The GEMINI Platform: Semi-Automatic Generation of Dialogue Applications

Stefan W. Hamerich[1], *Ricardo de Córdoba*[2], *Volker Schless*[1], *Luis F. d'Haro*[2], *Basilis Kladis*[3], *Volker Schubert*[1], *Otilia Kocsis*[3], *Stefan Igel*[4], *and José M. Pardo*[2]

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

[2] Grupo de Tecnología del Habla, Universidad Politécnica de Madrid, Madrid, Spain
{cordoba|lfdharo|pardo}@die.upm.es

[3] Knowledge S.A. (LogicDIS group), Patras, Greece
{bkladis|okocsis}@logicdis.gr

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

Abstract

The EC funded research project GEMINI (Generic Environment for Multilingual Interactive Natural Interfaces) has two main objectives: On the one hand the development and implementation of a platform able to produce user-friendly interactive multilingual and multi-modal dialogue interfaces to databases with a minimum of human effort and on the other hand the demonstration of the platform's efficiency through the development of two different applications using this platform. The platform consists of different assistants that help the user to semi-automatically generate dialogue applications. Its open and modular architecture simplifies the adaptability of generated applications to different use cases.

1. Introduction

In the project GEMINI [1] we exploit experience gained from previous projects (see e.g. [2, 3]) and from real-world use of similar systems, to create a generic platform for the development of user-friendly, natural, high quality, intuitive, platform independent and multi-modal interactive interfaces to a wide area of databases employed by information service providers.

GEMINI's main idea is that, given a database structure and a rough idea of the dialogue flow, the system should be able to semi-automatically generate the necessary dialogue scripts for the service application. In a sense, the information provided to the system corresponds to what a human operator in a call center needs to know in order to perform his job. Within the project we strive to get as close as possible to this ideal.

Specifically, the application generation platform of the GEMINI project contains generic dialogue components available for adaptation to new services and languages. Thus, generation of multilingual and multi-modal interfaces is achieved by incorporating the classes and semantic relations of the database, reducing the development time and facilitating the system's maintenance and transportability to different applications and languages. Furthermore, the platform enables a high degree of personalisation (by incorporating e.g. user modelling and speaker verification).

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.

In relation to other approaches GEMINI's way of setting up new dialogue applications differs in main points:

Compared with the REWARD system from [4] the GEMINI platform allows the generation of dialogues for several modalities and it generates dialogues in standardised description languages (VoiceXML and XHTML).

In [5] a rapid development environment for speech dialogues from online resources was described that composes a database using knowledge from various web applications. Contrary the GEMINI platform requires a filled database and allows the development of speech and web applications from it. So we do not need to extract any knowledge, which makes the GEMINI approach more domain independent.

This paper is organised as follows: First we describe in detail the three layers of the application generation platform (AGP). Afterwards we shortly introduce the two pilot applications that were set up using the GEMINI AGP and conclude our major findings.

2. Application Generation Platform

The main target of the GEMINI project is the development of a platform for generating interactive, multilingual and multi-modal dialogue interfaces to databases with a minimum of cost and human effort. We had several objectives in mind when building the AGP:

- Minimum effort for the designer to build a new dialogue application.
- Strong effort in standardisation to ensure usability, dissemination and guaranteed future of the platform. To achieve this goal, all models generated by the AGP are described in GDialogXML (GEMINI Dialogue XML), which is an object-oriented abstract dialogue modelling language. For a detailed description of GDialogXML refer to [6]. Furthermore all modules of the AGP have been written using Trolltech Qt, a multi-platform (Linux, Windows, Mac, etc.) programming environment, so that the code generated can be executed on any platform with a simple recompilation. And finally, the output of the platform is XHTML scripts for web applications and VoiceXML scripts for speech applications.
- To allow easy reusage and to speed up the development process all models in the AGP may be saved as libraries for future applications.

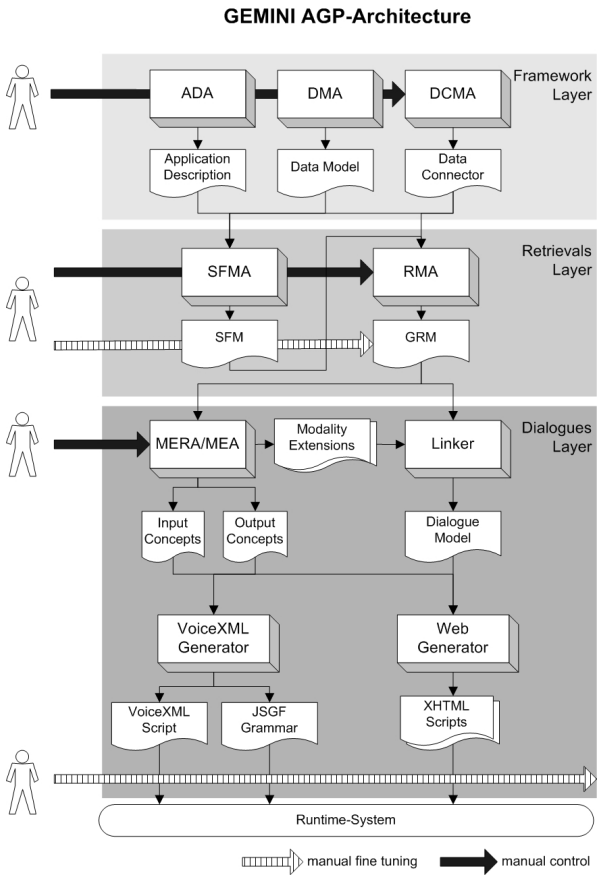

Figure 1: Architectural model of the platform.

All components of the AGP are integrated into one common GUI. This eases the use of the platform and enables the designer to switch back and forward to different tools in case she or he wants to add or modify certain dialogues.

In Figure 1 the architecture of the AGP is illustrated. The whole AGP consists of three layers. These layers are described in more detail in the following sections.

2.1. Framework layer

The framework layer is the first layer of the AGP (refer to Figure 1). It includes the application description assistant (ADA), the data modelling assistant (DMA), and the data connector modelling assistant (DCMA). As indicated by the black arrow in the upper left corner of Figure 1, all assistants are controlled manually.

The designer has to provide the application description, which mainly consists of the modalities for which the AGP should generate dialogue scripts, the languages of the application and settings for error handling. For further information about the error handling capabilities of the AGP refer to [7].

The DMA helps creating the data model, which consists of class descriptions. Each class is characterised by a list of attributes, a description, and a list of base classes (inheriting their attributes). The graphical view of a class, and its attributes, is presented in Figure 2. The attributes may be (a) of atomic type (e.g., string, boolean, float, etc.), (b) complex objects, obtained by embedding or referring to an existing class, or (c) lists of either atomic type items or complex objects.

Data model development is accelerated by using library classes. A new class is created very easily either by copying an existing

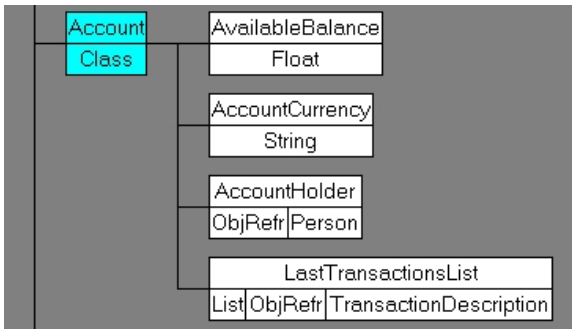

Figure 2: Graphical view of a class and its attributes.

library class, or by using attributes of several classes. The user has only to rename the new class and to set its attributes lists.

Finally the DCMA as the third assistant in the framework layer helps creating APIs and implementation references for application specific data access functions.[1] These functions could then be used in the runtime system without any knowledge of the existing database.

2.2. Retrievals layer

The retrievals layer (shown as the second layer in Figure 1) mainly consists of the state flow modelling assistant (SFMA) and the retrieval modelling assistant (RMA). This layer is modality and language independent, therefore no language or modality specific data is included here.

The designer first uses the SFMA to create the abstract dialogue flow by specifying the high-level states of the dialogue (including information about what slots are asked to the user and which are the following states). To specify the slots, attributes from the data model are offered to speed up the design. It is a very high level definition of the dialogue.

The following assistant is the RMA, which provides a user-friendly interface where the dialogue model is generated. The resulting output of the RMA is called generic retrieval model (GRM), which consists of the modality and language independent parts of a dialogue, which is mainly the application flow. The GRM is modelled in an object-oriented way using GDialogXML and mainly consists of dialogue modules. A dialogue module can call other modules as subdialogues or can jump to another top-level module.

Because the designer may need to develop different kinds of services we provide the following types of dialogues and features (in almost all cases, the designer can drag and drop the needed action or dialogue to complete the design):

Different kinds of actions: The assistant allows the creation, edition or deletion of assignments, conditional actions, loops, switches, calls to other states (dialogues or functions), etc. Each action has different possibilities, for instance: in a loop the designer can select the actions inside it (the condition, the step action etc.).

Automatic dialogues: Several dialogues are automatically created in order to obtain information from the user (called 'DGet dialogue' from now on) and to provide information to the user (called 'DSay dialogue'). These dialogues are based on information from the data model (classes and attributes defined for the service). For each class and attribute we generate a DGet and a DSay dialogue, which include a tag used by the modality extension assistant to know that the prompt to be presented to the user (for DSay) and the grammar used by the recogniser (for DGet) have to be specified.

[1] The implementation of data access functions has to be done outside of the AGP context, since special knowledge about the database itself is needed for this. Thus the AGP is database independent.

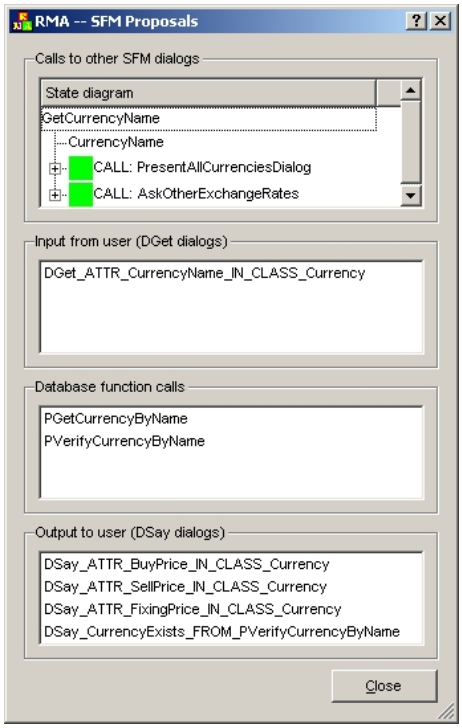

Figure 3: SFM proposals as offered by RMA.

Besides the data model information that we use to generate the automatic dialogues, the main source of the RMA is the output of the state flow modelling assistant (SFMA). For each state we generate automatically a dialogue. When that dialogue is edited, an 'SFM proposals' window pops up (see Figure 3), where all information specific to that state is offered to the designer, so that he can select predefined dialogues much faster.

Different kinds of dialogues: We have considered four basic types: dialogues based on a loop, based on a sequence of actions (or subdialogues), on information input by the user, and on the value of a variable (like a switch construction). We also allow empty dialogues, which are used to specify the call to a dialogue that will be defined afterwards.

Mixed initiative and overanswering: At the beginning of the development process when filling the application description, the designer defines if the service will allow system-driven or mixed-initiative strategy. Based on that, the assistant lets the designer input two or more slots in the same prompt, and offers different kinds of dialogues. Mixed initiative slots are defined in the SFMA, and the RMA generates automatically a DGet for that, which the designer just drops into the relevant window. Then, the code to handle that situation is automatically generated. For overanswering, the designer is always offered the possibility to add slots as overanswering whenever a DGet is dragged (these slots can be from the same state or from two following states).

Automated passing of arguments between dialogues: This is a critical aspect of dialogue design. Several dialogues have to be 'connected' as they use the information from the preceding dialogue. Typically three dialogues have to be connected. We can model that with just three drag & drop actions: (1) drag & drop a DGet information dialogue, (2) drag & drop a call to the database (from the data connector), and (3) drag & drop a DSay information dialogue. In all three actions all variable passing is automated, as the correct variable names are proposed to the designer, who just has to accept the proposals.

2.3. Dialogues layer

The dialogue layer is modality and language dependent as now the modality extensions from the modality extension retrieval assistant (MERA) and the language dependent extensions from the modality extension assistant (MEA) are added to the retrieval model.

The modality extensions consist of special subdialogues which are specific for one modality only. The current implementation of the AGP supports the generation of voice (speech modality) and web-based applications (web modality). This is the place where the designer can develop a dialogue with a complex output to the user (which is quite different between modalities), e.g. to present results of a query in a travel application, where many different results can be obtained, or in order to receive complex information from the user.

2.3.1. Modality extension retrieval assistant (MERA) for speech

At this stage we take care of dialogue aspects which are specific of a speech application, and so they have to receive a different treatment than in web:

Presentation of lists of objects: Lists of objects, which are usually the result of a database query, mean a lot of information. So, they need special treatment in a speech application. We have distinguished four cases as a function of the number of elements of the list: (1) The list is empty, (2) it has one item, (3) it has more than one item and less than a maximum allowed, (4) it has more items than the maximum allowed. Using the assistant, the designer can specify different behaviours for all cases, including the possibility to unset specific slots to make the queries more restrictive.

Confirmation handling: The result of speech recognition has to be confirmed before making a database query. We have considered two types of confirmation in our assistant: 'Simple' is recommended for dialogues with a very high confidence, as Yes/No or password questions. 'Complete' uses several levels of confidence to determine the confirmation type: none, implicit, explicit or repeat the question (like in a no match situation).

2.3.2. Modality extension assistant (MEA)

Within the modality extension assistant (MEA) the input and output behaviour of an application is described for each modality (see Figure 4). For speech modality the extensions consist of links to grammar and prompt concepts, which are language independent, while for web modality, the extensions consist of links to input and output concepts.

In addition, language dependent information, specifically wording for both speech prompts and web output concepts, is also set in MEA and it is saved in separate concept files for each language. For speech modality, prompts are first set for the main language of the application, using three alternatives: TTS prompts, audio files, or using runtime prompts generated by a natural language generation (NLG) module. In case of TTS prompts, the SSML mark-up language can be optionally used.

Additional language prompt specification is accelerated by using the main language prompt as a template. So the user has only to edit the string parts of a prompt. These prompts can be specified either at once for one language for all dialogues, or for each dialogue for all additional languages.

2.3.3. Generation of runtime scripts

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 web-page script generator depending on the selected modalities in the application description.

The VoiceXML generator has to deal with several problems and limitations of VoiceXML. One major problem is that VoiceXML

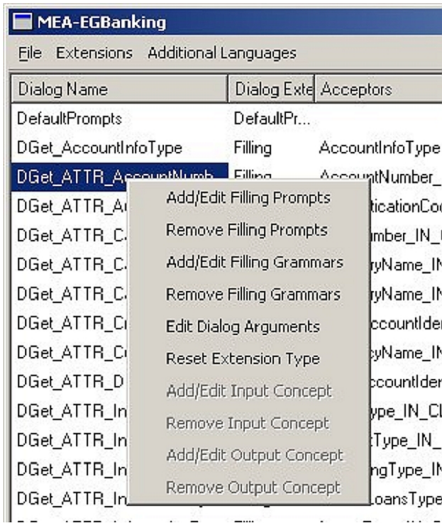

Figure 4: Input and output setting for each dialogue of the application.

does not allow returning calls as ordinary statements. Therefore, all complex statements and value expressions have to be 'flattened' into atomic operations.

The VoiceXML generator reads in all models and works directly on an internal representation of the models. This is only possible since the GDialogXML modelling language is very powerful and concept-oriented. The concept 'Variable', for instance, can be directly represented internally as a data structure. This means that there is one-to-one correspondence between the GDialogXML concepts and the model classes of the generator.

In order to connect the runtime services a data bridge has been developed. It allows for incorporating result values into VoiceXML by producing VoiceXML code dynamically. It constitutes the bridge between the VoiceXML script and the runtime services that are available via the newly developed GEMINI service protocol on top of HTTP. The main runtime services are

- the data connector, building the access layer to the database,
- the user level detector, switching between different user types,
- the prompt generator, generating natural language prompts on the fly,
- the speaker verification component, accepting or rejecting the speaker by performing pattern recognition on a sampled audio file,
- the language detector, identifying the currently spoken language.

For the web modality a web-page script is generated out of the dialogue model which enables dynamic web pages. For the speech modality, some more tools are relevant, namely the language modelling tool and the vocabulary builder.

3. Applications

Two pilot applications have been set up using the AGP for evaluation and validation. Both were generated in a very user friendly way, taking into account the error handling capabilities of the AGP.

The voice banking application called EG-Banking application constitutes a voice portal for user-friendly, high-quality interactions for bank-customers. The main functionality of EG-Banking includes a general information part available to the public and a transaction part available to customers of the bank only.

CitizenCare is an e-government dialogue system for citizen-to-administration interaction (via multiple channels like internet and public terminals), filled with content for an exemplary community. The main functionality is an interactive authority and information guide.

Please refer to [8] for more details about the applications.

4. Conclusion

During the GEMINI project we developed an application generation platform, which generates state of the art speech and web applications. The platform architecture is able to generate multi-modal and multilingual dialogue applications from different databases: Features like overanswering, user modelling, speaker verification, language identification can be included easily. Using the assistants of the AGP setting up new applications is done semi-automatically. The use of standards like VoiceXML and XHTML leaves the platform open for further development. Another important result of the project is the newly designed abstract dialogue description language, called GDialogXML.

5. Acknowledgements

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

6. References

[1] GEMINI Project Homepage: www.gemini-project.org.

[2] 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.

[3] 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.

[4] T. Brøndsted, B. N. Bai, and J. O. Olsen, "The REWARD Service Creation Environment, an Overview," in *Proceedings ICSLP*, Sydney, Australia, 1998, pp. 1175–1178.

[5] J. Polifroni, G. Chung, and S. Seneff, "Towards the Automatic Generation of Mixed-Initiative Dialogue Systems from Web Content," in *Proceedings EUROSPEECH*, Geneva, Switzerland, 2003, pp. 193–196.

[6] S. W. Hamerich, Y.-F. H. Wang, V. Schubert, V. Schless, and S. Igel, "XML-Based Dialogue Descriptions in the GEMINI Project," in *Proceedings of the 'Berliner XML-Tage 2003'*, Berlin, Germany, 2003, pp. 404–412.

[7] Y.-F. H. Wang, S. W. Hamerich, and V. Schless, "Multi-Modal and Modality Specific Error Handling in the GEMINI Project," in *Workshop on 'Error Handling in Spoken Dialogue Systems'*, ser. ISCA, Chateau d'Oex, Switzerland, 2003, pp. 139–144.

[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 *Workshop for Discourse and Dialogue*, ser. SigDial, Cambridge, USA, 2004, pp. 31–34.