The interaction management now uses the *Confirmation Prompt* "You measured {BS_Value}?" to check Mary's input. Mary answers and the system tries to find a match of her answer to an instance of *Confirmation Utterance* (i.e., "Oh no, sorry, I measured {BS_Value}").

Afterwards, the interaction management varies its response using once again an instance of *Confirmation Prompt*. This time, Mary confirms and the interaction management uses an instance of *Filling Prompt* to ask for information about Mary's nutrition state when she measured her blood sugar value ("Did you measure before meals?"). Mary agrees and the interaction management matches her utterance with an instance of *Filling Utterance* for this argument (BS_FBS). Using an instance of *Summary Prompt*, the interaction management summarizes what Mary has said and waits for Mary's confirmation. The dialog ends using the instance of the modeled *END Prompt* "Thanks, I stored this now. Let me know when I can help you again". With this prompt, the system tells Mary that the data was stored successfully. Afterwards, the system goes into the idle state.

5 Conclusion and future work

Designing VUIs differs from designing GUIs. Particularly, flexible and natural VUIs have to be designed intention oriented. Big players in language-based system technologies like Amazon and Google have created platform-specific cloud-based solutions and services to design interactions. However, they are not usable platform independently. Data protection and data privacy is often also a critical aspect in the realization of voice-based systems with these technologies.

In this paper, we presented the domain-specific modeling language RIML, which enables a platform independent design of intention models for voice-based systems. With RIML and our RIML-Modeler, intentions and corresponding dialogues can be described in a declarative manner. Based on a RIML model, model centered voice-based systems can keep the intention knowledge also local and protected. They can react flexibly and request the relevant information from the user in a natural way. In this paper, we used the domain of the AYUDO project as a use case.

The first AYUDO intention model has already been created with the RIML-Modeler (20 considered intentions have been modeled) and we are currently working on the development of the interaction management module of AYUDO (see Figure 3). Within the next months, we will evaluate the user experience of *AYUDO Voice* with 10 to 15 test users and adapt our AYUDO intention model to their requirements with regard to the planned AYUDO functions.

In future, we also plan to extend the RIML-Modeler to transform RIML models to platform specific formats (e.g. skill interaction models for ALEXA) and to extend RIML with more context information to make a dialogue more fluent.

References

1. Paterno', F., Santoro, C.,Spano, L. D. (2009). MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Transactions on Computer-Human Interaction (TOCHI), 16(4), 1-30.
2. Brambilla, M., Fraternali, P. (2014). Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML. Morgan Kaufmann.
3. Da Silva, P. P., Paton, N. W. (2003). User interface modeling in UMLi. IEEE software, 20(4), 62-69.
4. Dimaculangan, J. (2019). Will Voice Interactions Replace Screens?. https://careefoundry.com/en/blog/ux-design/will-voice-replace-screens/, last accessed 2020/06/25.
5. Amazon Webinar, How Building for Voice Differs from Building for the Screen, https://build.amazonalexadev.com/how-building-for-voice-differs-from-screen-on-demand-webinar-registration-ww.html, last accessed 2020/10/08.
6. Cutsinger, P. (2018). Situational Design: How to shift from Screen First to Voice First Design. https://build.amazonalexadev.com/vui-vs-gui-guide-ww.html, last access 2020/10/08.
7. Object Management Group: Meta Object Facility (MOF) Specification. www.omg.org/cgi-bin/doc/?formal/02-04-03.pdf, last accessed 2020/10/08.

8. Alvarez, I., Martin, A., Dunbar, J., Taiber, J., Wilson, D. M., Gilbert, J. E. (2011). Designing driver-centric natural voice user interfaces. In Adjunct Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications. online (pp. 156-159).
9. Alexa Skills Kit, https://developer.amazon.com/en-US/alexa/alexa-skills-kit/get-deeper/sdk, last accessed 2020/10/08.
10. Farage M. A., Miller K. W., Ajayi F., Hutchins D. (2012). Design Principles to Accommodate Older Adults. Global Journal of Health Science Vol. 4, No. 2,(March 2012),2-25. DOI: 10.5539/gjhs.v4n2p2
11. Regulation (EU) 2016/679 - General Data Protection Regulation. https://eurlex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679, last accessed 2020/10/08.
12. Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., Primet, M. (2018). Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. arXiv preprint arXiv:1805.10190.
13. Cavoukian, A. (2013). Privacy by design: leadership, methods, and results. In *European Data Protection: Coming of Age* (pp. 175-202). Springer, Dordrecht.
14. Branham, S. M., Rishin Mukkath Roy, A. (2019). Reading Between the Guidelines: How Commercial Voice Assistant Guidelines Hinder Accessibility for Blind Users. The 21st International ACM SIGACCESS on Computers and Accessiblity. ACM, 446 – 468, https://doi.org/10.1145/3308561.3353797.
15. Android Speech TTS overview, https://developer.android.com/reference/android/speech/tts/package-summary, last accessed 2020/10/08.
16. CMUSphynx homepage. https://cmusphinx.github.io/, last accessed 2020/10/08.
17. Meta-Modeling and the OMG Meta Object Facility (MOF), white paper (2017). https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf, last accessed 2020/10/08.
18. Mayr H. C., Al Machot F., Michael J., Morak G., Ranasinghe S., Shekhovtsov V., Steinberger C. (2016). HCM-L: domain-specific modeling for active and assisted living. In Domain-Specific Conceptual Modeling (pp. 527-552). Springer, Cham.
19. FFG Projektdatenbank - AYUDO, https://projekte.ffg.at/projekt/3311832, last accessed 2020/10/08.
20. Pearl, C. (2016). Designing voice user interfaces: principles of conversational experiences. O'Reilly Media, Inc..
21. J. R. Bellegarda (2013). Large-scale personal assistant technology deployment: The siri experience. In: Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH 2013.
22. Speech Synthesis Markup Language (SSML) Version 1.1, https://www.w3.org/TR/speech-synthesis11/, last accessed 2020/10/08
23. Jesse, Mathias Wolfgang (2019). Analysis of voice assistants in eHealth. Master Thesis, July 2019, Universität Klagenfurt.
24. Amunwa, J. (2017). The UX of Voice: The Invisible Interface. Digital Telepathy. Recuperado el, 10. https://www.dtelepathy.com/blog/design/the-ux-of-voice-the-invisible-interface. last accessed 2020/10/08.
25. Holoubek,S., Bowling E. (2019). Voice technology isn't just a trend; it's a paradigm shift, https://www.luminary-labs.com/insight/voice-technology-paradigm-shift/, last accessed 2020/10/08.
26. Frank, U. (2013). Domain-specific modeling languages: requirements analysis and design guidelines. In Domain Engineering (pp. 133-157). Springer, Berlin, Heidelberg.