A Toolkit for 3D-Gesture and Speech Dialog in Automotive Environments

Timo Sowa[1]*, Alexander Richter*[1]*, Dietmar Fey*[2]

[1]*Elektrobit Automotive GmbH,* [2]*Universität Erlangen-Nürnberg*
timo.sowa@elektrobit.com

Abstract: 3D-gesture is a new input modality that complements touch-screens and speech dialog systems for automotive infotainment systems. Yet, tools for implementing multimodal interfaces including 3D-gestures are rare. We present a toolkit for in-car human-machine interfaces (HMIs), that allows modeling 3D-gesture recognition and integration of gestures with speech. Our approach focuses on three interaction styles: Firstly, we demonstrate how dialogs containing isolated gestures, sequences of speech and gesture, and alternative uses of different modalities can be modeled based on the state graph paradigm. Secondly, we show how continuous gestural interaction, such as gradually modifying a value with continuous hand movements, can be designed by binding hand model parameters to HMI model values. Thirdly, we present an approach toward modeling and processing of multimodal utterances with semantics distributed over speech and 3D-gesture. The approach is based on a semantic representation of the individual modalities as typed feature structures which are collected in a short-term memory. An integrator module unifies partial meanings until a fully specified command is created which can be executed. The approach has been implemented in a demonstration system using a Leap Motion controller.

1 Introduction

3D-gestures are hand movements performed mid-air without contact to a touch-sensitive surface. Infotainment systems supporting 3D-gesture recognition have been recently introduced to the market. BMW, for instance, offers a system that allows controlling the sound volume with 3D-gestures, accepting or declining phone calls, rotating an image of the car, and freely binding a function to a custom gesture. VW's high-end infotainment system recognizes right/left swipe gestures for horizontally ordered menus, for photo browsing, or music album selection. Gestures possess ergonomic and cognitive advantages over touch-screens. They involve less strain, because no distant surface needs to be reached. In addition, targeting movements are unnecessary, leading to less visual distraction of the driver [1]. Gestures further facilitate a continuous style of interaction with gradually changing values. This is barely realizable with speech and, being constrained to one or two dimensions, restricted for controls like turning knobs or touch-screens.

When 3D-gestures are used in isolation, they are usually bound to a certain function of the human-machine interface (HMI) requiring the user to memorize which gesture means what. This led to research efforts defining how a suitable gesture set may look like [2]. Early spoken dialog systems in cars similarly required the users to memorize specific commands for certain functions. Present-day systems, however, allow for much more naturalness and variability in spoken language. Instead of defining isolated gesture sets, more naturalness could be achieved

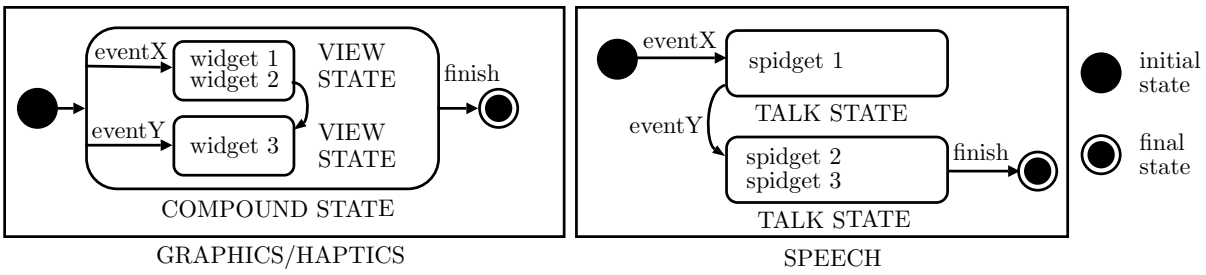

Figure 1 – Basic structure of an HMI model: State graphs with states and transitions. States contain interactive elements. Transitions are labeled with events.

by using gestures in the context of speech, because this is where they usually occur in conversation. The same gesture could be used for different functions depending on the verbal context and, vice versa, different gestures sharing a form property could trigger the same HMI function in a certain verbal context. Multimodal speech and 3D-gesture input offers new styles of interaction, in particular when considering current trends in automotive HMIs. With displays getting larger and more flexible, but also more distant, covering the entire dashboard or being projected in the driver's view, there are new opportunities for interaction design. Distant 3D-gestures and speech could be employed in these scenarios, even more so when thinking toward autonomous driving. Furthermore, 3D-gesture and speech interaction is useful for passengers, for instance, in backseat entertainment where distant displays could be more conveniently controlled.

The development of advanced multimodal infotainment systems benefits from the existence of HMI modeling tools that support these novel input techniques and allow their seamless integration with other modalities to a cohesive user experience. We present an extension of an existing toolkit for in-car HMIs [3], adding modeling means and run-time handling for 3D-gesture recognition and integration with speech. It enables non-programmers to easily model multimodal dialogs, develop prototypes, and deploy the HMI system on an embedded hardware. In the following section we discuss related work, and briefly explain the modeling approach of our existing toolkit in section 3. We describe the modeling extensions for integrating 3D-gestures in section 4, for continuous gesturing in section 5, and for integrating multimodal utterances in section 6. Finally, we present a prototype system where these concepts have been applied.

2 Related work

Seminal work on multimodal integration of speech and gesture was conducted by Bolt [4]. Johnston et al. [5] introduced the unification of typed feature structures as integration paradigm. Their approach is largely adopted in the current work. Skantze and Al Moubayed [6] suggest a statechart-based toolkit for modeling face-to-face interaction. The dialog flow is modeled with an extension to Harel's state chart formalism. This is comparable to the approach used here (cf. following section). Mehlmann and Andre [7] present an approach for the combined modeling of multimodal fusion and interaction management. Fusion is conducted within a finite state-machine by adding temporal, spatial, and semantic constraints. Latoschik [8] describes a framework for multimodal interaction in virtual reality also covering continuous gesturing with use cases comparable to the current work.

3 A modeling toolkit for automotive HMIs

Our HMI modeling approach is broadly based on Harel state graphs [9] (Figure 1).[1] States can be nested in *compound states*. Nested states inherit transitions defined for the parent. Transi-

[1]The approach is implemented in the HMI modeling software EB GUIDE.

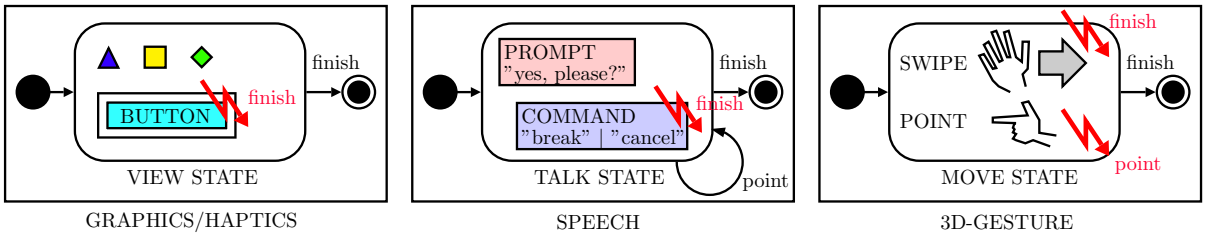

Figure 2 – An HMI model featuring three state graphs for different modalities.

tions between states are triggered by events which may be fired at any time during processing. During run time, a state machine is instantiated per graph and processes the state graph model.[2] An HMI may consist of multiple state graphs processed in parallel. Synchronization between different state machines is achieved with events, because they are distributed to all state machines. There are separate graphs for graphic/haptic and for speech interaction. The former contains *view states* describing the visual scene rendered on the display(s). View states contain *widgets* for visual/haptic interaction with the user. The state graph for speech contains *talk states* defining the dialog flow. Talk states contain *spidgets* ("speech widgets") describing the system's speech output (prompts) and recognizable input (commands with slots) [3]. When a state is entered, the elements it contains are getting rendered.

Each kind of interactive element (widgets and spidgets) is configured with a set of *properties*. Properties are named local variables that can be set by the modeler and may change during run time. A command spidget for speech input, for instance, has a property defining the recognizable utterances. The *datapool* is a common data model consisting of named global variables. The system's behavior may depend on the state of the datapool.[3] Widget/spidget properties can be linked to a datapool variable. That way, properties of different elements and even in different state graphs can be synchronized. Their values are automatically propagated. Note that "being in a state" by no means implies that the HMI is statically waiting for a user input. With bindings between properties and the datapool and perpetual changes in the datapool, the HMI may continually update and change even when there is no state transition.

4 Modeling isolated gestures, multimodal sequences, and alternations

For handling 3D-gesture input we introduce a third state graph (Figure 2). The graphics/haptic graph contains a view state with graphical elements and a touch button. Upon touching, the event *finish* shall be fired (indicated with the red flash). The speech graph contains a talk state with a system prompt and a command that recognizes the words "break" and "cancel". The event *finish* is fired on recognition. Finally the new graph for 3D-gestures contains a *move state* with two *gedgets (gesture widgets)* for the recognition of a swipe to the right and a pointing gesture.

At run time all three state machines directly transition from the start to the respective main states. The graphic/haptic elements defined in the view state are displayed on the screen. Simultaneously, the talk state is rendered causing the prompt "yes, please?" to be played, and triggering the speech recognizer. Recognition of the gestures defined in the move state is also concurrently triggered. If the user said "break", the finish event would be fired, leading to a transition to the final state in all state graphs. The same would happen in case of a swipe-to-the-right gesture. A pointing gesture, however, would cause the speech state machine to enter the talk state once again, playing the prompt and triggering speech recognition.

The example demonstrates how 3D-gesture can be smoothly integrated with other modal-

[2]The terms *state graph* and *state machine* are thus often used synonymously.

[3]Note that this is an extension to pure Harel state charts.

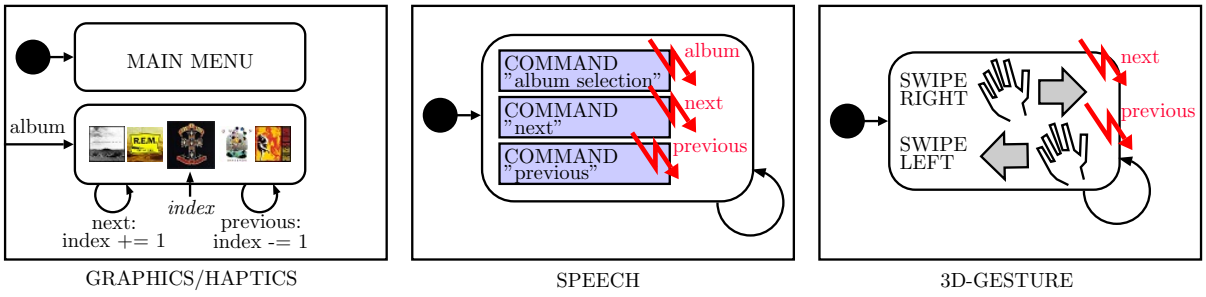

Figure 3 – An HMI model for subsequent and alternative use of speech and gesture input.

ities: For *isolated gestures* that trigger a specific function of the HMI, a move state with an appropriate gesture recognizer is modeled. On gesture recognition, an event is fired that may lead to a transition in another state machine (causing a display/menu to pop up or a speech dialog to be initiated). Similarly *sequences* of speech and 3D-gesture can be modeled using multiple state charts. For instance, a chooser for music albums can be summoned up with a speech command as depicted in Figure 3. Subsequent browsing in the album list can be controlled either with swipe gestures, or with speech commands ("next", "previous"), which exemplifies *alternative* use of modalities to achieve a goal. Note that the graphic/haptic state machine executes an action which is bound to receiving a *next* or *previous* event. A counter variable *index* which is defined in the datapool is increased or decreased accordingly. The state machine enters the view state showing the album covers again (internal transition), but the display is now showing the cover with the new index as the selected album.

5 Continuous interaction

5.1 Sustained dynamic gestures

A simple use case for continuous interaction is increasing and decreasing the volume setting with a rotational gesture. In this case the gesture cannot be regarded as a single event, but the system needs to keep track of when the rotation starts and when it ends. As long as the rotational movement is sustained, the volume is increased (or decreased). In order to model this behavior, the interactive element for rotational gesture recognition gets two properties. One defines the actions on movement onset, the other one on movement cessation. Figure 4 depicts an HMI model for this use case. The gesture state graph contains two elements, one for recognizing a clockwise, and one for counterclockwise rotations with the extended index finger. On onset recognition the event *tickInc* is fired (or *tickDec* for ccw movement). Furthermore some values are defined in the datapool. First, there is the current volume setting as an integer value. Additionally, two *conditional scripts* are defined. These scripts are triggered by the tick events, and cause the volume setting to increase/decrease. To make sure that the volume change sustains, the scripts themselves fire the tick events, but with a certain temporal delay. That way, the conditional scripts are getting called periodically, gradually modifying the volume setting. To stop the gradual volume change, any pending tick events are canceled on movement cessation. The volume controlling unit/device simply reads the current volume setting from the datapool.

5.2 Binding hand model values to the HMI model

A more elaborate form of continuous interaction with 3D-gestures consist of mapping hand values to values of the HMI model. This could relate, for instance, to the palm position in 3D space, joint angles, positions of the fingertips, etc. Rotating, moving, and scaling a 3D model

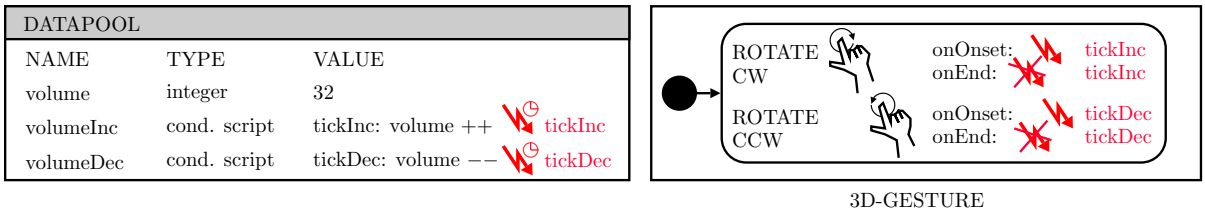

Figure 4 – An HMI model for volume control with sustained 3D-gestures. Volume is gradually increased/decreased with temporally delayed events.

of the car with a pinch gesture can be a use cases for this interaction style. 3D-gestures could also be used for volume control in space, for instance by moving the sound source more to the right/left or to the backseats. Note that only gesture offers this flexibility where multiple dimensions can be controlled at once.

We introduce a new interactive element to be used in move states of the gestural state graph. The *hand model* element is a gedget providing the hand configuration values in its properties. Figure 5 illustrates how to model the pinch-and-move gesture use case. The gesture state graph has two move states. The first one, entered initially, contains a gedget to recognize a pinch gesture (tip of thumb and index have contact). Upon recognition the *pinch* event is fired leading to a transition to the second move state. The state contains a hand model gedget providing the hand's configuration values. When the move state is entered the hand model gets activated and its properties are continuously updated. Via datapool binding, the values for palm x and y positions are propagated to a 3D display widget in the graphics/haptics state graph showing a model of the car and bound to the properties for object rotation (*rotX* and *rotY*). Concurrently to the hand model another gegdet for pinch gesture recognition is active firing the *release* event if the gesture ends. This leads to a transition to the first move state, thus deactivating the hand model until another pinch hand shape is recognized.

6 Processing multimodal utterances with 3D-gesture and speech

The previous use cases focused on multimodality in a sequential or alternative sense and on continuous gesturing. Understanding *true* multimodal utterances, however, means to integrate the meaning fragments coming from each modality to a complete structure. Figure 6 depicts our approach based on a use case where several warning icons are displayed on the dashboard while the user asks "What does this red icon mean?" and concurrently points to the display (indicated by a crossbar). In the following, we describe the modules and processing steps involved in the integration process.

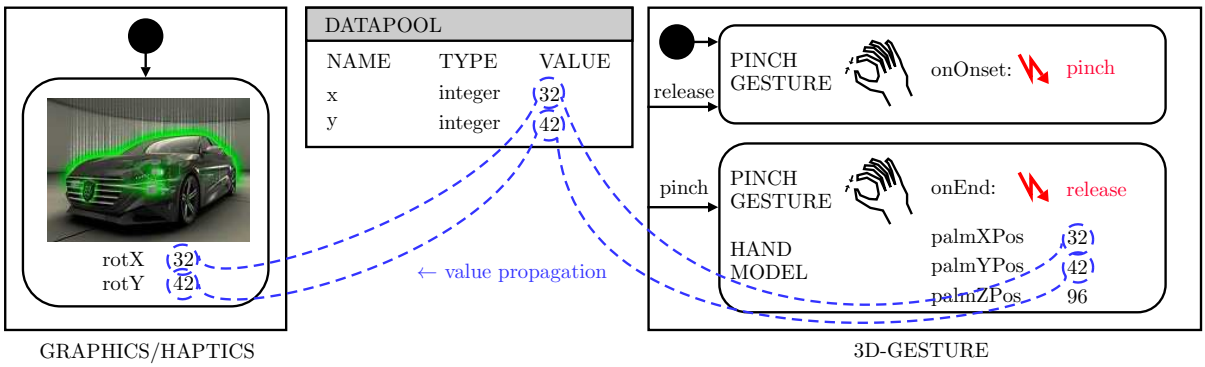

Figure 5 – An HMI model for a pinch-and-move gesture to rotate a 3D model in two axes. The x and y positions of the hand are translated into rotations via a datapool binding.

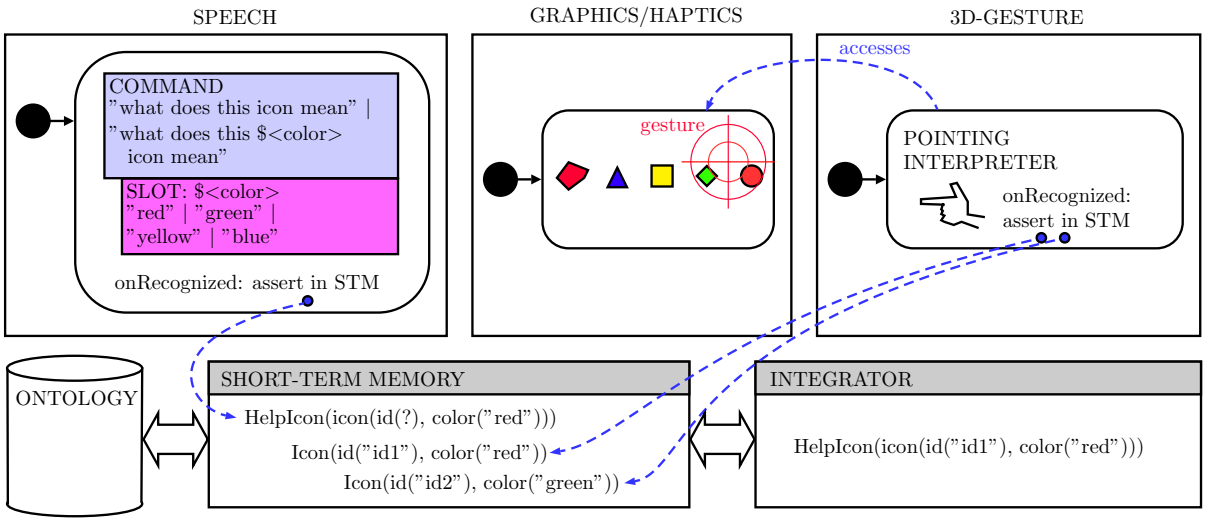

Figure 6 – Multimodal integration of 3D-gesture and speech for a pointing use case. User asks "What does that red icon mean?" while pointing toward the visual scene.

6.1 Ontology

The ontology is an external resource defining the known classes of objects the multimodal integration approach can deal with, including their hierarchy of inheritance and part-of relations. Figure 7 shows a simple ontology for the example use case. All classes are derived from `Object`. `Intent` serves as a superclass for all executable commands the user may issue to the HMI system. In the example, the ontology may contain a class `HelpIcon` derived from `Intent`, representing a user's query to get help about a specific icon being displayed. The icon is defined as a named member of class `Icon` deriving from `Entity`. An `Icon` has a unique `Id` and a `Color`, both of integral (string) type. Members (parts) of a class can be defined as mandatory or optional. An instance of a class is *completely defined* if all its mandatory members are completely defined. Instances of integral classes are completely defined if they have a value. The ontology can be freely modified by the modeler.

6.2 Unimodal interpretation

The information provided by each modality is independently interpreted in the talk/move states producing instances of the classes defined by the ontology. (cf. Figure 6). The talk state contains a command for a query on an icon with a *slot* for the color. Upon recognition, an instance of the intent `HelpIcon` is asserted in *short-term memory*. The verbal utterance ("What does that red icon mean?") includes the icon's color, but the id is yet unknown. Thus, the logical form `HelpIcon(icon(id(?), color("red")))` is asserted. The pointing gesture provides information on potential icons. The pointing interpreter computes screen coordinates from the

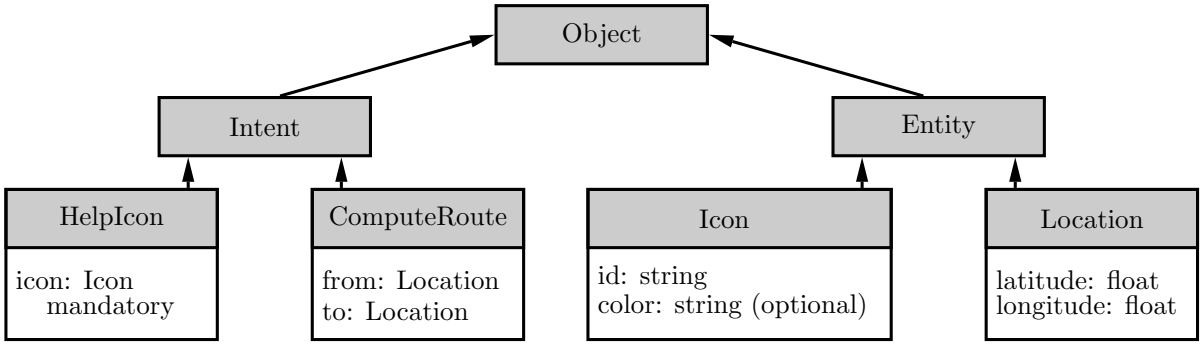

Figure 7 – A simple ontology for the pointing use case. Boxes depict classes, arrows show is-a relations.

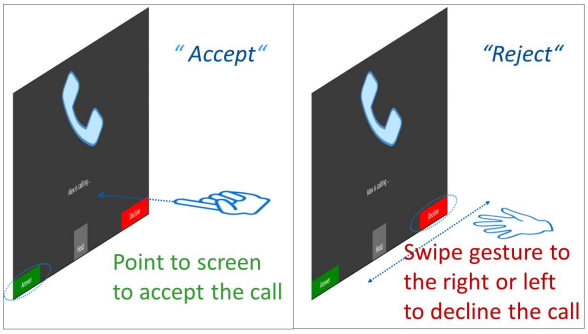

Figure 8 – Demo use case 1: Accept/decline an incoming call.

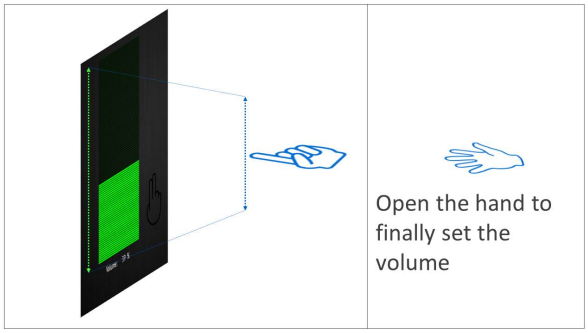

Figure 9 – Demo use case 2: Change volume with a pointing gesture and up/down movement of the hand.

gesture. Assuming that there is always some inaccuracy in pointing, the interpreter asserts icons in short-term memory, which roughly match the target coordinates – the green and red icons on the right-hand side in our example. This will lead to two instances of the entity class `Icon` being asserted, with the logical forms `Icon(id("id1"), color("red"))` and `Icon(id("id2"), color("green"))`.

6.3 Short-term memory and multimodal integration

All instances are collected in short-term memory. Only those instances asserted within a limited time span shall be integrated. This is because speech and gesture in a multimodal utterance typically have a close temporal relationship. The short-term memory keeps instances for a few seconds. If they cannot be integrated to a completely defined intent within this time span, they are deleted. The *integrator* component operates on short-term memory trying to unify and integrate incomplete instances that fit together. We adopted unification of typed feature structures as integration approach [5]. In case an intent could be integrated to the point where it is completely defined, it is forwarded to the HMI model and can be executed. In Figure 6 the logical form `HelpIcon(icon(id(?), color("red")))` from speech can only be integrated with the form `Icon(id("id1"), color("red"))` from gesture, while the second alternative `Icon(id("id2"), color("green"))` does not match due to the difference in color.

7 Prototype system

The modeling concepts described previously and the multimodal integration engine were implemented as extensions to EB GUIDE.[4] We used the Leap Motion 3D controller for the demonstration system.[5] The modeling tool GUIDE Studio was extended by a state machine for 3D-gesture input. Three gedgets were implemented, one for isolated recognition of several gestures including the swipe gesture, one for the hand model, and a gedget for the interpretation of pointing gestures to the visual scene. The gedgets use an API provided by Leap Motion to access the controller. Three use cases have been implemented. To demonstrate isolated gesture recognition, a simulated incoming phone call can either be accepted with a pointing gesture, by saying the command “accept”, or by pressing the accept button on the screen. It can be declined with a swipe gesture. For continuous interaction, a volume controller has been implemented that is activated with a pointing gesture. While this gesture is sustained, the volume can be changed by moving the hand up and down. Opening the hand will finally set the volume. Finally, we implemented multimodal integration of speech and 3D-gestures referring to icons on the screen.

[4] https://www.elektrobit.com/ebguide/
[5] https://www.leapmotion.com/

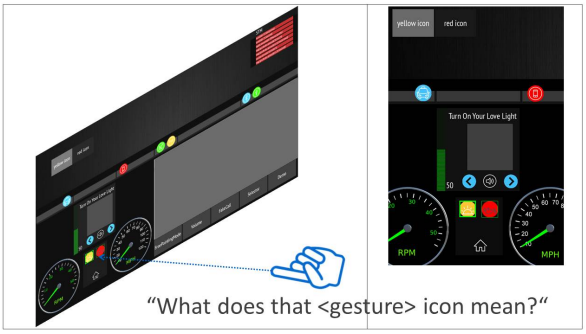

Figure 10 – Demo use case 3: Asking the system about the meaning of a warning icon on the display.

8 Conclusion

We have shown how 3D-gestures can be incorporated in automotive HMIs and how a state graph-based modeling tool can be extended such that modeling multimodal interaction in a common framework is possible. In particular, we proposed modeling techniques for continuous interaction with 3D-gestures and an approach toward interpreting multimodal utterances with distributed semantics.

References

[1] GEIGER, M., M. ZOBL, K. BENGLER, and M. LANG: *Intermodal differences in distraction effects while controlling automotive user interfaces*. In *Proc. Int. Conf. on Human-Computer Interaction (HCI 2001)*, pp. 263–267. Lawrence Erlbaum Ass., NJ, 2001.

[2] MAY, K., T. GABLE, and B. WALKER: *Designing an in-vehicle air gesture set using elicitation methods*. In *Proc. of the 9th ACM Int. Conf. on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI'17)*, pp. 74–83. ACM, New York, 2017.

[3] MASSONIE, D., C. HACKER, and T. SOWA: *Modeling graphical and speech user interfaces with widgets and spidgets*. In *Proc. of the 11th ITG Symposium on Speech Communication*. VDE, 2014.

[4] BOLT, R.: *"put-that-there": Voice and gesture at the graphics interface*. *Computer Graphics*, 14(3), pp. 262–270, 1980.

[5] JOHNSTON, M., P. R. COHEN, D. MCGEE, S. L. OVIATT, J. A. PITTMAN, and I. SMITH: *Unification-based multimodal integration*. In *Proc. of the 35th Annual Meeting of the Assoc. for Comp. Linguistics (EACL '97)*, pp. 281–288. 1997.

[6] SKANTZE, G. and S. AL MOUBAYED: *IrisTK: a statechart-based toolkit for multi-party face-to-face interaction*. In *Proc. of the ACM Int. Conf. on Multimodal Interaction (ICMI)*, pp. 69–76. 2012.

[7] MEHLMANN, G. and E. ANDRE: *Modeling multimodal integration with event logic charts*. In *Proc. of the ACM Int. Conf. on Multimodal Interaction (ICMI)*, pp. 125–132. 2012.

[8] LATOSCHIK, M.: *A general framework for multimodal interaction in virtual reality systems: PrOSA*. In *The Future of VR and AR Interfaces. Proc. of the Workshop at IEEE Virtual Reality 2001*, pp. 21–26. 2001.

[9] HAREL, D.: *Statecharts: A visual formalism for complex systems*. *Science of Computer Programming*, 8(3), pp. 231–274, 1987.