INTERACTING WITH ROBOTS – TOOLING AND FRAMEWORK FOR ADVANCED SPEECH USER INTERFACES

Christian Hacker[1], *Timo Sowa*[1], *Karl Weilhammer*[1], *Volker Springer*[1], *Dominique Massonie*[1], *Thomas Ranzenberger*[1,2], *Florian Gallwitz*[2]

[1]*Elektrobit Automotive GmbH*, [2]*Technische Hochschule Nürnberg Georg Simon Ohm*
dominique.massonie@elektrobit.com, florian.gallwitz@th-nuernberg.de

Abstract: In this paper we present an extension to our freely available modeling tool for specifying human-machine interfaces in automotive and non-automotive domains. The software tool has been extended to control and use a robot for speech synthesis, speech recognition and gestures. This enables linguists or human factors researchers to easily specify robot behavior for user studies or experiments on efficient human-robot interaction. The paper presents the way to model such a human-robot interface and a use case for a dialog scenario. In the modeling tool, an UML-based state graph is added to specify robot interaction and gestures. During runtime it is executed in parallel to the state graphs for speech dialog and haptic interaction. We showcased interaction of a human with a NAO robot connected to our system based on a weather inquiry system: After some small talk the user can request information from the robot, e.g. the weather forecast. While the robot queries a cloud recognition back end, it performs a thinking gesture and then responds with "Tomorrow it will be between 1 and 8° C and rain in Berlin."

1 Introduction

User interaction with machines is a challenging task for both the user and the applications running on the machine. From the user perspective the interface should be as intuitive as possible, with a seamless integration of multiple modalities. From the application's perspective there are multiple user profiles, multiple configurations, lots of possibilities and also someone needs to specify, run and possibly test all behaviors. For those reasons HMI development tools have been created for specification, prototyping, and usability testing. Our modeling tool and runtime framework EB GUIDE was presented in an earlier version in [1].

In the following sections we will shortly present related work in the field of human-robot-interaction and describe our model driven HMI development approach for haptic and speech dialog interfaces. Our tooling supports a plug-in mechanism that makes it easy to extend it to new modalities like gestures or new applications like robot interaction. Finally we illustrate gesture based speech dialog and describe the use case of a weather inquiry.

2 Human Robot Interaction

Speaking is the most natural way of communication for humans. In order to increase naturalness in human-robot communication, a human-like robot should make use of similar means of communication as humans do in conversations. It should react on speech input, and provide speech output combined with gestures that underline the intended message. Bertsch and Hafner [2] describe a robot that can recognize and learn gestures, in order to engage in gesture based communication with a human. A robot capable of multi-modal interaction is described

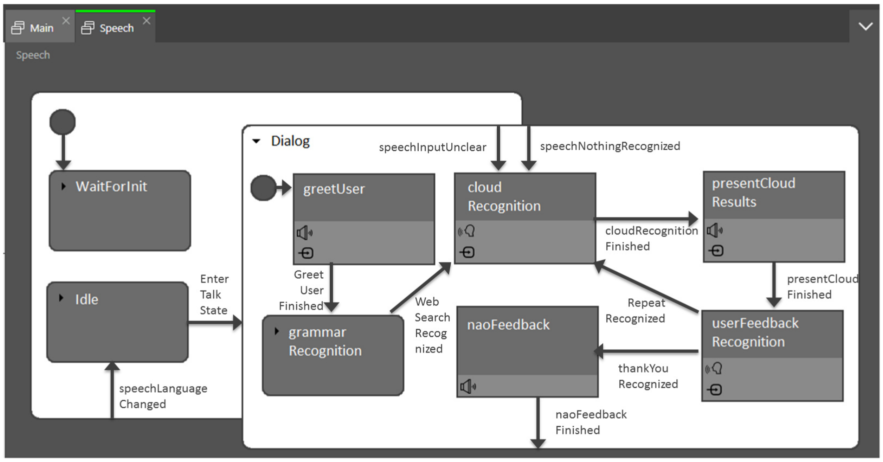


Figure 1 – State chart specifying the speech dialog of the NAO robot.

by Stiefelhagen et al. [3] utilizing pointing gestures and visual gaze. Ng-Thow-Hing et al. [4] built a robot that can synchronize speech and gesture output. In our demonstration system we used a humanoid robot named NAO [5]. It features four microphones for audio input and two speakers for audio output. The robot is equipped with two cameras. The manufacturer provides a simple tool for modeling gestures together with speech dialog.

3 Tool-based Modeling

One cornerstone of modeling HMIs with EB GUIDE is the ability to create multimodal user interfaces using similar modeling means for different modalities. The tool is designed to meet the requirements for HMIs in the automotive field, but can also be applied to other use cases as described in the following. The main modeling concepts are state machines to describe the flow of interaction with the user and modality-specific interaction elements within states that control user input or system output.

3.1 Dialog Model

Interaction with the user is modeled based on state graphs, which are very similar to the UML state diagram notation. A state may hold a *view* which contains graphical or haptic interaction elements called *widgets*, or a *talk* which contains interaction elements for speech output and input called *spidgets (speech gadgets)*. States may also contain scripts to connect to external modules such as the NAO robot.

Figure 1 shows an example of a state diagram for speech interaction. States are depicted as boxes. Some states contain talks for prompt playback, some contain speech recognition, others hide further complexity by recursively containing a (sub-)state graph. The dialog flow is specified with transitions between states, which are activated by events. Events may signal user activity or external input from other parts of the system. If, for instance, a specific command has been recognized or if a voice prompt is finished, the system will trigger an event in order to activate a new state in the dialog. Graph-based speech dialog modeling has a long tradition and has been applied in numerous systems [6]. Speech dialog was typically considered in isolation in such systems. We apply the same logic to all modalities of the dialog: graphic/haptic, speech, and robot gesture. Figure 2 shows another state diagram for haptics/graphics. Each state contains a small version of the view.

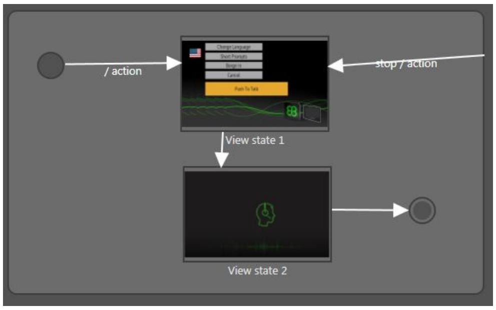


Figure 2 – State model for graphics showing a miniature of the *view* attached to each state.

When a state with a view or talk is reached, it is rendered, which results in a screen change and/or speech output. A speech dialog manager is responsible for rendering talks and synchronizes prompt playback, audio recording, and recognition, and is responsible for the result processing.

Modeling multimodal user interfaces is achieved by running parallel state machines. Each state machine is responsible for the execution of a modality-specific dialog graph. The state machines can be synchronized via the event mechanism, because an event is received and can be processed by all currently active state machines. The mechanism can be used to implement complex patterns such as switching the modality during interaction [7]. For instance, an item on a graphically displayed list can be selected either by touch or speech input. In our proposed system, the robot can be instructed by two modalities, speech dialog and UI control. The gestures of the robot are executed partly independently and are thus specified in a third state graph.

All state machines access a common data model (global properties) to avoid data synchronization issues [8]. The content of the data model can be modified when entering/leaving a state or during transitions. These changes are modeled by the designer in the dialog graph.

3.2 Widgets and Spidgets

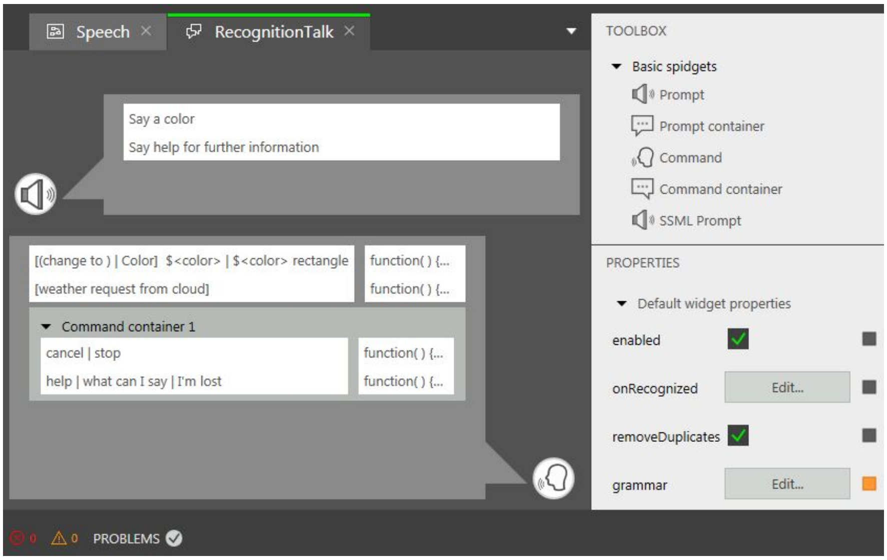
Widgets and spidgets are the basic elements for interaction with the user. They are contained in views/talks and are rendererd as soon as the corresponding state is entered. A widget is a representation for a graphical element with interactive behavior. This could be, for instance, a text field or a button.

Spidgets are the building blocks to construct speech interaction. The spidgets' behavior can be controlled with properties whose values can be set by the modeler. Each kind of spidget has its own set of properties. The *prompt* spidget has a string property defining the text to be synthesized as speech. The string can have plain text or SSML (speech synthesis markup language) format to add prosodic information [9].

The elements for speech input include the *command* spidget which allows recognition based on an SRGS grammar string [10]. The grammar is provided as a property. Slot spidgets are used to add dynamic data to the recognition, which can be any list in the model or content of an external database, e.g. an address-book. They are referenced from the placeholders within SRGS grammars. Figure 3 shows a talk with two prompts and four commands.

There are further dedicated spidgets for connecting to network based (cloud) recognition, or to load special grammars like for address entry. Natural language understanding spidgets react on intent rather than directly on task-oriented commands [11].

Specialized spidgets hide the inherent complexity of the use cases they are derived from, and offer a simplified interface for the dialog designer to be used without any speech expertise.



Say a color
Say help for further information

[(change to) | Color] $<color> | $<color> rectangle	function( ) {...
[weather request from cloud]	function( ) {...

Command container 1
cancel | stop	function( ) {...
help | what can I say | I'm lost	function( ) {...

enabled	✓
onRecognized	Edit...
removeDuplicates	✓
grammar	Edit...

0 0 PROBLEMS

Figure 3 – Talk editor: A Talk represents a single state and contains prompts and commands. The properties of each element (here: command spidget) are shown in the property panel (bottom right).

The spidget for address entry, for instance, only requires the designer to set a property in which country and how to recognize addresses, e.g. if only city, city and street, or combinations of city, street and house number are to be expected and used. The recognized (n-best lists of) cities, streets, and house numbers are provided through further properties as output.

3.3 Templates

Re-usability is one of the biggest advantages of the spidget/widget concept. Dialog patterns can be determined and then converted as a template to be re-used. The system implements only the behavior of the base elements and combining these elements allows to design more complex templates. Re-usability can be leveraged at two levels, namely inside a model and across models. Thus, templates allow higher abstraction and better scalability in the model.

Figure 4 (left) shows a simple button, which is a combination of a rectangle and a text field. The common size of both base elements can be specified in the interface of the more complex element as well as the text which is renamed to label. Additionally a click behavior is added to the button widget. Other properties, like the color are not necessary in the interface.

The same concept for re-usability can be applied to spidgets. Figure 4 (right) shows a simple *yes/no* dialog consisting of a prompt and two command spidgets. Only the prompt and the actions of the commands are part of the interface and can be adapted in each instance of the dialog. The SRGS grammar specifications for *yes* and *no* are hidden from the interface. When used within different states of the business logic, the command yes/no can result in different behavior. The actions specify the behavior which is in the most simple case an event trigger to activate a new state.

3.4 Multilinguality and Variant

A big challenge for speech dialog systems is multilinguality and variant handling [12]. In particular in automotive applications the market is spread all over the world and a number of languages have to be supported. Additionally, different variants of the system with partly different behavior are produced. This requires on the one hand variant dependent transitions in

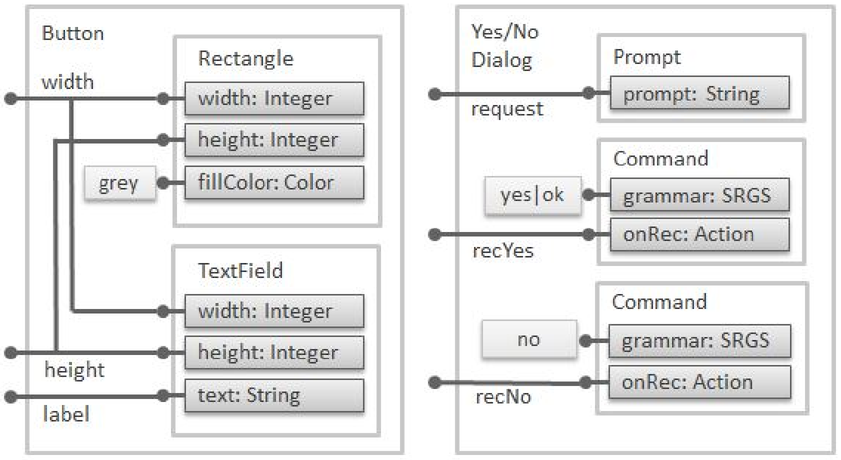


Figure 4 – An example of a widget template and a spidget template.

the business logic and on the other hand translations of display texts, voice prompts, and speech recognition grammars. Our response to these requirements is a common concept for variant handling of all global properties. A boolean property influencing the dialog flow can be in the same way language- and variant-dependent as a string property which is used in a display text. Also spidget properties have references to global properties. An SRGS grammar has language dependent structure and wording, but also the references to sub-grammars and dynamic content (slots) may have a language dependent order.

4 Connecting the Robot to the System

The aim of the system is to enable the robot to output speech and simultaneously execute gestures. We developed an inventory of robot gestures (see NaoGestures in Figure 5) that control the robot movements with the help of the NAOqi framework. To enable modeling of robot gestures together with our modeling approach additional components are needed.

Figure 5 shows an overview of the system components. Our modeling tool EB GUIDE Studio provides separate state machines to specify speech dialog and robot gestures and generates a model that can be read by the target framework. The Grammar Conversion Tool prepares all resources for the speech recognizer including recognition grammars. Studio and grammar converter run only on the PC.

Target Framework is the platform dependent runtime environment to run the exported model e.g. on Windows, Android, or Linux. It includes the speech dialog manager (SDM) which provides different possibilities for speech recognition. SDM can use a grammar based recognizer which is contained in the Speech Target Framework or connect to a cloud based recognizer via the SdmNetProxyPlugin. The SdmNaoPlugin extends the SDM: it forwards all voice prompts to the robot and provides additional script functions to allow the specification of the robot behavior within EB GUIDE Studio. The plugin uses the NaoGestures library to perform the published script functions. The NaoGestures library itself implements gestures like “moving hands” and connects to the NAOqi framework on the robot to perform the behaviors and to provide feedback about the speech output for the current dialog state. Our cloud based recognition scenario shows how the components work together. In this scenario the user starts a

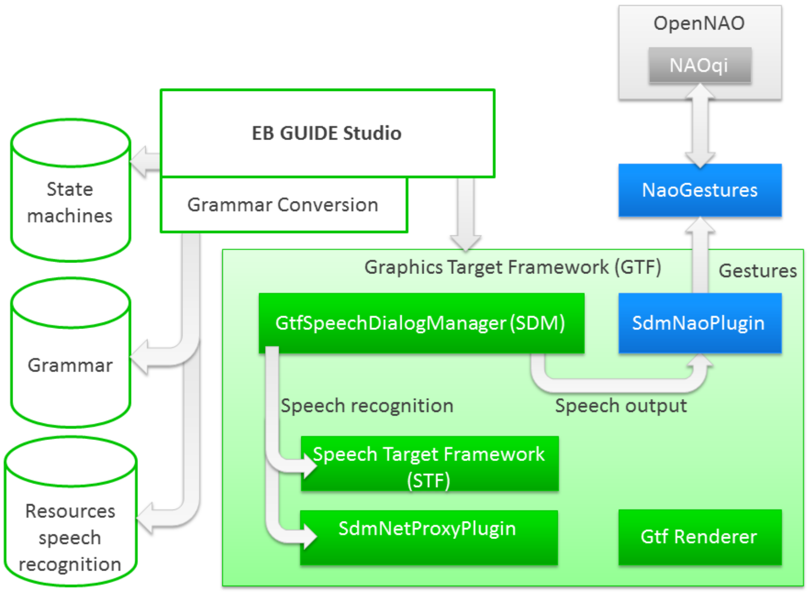


Figure 5 – System overview: The modeling tool EB GUIDE Studio (top, left) and the target runtime environment GTF (bottom) with speech dialog manager (SDM) is extended with a new plugin components (SdmNaoPlugin and NaoGestures) to connect to the NAOqi framework (top, right).

conversation with the robot. The speech dialog of this conversation is modeled in a state graph as shown in Figure 1.

Gestures and other non-speech related functions are modeled in an additional state machine. This state machine calls the available NAO functions, methods and behaviors. The states of the NAO state machine use entry actions to execute script functions of the NaoGestures library via the NaoSdmPlugin. For example by activating the state *startFaceTracking* the robot face tracking module is started. In the speech state machine events are fired to trigger corresponding states in the NAO state machine. As shown in Figure 1 the speech state machine has one state for a grammar based recognition. In this state SRGS based commands are specified. If the user speaks one of these commands, the speech state machine enters the state *cloudRecognition*. While entering the state an event starts the recognition animation (inside *Entry action*). If the user has finished speaking to the robot the cloud recognition starts and fires an event to initiate the thinking animation. So the NAO state machine will enable the state *startThinking* to start the animation. After the cloud result is received an event is triggered to enable the state *stopThinking* in the NAO state machine and *presentCloudResults* in the speech state machine. In the next section we will step by step describe what happens through a dialog with gestures.

5 Dialog with Gestures

We used our extended modeling framework to specify a simple speech dialog between user and robot where the user requests information from a cloud service, e.g. the current weather in any city or information about famous persons. Some gestures of the cloud recognition scenario are shown in Figure 6 and Figure 7. To activate the cloud recognition the user asks the robot to perform a web search. After the grammar based recognition for the search command the cloud based recognition is activated. During the cloud recognition the thinking animation is performed. The robot moves the right arm near the head while the head is moving a little bit up. While moving the head slightly up and down together with random lightning eye led's the robot provides the feedback to the user that the recognition is ongoing. After the user finished speaking the robot queries the cloud for information. If the cloud result is available the robot stops the thinking animation and presents the received result to the user: *"Tomorrow it will be between 1 and* 8° *C and rain in Berlin."* Another animation with moving hands is available while NAO is presenting the results to the user.

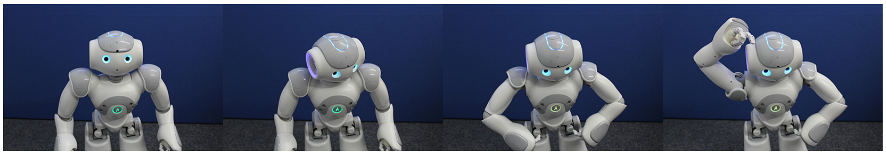


Figure 6 – Animation when switching from grammar based recognition to cloud recognition.

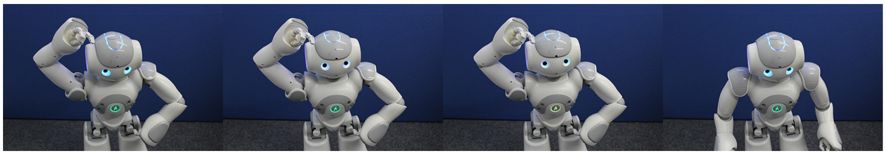


Figure 7 – Animation during one cloud recognition.

6 Conclusion

In this paper we presented a modeling tool for different modalities including speech dialog and gesture. The tool has been designed for automotive use cases including cloud based recognition. Different UML based state graphs can be designed and executed in parallel; they are synchronized via events and a common data model. The target framework is available for several platforms like Windows, Android and others. It contains the speech dialog manager to execute the model. The framework can be extended with plugins to connect the system to the NAO robot as described in this paper. This allows to specify NAO gestures in the modeling tool in addition to the speech dialog. We explained a weather inquiry system which uses a recognizer in the cloud.

References

[1] MASSONIE, D., C. HACKER, and T. SOWA: *Modeling Graphical and Speech User Interfaces with Widgets and Spidgets. ITG-Fachbericht: Speech Communication*, 252, 2014. VDE Verlag GmbH, Berlin/Offenbach, ISBN 978-3-8007-3640-9.

[2] BERTSCH, F. A. and V. V. HAFNER: *Real-time dynamic visual gesture recognition in human-robot interaction*. Tech. Rep., Humboldt-Universität zu Berlin, 2009. URL `https://adapt.informatik.hu-berlin.de/pub/papers/humanoids09_gestures.pdf`.

[3] STIEFELHAGEN, R., H. K. EKENEL, C. FUGEN, P. GIESELMANN, H. HOLZAPFEL, F. KRAFT, K. NICKEL, M. VOIT, and A. WAIBEL: *Enabling Multimodal Human Robot Interaction for the Karlsruhe Humanoid Robot. IEEE Transactions on Robotics*, 23(5), pp. 840–851, 2007.

[4] NG-THOW-HING, V., P. LUO, and S. OKITA: *Synchronized gesture and speech production for humanoid robots*. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4617–4624. 2010.

[5] GOUAILLIER, D., V. HUGEL, P. BLAZEVIC, C. KILNER, J. MONCEAUX, P. LAFOURCADE, B. MARNIER, J. SERRE, and B. MAISONNIER: *Mechatronic design of NAO humanoid*. In *ICRA '09, IEEE International Conference on Robotics and Automation*, pp. 769–774. 2009.

[6] COLE, R.: *Tools for Research and Education in Speech Science*. In *Proc. of the Int. Conference of Phonetic Science*, pp. 1277–1280. 1999.

[7] FLEISCHMANN, T.: *Wie sage ich es meinem Auto? AutomobilKonstruktion*, 1/2014, pp. 46–47, 2014.

[8] HAYES-ROTH, B.: *A blackboard architecture for control. Artificial Intelligence*, 26(3), pp. 251–321, 1985.

[9] SHUANG, D., Z.W.AND BURNETT: *Speech Synthesis Markup Language (SSML) Version 1.1*. W3C recommendation, W3C, 2010. URL `http://www.w3.org/TR/speech-synthesis11/`. Http://www.w3.org/TR/2010/REC-speech-synthesis11-20100907/. Latest version available at http://www.w3.org/TR/speech-synthesis11/.

[10] HUNT, A. and S. MCGLASHAN: *Speech Recognition Grammar Specification (SRGS) Version 1.0*. W3C recommendation, W3C, 2004. URL `http://www.w3.org/TR/speech-grammar/`. Http://www.w3.org/TR/2004/REC-speech-grammar-20040316/. Latest version available at http://www.w3.org/TR/speech-grammar/.

[11] WEILHAMMER, K., P. KUMAR, V. SPRINGER, and D. MASSONIE: *Spoken language understanding in embedded systems*. In O. JOKISCH (ed.), *Elektronische Sprachverarbeitung 2016, Tagungsband der 27. Konferenz*, vol. 81 of *Studientexte zur Sprachkommunikation*, pp. 69–76. TUDpress, Leipzig, Germany, 2016.

[12] ZUE, V. and J. GLASS: *Conversational Interfaces: Advances and Challenges. Proceedings of the IEEE*, 88(8), pp. 1166–1180, 2000.