IMPROVING SPEECH INTERACTION IN VEHICLES USING CONTEXT-AWARE INFORMATION THROUGH A SCXML FRAMEWORK

Álvaro Sigüenza[1], José Luis Blanco[1], David Conejero[2] and Luis Hernández[1]

[1] Signal, Systems and Radiocommunications Department, Universidad Politécnica de Madrid
[2] Telefónica Research and Development

alvaro.siguenza@gaps.ssr.upm.es

Abstract: Speech Technologies can provide important benefits for the development of more usable and safe in-vehicle human-machine interactive systems (HMIs). However mainly due robustness issues, the use of spoken interaction can entail important distractions to the driver. In this challenging scenario, while speech technologies are evolving, further research is necessary to explore how they can be complemented with both other modalities (multimodality) and information from the increasing number of available sensors (context-awareness). The perceived quality of speech technologies can significantly be increased by implementing such policies, which simply try to make the best use of all the available resources; and the in-vehicle scenario is an excellent test-bed for this kind of initiatives. In this contribution we propose an event-based HMI design framework which combines context modelling and multimodal interaction using a W3C XML language known as SCXML. SCXML provides a general process control mechanism that is being considered by W3C to improve both voice interaction (VoiceXML) and multimodal interaction (MMI). In our approach we try to anticipate and extend these initiatives presenting a flexible SCXML-based approach for the design of a wide range of multimodal context-aware HMI in-vehicle interfaces. The proposed framework for HMI design and specification has been implemented in an automotive OSGi service platform, and it is being used and tested in the Spanish research project MARTA for the development of several in-vehicle interactive applications.

1 Introduction

The appearance and increasing presence of communication and information technologies in vehicles is providing drivers with an also increasing number of new services as In-Vehicle Information Systems (IVIS) and Advanced Driver Assistance Systems (ADAS). Driving is a complex task that demands high interaction and coordination of driver's physical, perceptual and mental skills. Therefore the interaction between the driver and these new services must be carefully designed, as a secondary task, so that the driver can keep his/her attention on driving safely.

Conventional in-car manual interfaces require drivers to take their hands from the wheel, look away from the road and press a button. Unlike these manual interfaces, speech interaction allows drivers to keep "their hands on the wheel and eyes on the road" so they present important benefits for them to be used in vehicle scenarios ([1], [2]). However, in spite of these benefits, previous contributions have pointed out possible situations where in-vehicle speech interaction can entail dangerous distractions to the driver. Distractions may appear due to several reasons, but mainly because of speech recognition errors, caused by: environmental factors (such as noise), intrinsic user factors (driver's mood state), task variables (such as workload) or even by an inappropriate design of the interaction flow. For example, the work presented in [3] shows that when an interaction is too complex or drivers are emotionally involved, important interferences with the driving primary task arise. In the same way, [4]

illustrates that driving is a rather special context where speech recognition errors are likely to appear, increasing drivers mental load.

Consequently, although the maturity of speech technologies is nowadays widely accepted for many applications, their use for in-vehicle interactive systems still requires further research efforts to make them more robust. Currently, the two main research fields intend to provide more efficient and robust speech interfaces are: multimodality and context-awareness. Through a proper combination of speech interaction with other input and output modalities (i.e. buttons, screen messages, haptics etc.) driver's interaction can be made more robust (for example introducing the Push-to-Talk button to avoid false recognitions), more simple (using complementary information in screens such as icons or short messages) and thus with lower workload and higher safety. Also the increasing number of different sensors, corresponding to an increasing number of different sources of information, can be used to identify particular driver's contexts or situations that once more can provide a more efficient and robust interaction. Thus, for instance, by considering the noise level from an acoustic sensor the HMI interface could decide not to use the speech recognition input; or based on the driving behaviour inferred from several sensors (steering wheel, road, weather,…) it could decide to suspend the interaction with the driver and display a warning message.

Following the previous description, the in-car interaction scenario must be considered not as a simple "local" driver-system interface, but, as illustrated in Figure 1, as a more general one where both different driver interaction modality-devices (speech–microphones and loudspeakers; vision–displays; haptic–knobs, buttons, touch screen; etc.) and sensors (generally accessed through CAN bus) are involved. Therefore, for the global interactive scenario in Figure 1, it can be seen that the corresponding in-car Human-Machine Interaction (HMI) management is, consequently, becoming a complex task too.

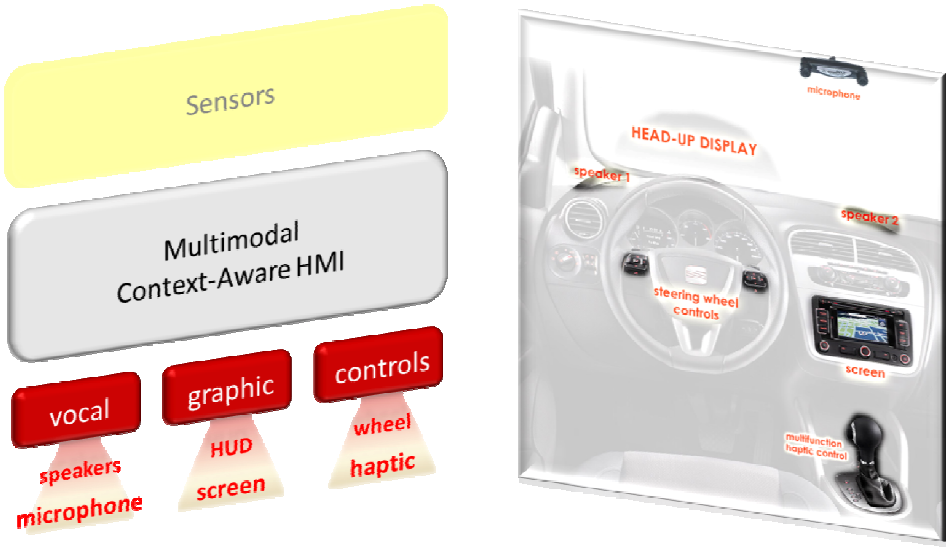

Figure 1 – Global in-car Multimodal and Context-Aware interactive scenario

HMI interaction management can be generally modelled as set of possible states, in which different information is presented to the user, while the transition onto the next state is based on the actual user's input information. Spoken dialogue is a broad HMI research area where two main different approaches are being developed [5]: deterministic, based on rules for

interpreting each user input and updating the interaction state (VoiceXML is important industrial reference [6]), or stochastic, where state transition probabilities are estimated using machine learning techniques. Both spoken dialogue [7] and its extensions to multimodal interaction are being active research areas in driver-car interaction ([8] [9]). The complexity of designing on-board multimodal interfaces has been addressed in [10], and their capabilities for providing more robust error management in [11]. The TALK EU Project [9] was focused on several in-car multimodal systems extending different existing spoken dialogue approaches [12]. Modelling and using context information for smart applications and services is also a recent research area, while integrating both multimodal interaction and context for in-vehicle applications has also been addressed. To give some reference works: in [13] a multimodal interaction is combined with personal driver information (daily schedule, and the environment), while the challenges of HMI design over an OSGi framework for managing the interaction between different car components are discussed in [14]. A common approach when modelling context for vehicle applications, as is detailed in [15], is to consider three independents domains: driver, vehicle and environment. This information is not only provided to applications for querying, but also for adapting their behaviour to the specific context through a Multimodal Interaction Manager.

In this work, based on some of these previous research and technologies we present an XML-based design and development environment for HMI multimodal context-aware. We also present how this HMI framework has been implemented in an embedded vehicle OSGi platform. Our approach makes use of the general process control mechanism provided by well-known state-chart machines. In particular, we rely on the possibilities of the W3C standard, State Chart eXtensible Markup Language, SCXML [16], which provides a generic event-based state-machine execution environment based on Harel statecharts [17]. SCXML is a candidate for control language within the W3C VoiceXML 3.0 [18] (under development) which, for digital telephony services, already has an anticipated development in CCXML 2.0, a very useful language for combining Call Control and current VoiceXML 2.1. SCXML is also being proposed as the multimodal authoring language under development by the W3C Multimodal Interaction working group [19]. Therefore, in this work we anticipate some of these developments and present the use of current SCXML for the design of in-vehicle multimodal interactions. Additionally, we also propose to exploit the modelling and processing capabilities of SCXML for identifying specific situations and contexts (driver, vehicle and environment) to enrich and enhance multimodal HMI interactions. The final aim is a flexible SCXML-based approach for the design of a wide range of multimodal context-aware HMI in-vehicle interfaces. As it was said before, we will also discuss how the resulting HMI applications have been implemented in an automotive OSGI service platform.

The rest of the paper is organized as follows: Section 2 presents SCXML as a technology to implement multimodal HMI applications. In Section 3, we describe how contextual information provided by sensors may be used to adapt the control flow of an application using SCXML, while section 4 presents our OSGi implementation. Finally, conclusions and future work are discussed in Section 5.

2 SCXML for Multimodal HMI

To introduction to SCXML in a simple way let us consider a multimodal interaction flow as a finite sequence of states (and consequently a finite number of transitions), so that a complete interaction will be the result of concatenating consecutive interaction turns in a proper way. Using SCXML, each state represents a different interaction situation that is reached through the recognition of events coming from users. The SCXML example below describes a use case in which only the control flow of a phone call application is implemented.

```
<state id="selectAction">
  <invoke . . . />
  <transition event="action.selectContact" target="selectContact"/>   …
  <transition event="action.selectNumber" target="selectNumber"/>
  <transition event="action.addContact" target="addContact"/>
</state>
<state id="selectNumber">
  <invoke . . . />
  <transition event="action.call" target="phoneCall"/>
  <transition event="action.sendSMS" target="sendSMS"/>
</state>
<state id="selectContact"/>
  …
<state id="addContact"/>
  …
<state id="phoneCall"/>
  …
<state id="sendSMS"/>
  …
```

In this example, the "*selectAction*" state first presents the user a list of available actions, this is done using the `<invoke>` tag that executes, asynchronously, an external process for managing multimodal user's input/output (as it will be explained later). Then the user can select a specific action, and subsequently the invoked process generates the corresponding event for this action: "*action.selectContact*" if the user selects a contact from a contacts list; "*action.selectNumber*" if the user provides a phone number; or "*action.addContact*" if the user adds a new contact to the list. As it is shown in the example, if a phone number is chosen, the SCXML control receives an *action.selectNumber* event and enters the "*selectNumber*" state where the user is asked (again using `<invoke>` tag) whether he/she wants to do a phone call or send a SMS to this particular number. Also in this same state, the transition to *phoneCall* or *sendSMS* states are specified to manage the possible events received from the invoked process (*action.call* or *action.sendSMS*).

Besides the control flow, SCXML can invoke external processes using the `<invoke>` tag. In that way a process can be invoked both to present (output) information to the user and/or to collect (input) information from the user. So, considering a speech-based interface, a VoiceXML interpreter could be invoked (as it is being proposed in VoiceXML 3.0 initiative [18]) to utter a sentence to the user (for example using Text-to-Speech (TTS)) and to recognize a voice command through Automatic Speech Recognition (ASR). In our approach, as VoiceXML 3.0 is still under development [18], we have followed a simpler, but flexible, solution just allowing the dialogue designer to use an atomic order combining speech synthesis (TTS) followed by ASR. This atomic order needs to be specified in an XML document where the text to be prompted is defined together with the specific ASR recognition grammar (we use JavaSpeech Grammar format [20]). The invoked interpreter, in charge of processing this XML voice interaction file, will generate different events depending on what grammar rules are recognized, so that the SCXML control flow could progress.

Bearing that in mind, apart from speech another modality or combination of modalities can help in providing a more efficient and safe interaction with the driver. The `invoke` process in our implementation also provides support for user's inputs/outputs using different modalities. Once again, these modalities are defined using external XML definition files, where both the particular elements of the modality and the possible events to be derived from user actions are specified. For instance, for a graphic modality different elements as buttons, icons, text-boxes, or checkboxes could be defined as well as the user's events associated to each element.

In the example below, when the state machine enters the *selectAction* state, taking advantage of SCXML properties, it evaluates a condition (not specified in this example, but considered later) to decide which modality is the most suitable to present information to the user. Thus, if the condition is met, the state machine invokes an external process which will generate a TTS

prompt and upload an ASR recognition order according to the grammar defined in the `vocalInterface.xml` file. Otherwise, a tactile Screen controller is invoked to display the graphic information and collect those driver's events defined in `graphicInterface.xml`.

```
<state id="selectAction">
   <if cond=" . . . ">
      <invoke targettype="ASR+TTS" src="'vocalInterface.xml'"/>
   </if>
   <else>
      <invoke targettype="tactileScreen" src="'graphicInterface.xml'"/>
   </else>
  <transition . . . />
</state>
```

3 SCXML for adapting applications to context

In the driving environment a great variety of heterogeneous information provided by sensors and other external sources is available. In the same way as the application flow makes use of the user events and the external information provided by modality controllers, when an external process is invoked, the sensor data as well as the other external sources may be processed to adapt the application behavior to context situations.

Following the example introduced in the previous section, while the control flow of an application remains in the "*selectAction*" state, an event from a driver state sensor could arise indicating the user is drowsy. This event could be handled by the control flow process in the same ways as user events. According to the suggestions from Nishimoto's studies addressed in [21], the application flow should be interrupted (enters in the corresponding idle state). Once the dangerous situation disappears (as the "*normal.situation*" event arrives) the state machine shall go back to the previous state (the *history* element is a SCXML element which allows saving the identifier for last state before the last transition). The next snippet represents such situation:

```
<state id="selectAction">
  <invoke . . . />
  <transition event="action.selectContact" target="selectContact"/>   …
  <transition event="action.selectNumber" target="selectNumber"/>
  <transition event="action.addContact" target="addContact"/>
  <transition event="drowsy.situation" target="interrumptInteraction"/>
</state>
<state id="selectNumber">
  <transition event="action.call" target="phoneCall"/>
  <transition event="action.sendSMS" target="sendSMS"/>
</state>
<state id="selectContact"/>
<state id="addContact"/>
<state id="phoneCall"/>
<state id="sendSMS"/>
<state id="interrumptInteraction">
  <transition event="normal.situation" target="history"/>
</state>
```

When SCXML invokes an external process, sometimes it is necessary not only to receive an event but also to collect some additional information from the invoked process, as, for example, ASR recognition results, sensor measurements or application specific external data (i.e Web data). SCXML introduces a `<datamodel>` element which encapsulates any number of `<data>` elements defining variables, each one containing specific information in XML format. This data can be fetched from an external source or specified in-line and can be used to both evaluate guard conditions when a transition in the machine is required or to store data when a new state is entered. Thus, for example, if the user is driving while the vehicle windows are open, a noisy situation identified by sensors updates the corresponding field in a SCXML datamodel. Therefore, when the control flow enters "*selectAction*" state, it will

examine the situation (stored in the datamodel) and decide that the information must be presented through the tactile screen as the ASR will be prone to errors. The next fragment of code below describes this specific case.

```
<state id="selectAction">
   <if cond="Data(situation) eq 'noisy'">
      <invoke targettype="ASR+TTS" src="'vocalInterface.xml'"/>
   </if>
   <else>
      <invoke targettype="tactileScreen" src="'graphicInterface.xml'"/>
   </else>
   <transition event="action.selectContact" target="selectContact"/>
   <transition event="action.selectNumber" target="selectNumber"/>
   <transition event="action.addContact" target="addContact"/>
 </state>
```

4 OSGi Implementation

The Open Service Gateway Initiative is a consortium of technology to create open specifications for facilitating the interoperability of applications and services over a variety of networked devices in homes, cars and other environments. OSGi is a service platform for Java programming language that allows applications to be developed from small, reusable and collaborative components named as bundles. So far, there are several efforts to adopt OSGi technology as a standard framework for telematic in-vehicle devices carried out by the OSGi Vehicle Expert Group [22].

Our SCXML-based in-car HMI framework has been implemented according to the 3.4 release of the OSGi platform [23] (Figure 2). Firstly, we have implemented the HMI interface as a separate bundle including the SCXML engine provided by the Apache *Commons* SCXML implementation [24]. The control flow of an application may subscribe to specific sensors on which it is interested using the *EventAdmin* OSGi Service. *EventAdmin* provides a standard way for working with events in the OSGi Environment using the publish/subscribe model.

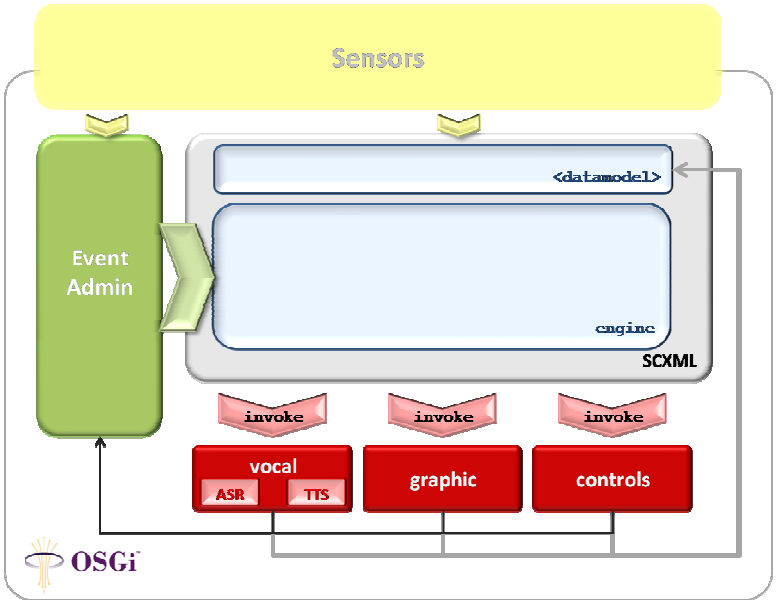

Figure 2 – Implementation of the SCXML-based HMI in an OSGi platform

For invoking processes from the SCXML control flow we have also implemented several independent *bundles*. The speech technologies (ASR and TTS) for the vocal interface *bundle* have been provided by Telefónica R&D. So, when the vocal interface is invoked for TTS and

ASR, once recognition results are available the interface *bundle* shall send a message to the *EventAdmin* notifying that a recognition process has taken place (the *EventAdmin* will trigger this same event to the SCXML engine), and, if necessary, this information will be stored in the SCXML datamodel so that the HMI interface may use this information in the future. In a very similar way, other components (graphic interface, sensors, etc.) were also implemented as independent *bundles* that can notify events and submit data to SCXML datamodel. Therefore, full multimodal and context-aware capabilities are provided to the in-vehicle HMI.

The described architecture was designed in the course of our research activities within MARTA (Mobility for Advanced Transport Networks; www.cenitmarta.org), a Spanish, public-funded project, where several context-aware interactive applications are being designed and implemented for several in-vehicle scenarios.

5 Conclusions and Future Work

In this paper we have presented a HMI framework for improving the design of in-vehicle speech interactive applications using multimodal and context-ware information. Trying to anticipate and extend current W3C initiatives towards advanced voice and multimodal interactive systems, we have proposed the use of a simple and flexible design framework based on W3C SCXML language. We have discussed how SCXML provides a general process control mechanism suitable for combining basic speech interaction (TTS/ASR) with other modalities as well as with other sources of information from sensors available in the vehicle. An automotive platform on top of the OSGi framework has also been presented as a suitable technological platform for enabling events management and data exchange, both demanded for our HMI framework. The resulting architecture is currently being used and tested for the design of several in-vehicle interactive applications. Therefore future research will address particular needs that will demand specific applications and the limitations imposed by state-chart schemes, i.e. the difficulty in both managing and designing, particularly as no graphical development environment is yet available for SCXML.

Acknowledgements

The reserach described in this paper has been done within the framework of the project MARTA (Mobility for Advanced Transport Networks) funded by the *Centro para el Desarrollo Tecnológico Industrial* (CDTI) for the 3rd CENIT Programme, as a part of the INGENIO 2010 initiative of the Spanish Government.

References

[1] Graham, R. and Carter, C.: Comparison of speech input and manual control of in-car devices while on the move. In: Personal and Ubiquitous Computing 4(2): 155-164 (2000)
[2] Lee, J.D., Caven, B., Haake, S. and Brown, L.T.: Speech-based interaction with in-vehicle computers: The effect of speech-based e-mail on drivers' attention to the roadway. In: Human Factors 43(4): 631. (2001)
[3] Vollrath, M.: Speech and driving – Solution or problem?. In: Intelligent Transport Systems, IET 1 (2): 89-94. (2007)
[4] Kun, A., Paek, T. And Medenica, Z. The Effect of Speech Interface Accuracy on Driving Performance. In: Interspeech: 27-31 (2007)
[5] Paek, T., Pieraccini, R., Automating spoken dialogue management design using machine learning: An industry perspective, Speech Communication, Vol. 50, 2008, pp 716-729
[6] W3C: Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C Recommendation 16 http://www.w3.org/TR/voicexml20/ March (2004).
[7] Lawrence C., Fuliang W., Rohit M., Harry B., Badri R., Hua C., Hauke S., Danilo M.,

Ben B., Heather P., Tobias S., Brian L., Joyce C., Stanley P., Liz S., Carsten B., Developing a Conversational In-Car Dialog System, in Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT), Rochester, NY, April 2007.

[8] Pieraccini R., Krishna Dayanidhi K., Bloom J., Dahan J., Phillips M., A Multimodal Conversational Interface for a Concept Vehicle, Communications of the ACM archive, Special Issue: Multimodal interfaces that flex, adapt, and persist table of contents, vol. 47 ,Issue 1, pp. 47-49 (January 2004)

[9] TALK EU project "Talk and Look: Tools for Ambient Linguistic Knowledge" http://www.talk-project.org/

[10] Sinha A. K. y Landay J.A. (2003) Embarking on multimodal interface design, Proceedings of the 4th IEEE International Conference on Multimodal Interfaces, p.355, October 14-16, 2002

[11] McGlaun G., Althoff F., Lang M., and Rigoll G., Towards Multimodal Error Management: Experimental Evaluation of User Strategies in Event of Faulty Application Behavior in Automotive Environments, 7th World Multi-Conference on Systems, Cybernetics, and Informatics, Orlando, FL, USA, 2003

[12] Becker T.,, Poller P., Schehl J., Blaylock N., Gerstenberger C., and Kruijff-Korbayová I., The SAMMIE system: Multimodal in-car dialogue. In Proceedings of COLING/ACL 2006, Sydney, Australia, July 15-21 2006.

[13] Siewiorek D., Smailagic A., Hornyak M., Multimodal Contextual Car-Driver Interface, Proceedings of the 4th IEEE International Conference on Multimodal Interfaces, p.367, October 14-16, 2002.

[14] Nelson E., Defining Interfaces as Services in Embedded Vehicle Software, Research and Advanced Engineering, Ford Motor Company, January 2004, Automotive Software Workshop San Diego.

[15] Amditis, A., Kussmann, H., Polynchronopoulos, A., Engström, J., and Andreone, L.: System Architecture for integrated adaptive HMI solutions. In: Intelligent Vehicles Symposium. Tokyo, Japan (2006)

[16] W3C: State Chart XML (SCXML): State Machine Notation for Control Abstraction, http://www.w3.org/TR/scxml/, W3C Working Draft 13 May (2010)

[17] Harel, D. (1987). StateCharts: A visual Formalism for Complex Systems. In: Science of Computer Programming 8, North-Holland.

[18] W3C: Voice Extensible Markup Language (VoiceXML) 3.0, W3C Working Draft 17 June 2010 http://www.w3.org/TR/2010/WD-voicexml30-20100617/

[19] W3C: Multimodal Interaction Activity, http://www.w3.org/2002/mmi/

[20] JAVA: Grammar Format Specification, Version 1.0 - October 26, 1998, http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/

[21] Nishimoto, T, Shioya, M., Takahashi, J., and Daigo, H.: A study on Dialogue Management Principles Corresponding to the Driver's Workload. Advances for In-Vehicle and Mobile Systems:251 -264. (2007)

[22] OSGi Alliance, VEG - Vehicle Expert Group, http://www.osgi.org/VEG/HomePage

[23] OSGi Alliance, OSGi Service Platform Release 3, The Netherlands, March 2003, http://www.osgi.org/Download/File?url=/download/r3/r3.book.pdf

[24] Apache Commons, Commons SCXML, http://commons.apache.org/scxml/