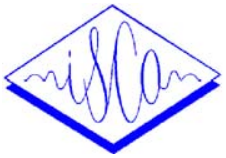
ISCA Archive
http://www.isca-speech.org/archive



INTERSPEECH 2004 - ICSLP
8th International Conference on Spoken Language Processing
ICC Jeju, Jeju Island, Korea
October 4-8, 2004

Modelling Generic Dialog Applications for Embedded Systems

Stefan W. Hamerich, Gerhard Hanrieder

TEMIC Speech Dialog Systems
Ulm – Germany
{stefan.hamerich|gerhard.hanrieder}@temic-sds.com

10.21437/Interspeech.2004-127

Abstract

This paper presents an industrial point of view regarding dialog description languages for dialog applications on embedded devices. We present GDML, a dialog description language developed for embedding dialogs in speech control systems in cars. Furthermore we present the environment used for developing speech dialog applications with GDML. As GDML is already used in existent products for years, we have collected a lot of experiences with it. This enables us, since we also worked with VoiceXML finally to compare both approaches.

1. Introduction

As a leading manufacturer of speech control systems for the automotive market, Temic Speech Dialog Systems developed an XML-based dialog description language called GDML (Generic Dialog Modelling Language).

Speech control systems in cars are embedded solutions, provided either as a separate hardware box or integrated into infotainment systems running under a variety of real-time operating systems. Typically, such embedded systems are characterised by limited resources (memory is still a cost factor). Besides that, even the most complex dialogs running on these systems are still closed applications whose primary task is to control on-board devices (telephone, tuner, CD player, navigation system, etc.). That is, we do not have the need to access data over the web dynamically. Instead, a flexible easy-to-use interface is needed between the speech dialog and the application program controlling external devices. This also holds for the interface to the speech recogniser, where we have the need to control advanced features. Additionally for rapid development of applications in different languages it is needed to port the dialog scripts easy from one language to another. Furthermore a modular structure of the development environment is needed. To sum it up, the following key requirements for dialog management on embedded systems have been identified, which finally lead to the development of GDML:

- Flexible interface able to communicate with every external component,
- Interpreter using very small amount of memory,
- Availability of advanced recogniser features in the dialog description,
- Language independent dialog description.

This paper is organised as follows. First we present the architectural model of GDML, then its features and give a code example. Afterwards the development process of GDML applications is illustrated. Finally we compare GDML with VoiceXML.

2. Generic Dialog Modelling Language

GDML is a XML-based dialog description language. It was designed based on the experiences we gained in the EC funded research projects ACCeSS [1] and IDAS [2]. First ideas of GDML were presented in [3]. Product development of GDML applications started in 2001, and the first speech dialog systems running GDML applications are available since 2003 (speech control for Audi A8 and Mercedes-Benz E-class).

GDML is a compiled language to have optimised binary code through offline compilation. This is needed to meet the memory restrictions that are often imposed by target systems. The compiled dialog is processed by a generic dialog manager (GDM), which is based on ideas first presented in [4]. The GDM works as well in embedded as on desktop environments. Figure 1 gives an overview of the architectural model as assumed for the processing of GDML applications.

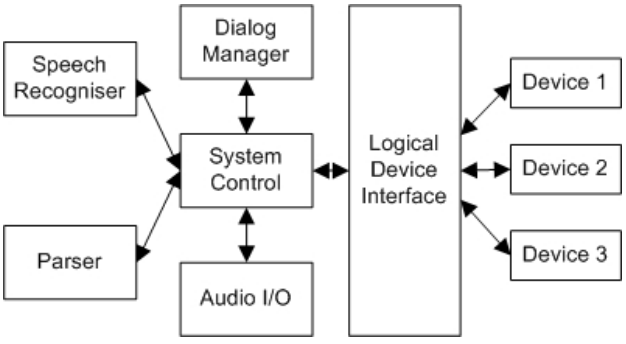


Figure 1: Architectural Model of GDML.

In the architectural model the dialog manager is a separate system component, just like speech recognition and syn-

thesis components. Whereas traditional systems often use only one main component which implements system control tasks (e.g. communicating with external databases) as well as the application-specific dialog control, our model aims towards a modular approach that strictly separates these two functionalities. The advantage of this approach is that the application dialog itself can be implemented rather independently from the specific target platform. This is the benefit of the logical device interface, which abstracts from the devices itself and allows platform-independent interactions with external devices by passing abstract action requirements to the system control. Furthermore this strict separation of the application-specific dialog flow from generic processing tasks supports rapid development of new applications.

2.1. Features of GDML

In this section, some of the important features of GDML are described in more detail.

2.1.1. Explicity

GDML was designed as an explicit language, which means, that every logic of a dialog script is to be contained in the GDML code itself. The advantage of this feature is, that everything is to be controlled by the dialog, what means that several options could be set separately in different dialog steps.

2.1.2. Typed Variables

Furthermore GDML requires typed variables. This is an adopted measure due to the memory requirements on embedded devices. The compiler could be more efficient, when the type of variables is known.

2.1.3. Language Independent

Additionally GDML allows to write language independent dialogs. This was realised by including prompt concepts, which means the dialog flow only contains abstract representations of what shall be said by the system. The definition how a concept is to be realised, i.e. the wording, is contained in separate language-specific tables.

2.1.4. Extended Application Interface

An extended application interface was realised in GDML, establishing the logical device interface, as described above. The function calls for this interface in GDML are called system calls. The interface itself is open which allows external components to be called from the dialog. The call may have parameters and always needs a return code. The functionality of the call may be in any external hardware or software component, which must be connected to the dialog control.

2.1.5. Advanced Recogniser Interface

TEMIC's recogniser, the StarRec DSR allows the setting of several parameters, e.g. initial timeout, maximum timeout, maximum number of results to be returned, recognition to file, and recognition from file. All these options could be set in the GDML dialog script using the system call interface, which allows a very flexible usage of the DSR. It even allows in each dialog step to have different settings of the recogniser.

2.2. GDML code example

To give a better impression, of how GDML dialogs look like, we like to give a short example. The presented dialog asks the user, if he wants to hear the current date, if this is confirmed, the date is uttered, else the dialog ends. Execution of the dialog starts at the tag named `dialog`. Each dialog is divided into steps. The first defined step is the first to execute, afterwards the next steps explicitly have to be assigned. The functionality `get_date` is implemented as an external system call.

```
<?xml version="1.0"?>
<gdml version="1.0">

<!-- Declaration Block -->
<def_events events="PTT, ABORT"/>
<def_system_call name="getDate"
  results="d:string, m:string, y:string"/>
<def_enum name="recog_errors"
   elements="RECOG_OK,RECOG_ERROR"/>
<def_prompt concept="start"/>
<def_prompt concept="want_date"/>
<def_prompt concept="say_date"
  params="m:string, d:string, y:string"/>
<def_prompt concept="abort"/>
<def_prompt concept="goodbye"/>
<var decl="parseres:recres_list"/>
<var decl="res_err:recog_errors"/>
<var decl="day:string"/>
<var decl="month:string"/>
<var decl="year:string"/>

<!-- Global Event Handler -->
<catch event="ABORT">
  <prompt concept="abort"/>
  <restart/>
</catch>

<!-- Main Entry Point -->
<dialog name="main">

  <step name="first">
    <var decl="cmdval:string"/>
    <wait events="PTT"/>
    <prompt concept="start"/>
    <prompt concept="want_date"/>
    <recog grammar="'date'" rules="'yes_no'"
      results="parseres, res_err"
      init_timeout="10000"/>
    <getrecog in="parseres[0]" feature="cmd"
       result="cmdval"/>
    <next cond="cmdval eq 'yes'" name="date"/>
    <next cond="cmdval eq 'no'" name="end"/>
  </step>
```

```
  <step name="get_date">
    <system_call name="getDate"
      results="day, month, year"/>
    <prompt concept="say_date"
      params="month, day, year"/>
    <next name="end"/>
  </step>

  <step name="end">
    <prompt concept="goodbye"/>
    <exit/>
  </step>

</dialog>

</gdml>
```

The representation of the prompts is done in a separate file, which for English would look like the following:

```
<?xml version="1.0"?>
<output_definitions version="1.0"
    language="en_UK">

  <prompt_body concept="start">
    <tts>welcome to the date dialog</tts>
  </prompt_body>

  ...

  <prompt_body concept="goodbye">
    <tts>good bye</tts>
  </prompt_body>

</output_definitions>
```

2.3. Developing Applications in GDML

The development process of GDML applications is supported by different tools. Grammars and dialog script are compiled to provide optimised binary code. Compiled dialogs then are processed by the dialog manager, which on the runtime system is controlled by the system control. For interactive testing of dialog applications, the DDS (Dialog Development Studio) is replacing the system control. See Figure 2 for the tool chain used for developing GDML dialogs.

2.4. Usage of GDML

Since its formulation GDML has been used in all of our products as dialog description language. This means currently GDML applications can be found in the Linguatronic of Mercedes-Benz cars (refer to [5]) and in speech control interfaces of other car manufacturers, among them Audi, BMW, Lancia, Porsche and Rolls-Royce.

3. Voice Extensible Markup Language

The VoiceXML specification version 2.0 was published by the Voice Browser Working Group of the W3C [6]. During the last years VoiceXML has been used in a lot of projects and applications (see e.g. [7]).

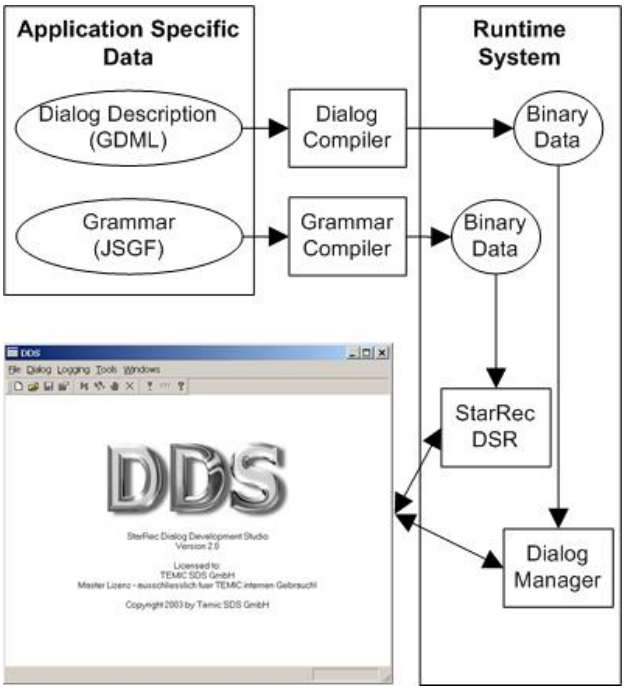


Figure 2: Tool Chain for GDML Dialogs.

TEMIC collected a lot of experiences with VoiceXML in the EC funded research project GEMINI (refer to [8]). Based on these experiences, we make a comparison of GDML and VoiceXML in the next section and name some of the differences between both languages.

4. GDML vs. VoiceXML

In some respects, GDML bears close resemblance to VoiceXML, and was actually influenced by it: at the time when we defined GDML, version 0.9 of VoiceXML was out for discussion. But since VoiceXML did not meet the requirements imposed by our target systems, we decided not to use it.

In this section, we illustrate some of the differences of both languages.

4.1. External Interface – Web Mechanism vs. Function Calls

The current VoiceXML standard allows two methods to access external resources. The first are CGI scripts, which need an HTTP server. The second method bases on ECMAScript, but requires an ECMAScript interpreter. Centralised this leads for both methods to additional requirements for the respective device, because extra memory is needed and the existence of a file system is supposed.

In GDML a system call is defined as an code element of its own. Several system calls can be defined and used. When executing the call, the dialog manager sends a message to the defined external system call handler and gets an answer,

which could be used in the dialog. A call handler may be any external device (e.g. a car's radio or a database) which needs to have a system call interface. Since the external devices do the processing of the system call by themselves, no server component is needed at all. Additionally this works without a file system on the embedded device.

4.2. Recogniser Interface – Implicit vs. Explicit

In VoiceXML the recogniser interface is hidden in the interpreters form interpretation algorithm (FIA). Everytime when a `field` is found in a VoiceXML script, the recogniser is opened by the FIA. This allows for small and concise code but does not allow an advanced recogniser interface.

In GDML every recogniser call explicitly has to be done in the code, leading to a lengthy but precise code. In every recogniser call several options may be set. This explicit interface makes the code appear verbose but gives more freedom to the dialog designer, since in separate dialog steps different recogniser options may be used. Additionally this interface allows the usage of text and voice enrollments.[1] Both are not implemented in VoiceXML.

4.3. Prompts – Inline Formulation vs. External Representation

Prompt concepts in GDML ease the portability of dialogs to different target languages. This concept is missing in VoiceXML as TTS prompts have to be denoted explicitly in the dialog script itself.

4.4. Program Concepts – Light Language vs. Program Language

GDML offers the full functionality of a real programming language like e.g. C, which leads to the offering of several types of loops, lists and array handling. Here is a weakness of VoiceXML, since all higher functionality has to be provided by using ECMAScript or CGI scripts.

It would be good to extend VoiceXML to allow more complex and flexible dialogs, which would appear to be more natural. Especially needed is the introduction of loops, which would allow more complex and flexible dialogs. Additionally it would be important, to widen the context of `subdialog` in VoiceXML to allow subdialogs to be called even in `field` or `filled`.

4.5. Execution – Online Interpretation vs. Offline Compilation

Since dialogs for embedded systems mostly do not have to be dynamic, offline compilation saves a lot of memory and performance. GDML dialogs have to be compiled and may be executed by a small software component. This is completely different for VoiceXML, here a online interpretation is done by the VoiceXML interpreter. The interpreter contains lot of the process logic which is needed to call concepts like `help` during runtime.

[1] A text enrollment means the addition of textual representations to the vocabulary, whereas a voice enrollment stands for the addition of recorded voice fragments to the vocabulary. Both additions could be done while the dialog is running.

5. Conclusion

In this paper we presented the dialog description language GDML. We illustrated its architecture, discussed its features and showed an example dialog. Furthermore the tool chain and the usage of GDML have been presented. Additionally we compared GDML with VoiceXML.

6. References

[1] U. Ehrlich, G. Hanrieder, L. Hitzenberger, P. Heisterkamp, K. Mecklenburg, and P. Regel-Brietzmann, "ACCeSS - Automated Call Center through Speech Understanding System," in *Proceedings EUROSPEECH*, Rhodes, Greece, 1997, pp. 1819 – 1822.

[2] G. Lehtinen, S. Safra, M. Gauger, J.-L. Cochard, B. Kaspar, M. E. Hennecke, J. M. Pardo, R. de Córdoba, R. San-Segundo, A. Tsopanoglou, D. Louloudis, and M. Mantakas, "IDAS: Interactive Directory Assitance Service," in *Workshop on 'Voice Operated Telecom Services'*, ser. COST 249, Ghent, Belgium, 2000, pp. 51 – 54.

[3] M. E. Hennecke and G. Hanrieder, "Easy Configuration of Natural Language Understanding Systems," in *Workshop on 'Voice Operated Telecom Services'*, ser. COST 249, Ghent, Belgium, 2000, pp. 87 – 90.

[4] G. Hanrieder, "Integration of a Mixed-Initiative Dialogue Manager into Commercial IVR Platforms," in *Proceedings IVTTA*, Torino, Italy, 1998, pp. 77–82.

[5] P. Heisterkamp, "Linguatronic – Product-Level Speech System for Mercedes-Benz Cars," in *Proceedings HLT*, San Diego, USA, 2001.

[6] S. McGlashan, D. C. Burnett, J. Carter, P. Danielsen, J. Ferrans, A. Hunt, B. Lucas, B. Porter, K. Rehor, and S. Tryphonas, "Voice Extensible Markup Language (VoiceXML) Version 2.0," W3C Recommendation, www.w3.org/TR/voicexml20, 2004.

[7] C. Bennett, A. FontLlitjós, S. Shriver, A. Rudnicky, and A. W. Black, "Building VoiceXML-Based Applications," in *Proceedings ICSLP*, Denver, USA, 2002, pp. 2245–2248.

[8] S. W. Hamerich, V. Schubert, V. Schless, R. de Córdoba, J. M. Pardo, L. F. d'Haro, B. Kladis, O. Kocsis, and S. Igel, "Semi-Automatic Generation of Dialogue Applications in the GEMINI Project," in *SIGdial Workshop on Discourse and Dialogue*, Cambridge, USA, 2004, pp. 31–34.