# XML-Based Dialogue Descriptions in the GEMINI Project*

Stefan W. Hamerich[1], Yu-Fang H. Wang[1], Volker Schubert[1†],
Volker Schless[1], and Stefan Igel[2]

[1]TEMIC Speech Dialog Systems
Ulm – Germany
{stefan.hamerich|helena.wang|volker.schubert}@temic-sds.com

[2]Forschungsinstitut für anwendungsorientierte Wissensverarbeitung (FAW)
Ulm – Germany
sigel@faw.uni-ulm.de

**Abstract:** GEMINI (Generic Environment for Multilingual Interactive Natural Interfaces) is an EC funded research project. The goal of GEMINI is to provide a flexible platform for the generation of dialogue applications, able to produce multi-modal and multilingual dialogue interfaces to databases with a minimum of human effort. This platform consists of a toolset supporting all steps of application generation by producing a set of model files each describing a different aspect of the dialogue application. These models are expressed in our newly designed XML-based description language called GDialogXML (Gemini Dialog XML). It is an abstract and object-oriented dialogue description language which is flexible enough to account for modality independent and modality specific features of the final representation languages, which are VoiceXML for the speech modality and xHTML for the web modality.

In this paper we introduce the research project GEMINI and present the dialogue modelling language GDialogXML. Furthermore we discuss the limitations of VoiceXML, which we found out while developing an automatic VoiceXML generator using GDialogXML scripts as input.

## 1 Introduction

VoiceXML is existing for several years and has emerged to the standard description language for spoken dialogue applications. On the other hand (x)HTML is the undisputed standard description language for web-based applications. Though several approaches have been undertaken, no description language is yet able to cover both modalities.
Therefore a complete new abstract dialogue description language was developed for the

*This work was partly supported by the European Commission's Information Society Technologies Programme under contract no. IST-2001-32343. The authors are solely responsible for the contents of this publication.

†formerly with FAW

research project GEMINI, which covers multi-modal and modality specific functionalities. During the project's runtime an application generation platform (AGP) was designed, which allows automatic generation of VoiceXML and xHTML from our newly defined language. The underlying core concept here is to use a modality and language independent representation of a dialogue description, and to extend this generic dialogue form by modality-specific aspects.

This paper is organised as follows: After introducing the objectives of GEMINI, we briefly give an overview of the application generation platform built during the project. Then the concepts of GDialogXML are introduced. Afterwards, we discuss some of the limitations of VoiceXML we found out during the development of the speech script generator of the AGP.

## 2 Project GEMINI

GEMINI[1] is funded within the European Union's Fifth RTD Framework Programme for two years having started in April 2002. Its consortium consists of the following partners: Knowledge S.A. (Greece) as coordinator, University of Patras – Wire Communications Laboratory (Greece), TEMIC Speech Dialog Systems (Germany), Universidad Politecnica de Madrid – Grupo de Tecnología del Habla (Spain), Forschungsinstitut für anwendungsorientierte Wissensverarbeitung (Germany) and Egnatia Bank S.A. (Greece).

### 2.1 Project Goals

GEMINI has two main objectives. First the development of a flexible platform able to produce user-friendly, high quality, multi-modal and multilingual dialogue interfaces to a wide area of data sources with a reduction of human effort to entering parameters while being guided by a graphical user interface. Second, the demonstration of the platform's efficiency, effectiveness, and adaptability to new applications and languages by developing two pilot applications. In this paper the main focus was put on the first goal, please refer to [WHS03] for more aspects of the GEMINI project.
The core idea of GEMINI is that given (1) a database, (2) a description of the database structure, (3) a connector to access the data, and (4) a list of the kinds of requests the user may make, the system should be able to semi-automatically generate the necessary dialogue scripts to run the service.
During specification and implementation, the project exploits experience gained from previous EC-projects (see e.g. [EHH+97, BLM+98, LSG+00]).

[1] See `www.gemini-project.org` for the project homepage.

## 2.2 The Application Generation Platform (AGP)

The AGP is the GEMINI approach of a toolset to generate multi-modal and multilingual dialogue applications. The platform was designed as a three-level architecture with increasing specificity regarding the modalities. In Figure 1 the architecture of the AGP is illustrated. A system control unit serves as a central user interface to access the different tools and assistants.

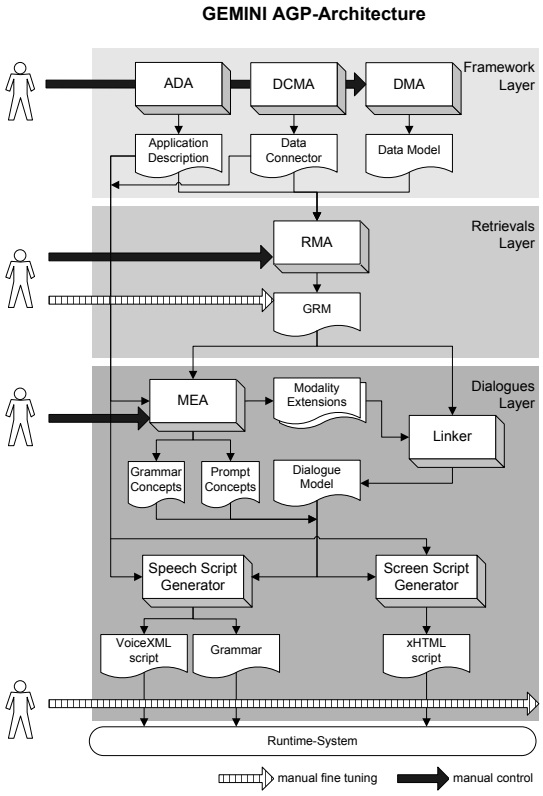


Figure 1: Schematic view of the AGP architecture.

Processing starts in the framework layer. It includes the application description assistant (ADA), the data connector modelling assistant (DCMA) and the data modelling assistant (DMA) which all create XML-based output files. As indicated by the black arrow in the upper left corner of Figure 1, all assistants are controlled interactively. First a system designer has to provide the application description, containing descriptions of (1) the modalities, for which the AGP should generate dialogue scripts, (2) the languages for which the dialogues should be available, (3) a description of the system behaviour (e.g.

system-driven or mixed-initiative) and (4) error handling parameters. He is supported by the ADA, that writes the information into the application description model.
The DCMA helps creating APIs and implementation references for application specific data access functions. E.g. the function `getPersonByName` (that may be used to extract person details by providing the name of the respective person) will be defined and implemented here and can be used in the runtime system without any knowledge of the existing database. The description is stored as data connector model.
The DMA helps creating the data model, which consists of class descriptions of the relevant contents of the database. Also, the attributes and elementary types of the data are specified here and written to the data model.

The second layer is called retrieval layer, as it generates a generic retrieval model (GRM) out of the application description and the data model. This layer mainly consists of the retrieval modelling assistant (RMA) and is modality and language independent. Therefore no language or modality specific data is included here.
The designer controls the RMA, which provides a user-friendly UI to automate the design process. In this step, the RMA assigns the basic dialogue acts to parts of the application description and the data model. The resulting output is the GRM, which consists of the modality and language independent parts of a dialogue, i.e. the application flow.
As indicated by the dashed arrow, it may be necessary to do some manual fine tuning on the GRM, as the complexity of the RMA depends on the application and may be rather high. Furthermore, there are often several ways to implement the application and the designer has to choose the most appropriate solution.

The third and final layer of the AGP is called the dialogue layer. This layer is modality and language dependent since here the modality extension produced by the modality extension assistant (MEA) is provided. The modality extension contains the input and output behaviour of an application and is designed for a specific modality[2].
The GRM is enriched by the modality extensions in the Linker. The resulting model is called dialogue model, which is processed by the speech script generator and/or the screen script generator depending on the selected modalities in the application description. The generators automatically generate dialogue scripts for the runtime interpreters, e.g. in case of the speech modality VoiceXML scripts with some additional CGI scripts are generated[3].

To have the runtime system ready for use, little effort has to be spent on manual fine tuning again. For the speech modality language models and vocabulary must be built as well.

## 3 GDialogXML

As mentioned above, GDialogXML (Gemini Dialog XML) has been developed within the GEMINI project to support the generation of dialogue applications using the toolset of the

[2]The current implementation of the AGP supports the generation of voice (speech modality) and web-based applications (web modality). For the speech modality the modality extension consists of references to grammar and prompt concepts, which are language independent. The definition of the grammars and prompts is done in the concept files for each language of the application.

[3]The grammars are taken from the grammar concepts of the MEA.

AGP. As it is an XML-based description language it can be parsed and processed very easily by the platforms assistants. But this very verbose and explicit language is not well suited for writing dialogues manually.

All model files produced by the assistants mentioned in the previous section are represented in GDialogXML. Therefore, GDialogXML has to provide concepts for modelling data, procedures and data access functions, dialogue flow and user interaction.

**General Concepts** GDialogXML follows an object oriented approach to model data, procedures and dialogues and provides ways to

- define typed constants and variables,
- deal with complex data structures like lists, objects and object references,
- define classes and inheritance of classes,
- model procedures with a body and input and output parameters,
- model dialogues, where a dialogue model is considered to be a special procedure with additional input and/or output behaviour,
- express control structures for iterating and branching,
- perform arithmetic, boolean, and string operations modelled as internal procedures.

GDialogXML is based on concepts rather than on imperative programming. It makes strong usage of the tree-structure of XML documents. So, instead of saying ”if - then - else” we have the concept ”SeqBranching” which contains the branches as subnodes. The main concept categories are (1) commands, (2) values, (3) procedures (including dialogue modules), (4) variables (including constants), and (5) procedure calls. Certain concept instances like procedures and variables have names (given during definition) by which they can be referenced and used (utilising the DOM attribute ”refr”). Others like commands and values are used ”inline”.

Example of a control flow structure (like ”switch” in C):

```
<SeqBranchingByValue>
  <xValue>
    <ValueOfVar><Var refr="answer"/></ValueOfVar>
  </xValue>
  <BranchForValue>  // if "answer" is "yes"
    <xValue>
      <StringC>yes</StringC>
    </xValue>
    <xBody>
      // define some action here
    </xBody>
  </BranchForValue>
  // define more branches here
</SeqBranchingByValue>
```

Example for the definition of a variable:

```
<Var id="answer">
  <xType>
    <Type refr="String"/>
  </xType>
</Var>
```

**Data Modelling** The data model describing the data used in the dialogues is generated by the DMA. The data is modelled as classes with attributes, which can have simple data types like string, integer, boolean or complex types like embedded or referenced objects or lists. GDialogXML supports inheritance from base classes.

**Dialogue Modelling** A central concept of GDialogXML is the definition of dialogues. Within the AGP the dialogues are modelled in two steps: (1) in the RMA the modality independent part of a dialogue is realised, producing the GRM, (2) in the MEA modality specific extensions to the basics dialogue are done.

The final dialogue model is created by the Linker by combining the GRM and the modality extensions. This is done automatically. All dialogue models consist of dialogue modules that call each other. The most important elements of a dialogue are:

- Sections for variable definitions and usage declarations for defining local variables (`<xLocalVars>`), and referencing them, if they are used as input field variables (`<xInputFieldVars>`), parameters (`<xArgumentVars>`) or return values (`<xReturnValueVars>`).
- A section to describe the output behaviour (`<xPresentation>`). Here the output of a dialogue is modelled, e.g. prompts for speech dialogues. The presentation of a dialogue is modality dependent and therefore modelled in the modality extension.
- A section to describe the input behaviour of the dialogue (`<xFilling>`). In this part of the dialogue it is modelled how the dialogue will interact with a user to obtain information. This filling can be done by direct input (e.g. speech recognition) or by calls to subdialogues. This part is only to be found in the modality extension because it is highly modality dependent.
- A section to model the control flow, depending on the calling parameters and the values of input field variables (`<xReaction>`). This part is equivalent to the body of a procedure. The dialogue flow logic is implemented here including calls to procedures and subdialogues and thus resulting in a hierarchy of dialogues and sub-dialogues. The complete dialogue application is modelled as hierarchy of modular dialogue acts, starting with a global dialogue at the root and ending with highly specialised subdialogues. This section is only modelled in the generic definition (GRM). I.e. the over-all dialogue flow is modelled modality independent.

Whereas the control flow of a dialogue is independent of the chosen modality, the input and output behaviour will be different for every modality. The above mentioned sections of a dialogue module belong to the complete dialogue model. The generic module definition (in the GRM) and the modality extension only contain certain sections. Whereas both contain variable definitions,

- `<xReaction>` is only in the generic definition (GRM), not in the modality extension. That is, the over-all dialogue flow is modality independent.
- `<xFilling>` and `<xPresentation>` are only in the modality extension, not in the GRM. That is, the IO behaviour is modality specific[4].

Every dialogue module defined in the GRM must have a matching modality extension, i.e. a module extension. Sometimes it may be necessary to define modality dependent auxiliary dialogues, which can be completely defined in the modality extensions. Note that every control section, `<xReaction>`, `<xFilling>` and `<xPresentation>`, can expand to a call hierarchy of dialogue modules. This does not necessarily mean that the calls are executed sequentially: For certain modalities like mixed-initiative speech and web-pages the calls should be executed simultaneously allowing for parallel filling and presentation. GDialogXML supports concurrent calls. Another point is that it is an objective of dialogue modelling within the AGP to use predefined dialogue modules (kept in libraries) as subdialogues. This is possible since dialogue modules should be bound to data modules, and therefore be application independent. They can be thought of as experts for certain data structures. After linking GRM and the modality extensions the result is a complete but modality dependent description of the dialogue application.

Prompts and grammars are only referenced in GDialogXML leading to completely language independent dialogue descriptions.

## 4 Limitations of VoiceXML

GEMINI aims at a runtime system for the speech modality running on a standardised dialogue description language. The preferred language was VoiceXML mainly because of the high level of standardisation and because of the availability of tools.
However, the AGP should be able to go beyond simple speech applications. To demonstrate this, two major applications were set up for later evaluation.

The dialogue scripts for the runtime system are to be generated automatically from the dialogue models written in GDialogXML. The generators are part of the dialogue layer of the AGP. For the speech modality, this generator was named speech script generator or VoiceXML generator, since the produced dialogue scripts were written in VoiceXML. Due to the fact that the generators work automatically, a generic approach had to be found for

[4]But this is a very tricky point, since filling and presentation might actually be modality-independent, for example, if these parts only use calls to subdialogues and do no user-interaction of their own. There are constructs in GDialogXML that allow the specification of modality-independent call hierarchies, which form a basis to the actual filling and presentation (the concepts `<xFillingCalls>` and `<xPresentationCalls>`).

them. During the design phase for the speech script generator we found out about some limitations of VoiceXML.

In this section we like to sum up the most important problems we encountered and mention some proposals to solve these problems for future releases of VoiceXML.

**Modelling of Returning Dialogue Flow** One severe difficulty in VoiceXML is to model a returning dialogue call. VoiceXML does offer a mechanism to handle it, but unfortunately this is only allowed in a `<form>`. However, we believe that calling subdialogues from within fields or blocks is a straightforward requirement and propose to include this in future versions of VoiceXML.

**Program Logic** Compared to other programming languages, VoiceXML lacks common constructs for program logic, notably loops (e.g. `while` and `for`). Generally the usage of ECMA script is recommended instead. But this solution is useless when operations across fields or dialogues are needed. Therefore we would like to propose the introduction of loops in VoiceXML.

**Dataflow Between VoiceXML Scripts and an External Database** As said in [MBC+03], access to external sources (e.g. a database) is only possible via CGI scripts. This is in fact easy for one direction, namely when transmitting data to a database (typically for a query). Nevertheless problems arise, when data from a database is to be returned into the dialogue flow. Here VoiceXML code has to be generated by a CGI script that contains assignments of string constants to variables.

Importing data could be improved by supporting explicit HTTP requests, extending the possibilities of `<subdialog>` and `<data>`.

**Prompt Concepts** To ease the automatic generation of VoiceXML scripts and simplify the reuse for different languages (localisation) it would be useful to establish the external representation of prompts by introducing prompt concepts.

## 5 Conclusion and Future Work

During the EC funded research project GEMINI with its application generation platform we designed a new abstract dialogue description language called GDialogXML, which is based on XML and covers aspects like multi-modality and multilinguality. During our work we encountered some difficulties while converting GDialogXML dialogues into VoiceXML scripts. We claim that VoiceXML should be less restrictive in its syntax and propose the introduction of loop constructs and returning subdialogue calls e.g. inside blocks.

For the final phase of the GEMINI project, several extensions to the AGP are to come.

Among others a user modelling component, speaker identification, language recognition, and document parsing will be added. Furthermore we are thinking about adding a new modality to the AGP.
Since the evaluation phase of the AGP has started, applications of higher complexity and with more languages are to be generated by the AGP.

## References


[BLM+98] T. Brøndsted, L. B. Larsen, M. Manthey, P. McKevitt, T. Moeslund, and K. G. Olesen. The IntelliMedia Workbench – a generic Environment for multimodal Systems. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Sydney, Australia, 1998.

[EHH+97] U. Ehrlich, G. Hanrieder, L. Hitzenberger, P. Heisterkamp, K. Mecklenburg, and P. Regel-Brietzmann. ACCeSS - Automated Call Center through Speech Understanding System. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1819 – 1822, Rhodes, Greece, 1997.

[LSG+00] 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 Assistance Service. In *Proceedings of the international Workshop 'Voice Operated Telecom Services'*, COST 249, pages 51 – 54, Ghent, Belgium, 2000.

[MBC+03] 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 – Voice Browser Working Group, www.w3.org/TR/voicexml20, February 2003.

[WHS03] Y.-F. H. Wang, S. W. Hamerich, and V. Schless. Multi-Modal and Modality Specific Error Handling in the GEMINI Project. In *Proceedings of the ISCA Tutorial and Research Workshop on 'Error Handling in Spoken Dialogue Systems'*, Chateau-d'Oex-Vaud, Switzerland, 2003.