Multi-Modal and Modality Specific Error Handling in the GEMINI Project

Yu-Fang H. Wang, Stefan W. Hamerich, Volker Schless

TEMIC Speech Dialog Systems
Research Department
Soeflinger Str. 100 – 89077 Ulm
Germany
{helena.wang|stefan.hamerich|volker.schless}@temic-sds.com

Abstract

GEMINI (Generic Environment for Multilingual Interactive Natural Interfaces) is an EC-funded research project which started in April 2002. The goal of GEMINI is to provide a flexible platform for the generation of applications, able to produce multi-modal and multilingual dialogue interfaces to databases with a minimum of human effort. All description models which are generated by the tools of the platform are based on our newly designed XML-based description language called GDialogXML (Gemini Dialog XML). It is an abstract dialogue description language, which is flexible enough to account for both modality independent and modality specific features of the target language, i.e. VoiceXML or xHTML. The platform will generate high-quality, state-of-the-art applications, e.g. a speech application that includes mixed-initiative dialogues and sophisticated error handling.

In this paper, we focus on the treatment of error handling for speech. Error handling for the web will be done in a future stage of the project.

1. Introduction

GEMINI is funded within the European Union's Fifth RTD Framework Programme for two years beginning 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).

The main objective of the GEMINI project is to provide an application generation platform (AGP) for the semi-automatic generation of high-quality applications (see section 2.2). One of the pilot applications will be a citizen-to-administration information system. In Figure 1 an example dialogue is presented.

GEMINI's special feature is the capability to easily generate an application in several modalities (e.g. speech or web) and several languages. 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. For example, certain prompts will be generated in the generic dialogue form since they will occur regardless of a modality (e.g. database connection failure or help prompts). Because of that, prompts are represented by references rather than concrete verbalisations.

sys:	Welcome to CitizenCare. Please say your concern or the name of an authority.
usr:	Identity Card
sys:	The registration office is concerned with identity cards. Do you want information about the authority, the contact person, or the procedure?
usr:	About the authority please
sys:	You can ask for special information like the address or opening hours, or just say "all"
usr:	When is it open today?
sys:	Today the office hours are from 9 am to 1 pm.

Figure 1: Example dialogue for the CitizenCare application.

To ease the generation of a specific application out of the modality-independent and the modality-dependent part of the dialogue description, we have developed a generic description language, GDialogXML (see section 2.3).

The platform is accessible via an assistant that helps the developer to create the target application. The intended developer is a speech or web application designer who does not want to care about the peculiarities of database queries, thus GEMINI enables non-database-experts to use access to database applications.

The resulting applications make use of prominent features of the target language, i.e. the resulting VoiceXML scripts will allow for mixed-initiative dialogues and access to external database servers. Additionally the error handling capabilities of VoiceXML will be extended (refer to section 3.2.4).

When using the term error handling, we distinguish several types of errors:

1. feedback to the users on errors outside the dialogue, e.g. database connection failure (see section 3.1);
2. reducing the occurrence of user input that does not match the current grammar: design of prompts, especially user-level-dependent prompts and help prompts (refer to section 3.2.1);
3. avoiding incorrectly recognised user input by using confirmation questions (see section 3.2.4).

This paper is organised as follows: After introducing the objectives of GEMINI, we briefly give an overview of the platform built during the project and the description language that is

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.

used to model abstract and modality independent dialogues. Afterwards, we present the implementation of the error handling concepts in GEMINI and the evaluation criteria used.

2. GEMINI

2.1. Project Goals

GEMINI [1] 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 databases 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 based on this platform: CitizenCare, an eGovernment platform framework for citizen-to-administration interaction which will be available for spoken and web-based user interaction (German and English), and EG-Banking, a voice-portal for interactions with bank customers (English, German, Greek, and Spanish).

One 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 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. [2, 3, 4]).

Other features are:

- user-friendly and transparent GUI
- availability of linguistic resources: libraries for dialogue acts and grammars for several languages.
- separation of dialogue modelling and data query implementation
- easy development of an interface to a database application without being a database expert
- connectivity to several data sources (e.g. various DBMS, XML documents)
- high-degree of personalisation: i.e. user modelling and speaker verification
- support of different run-time platforms

The expected gains from GEMINI are faster development times and cost reductions (as compared to standard development).

2.2. The Application Generation Platform (AGP)

The approach for setting up the AGP is a three-level architecture with increasing specificity regarding the modalities. For a graphical overview see Figure 2.

The first level is the most general one which specifies (1) the basic application description, (2) the database connector (containing e.g. the database settings), and (3) the data model. We call this layer framework layer. The application description contains global variable settings for error handling like timeout for speech detection and database connection, and the number of retries for getting the user's input.
All input to this layer is done manually using graphical user interfaces.

On the second level, the modality- and language-independent dialogue structure is generated in a semi-automatic

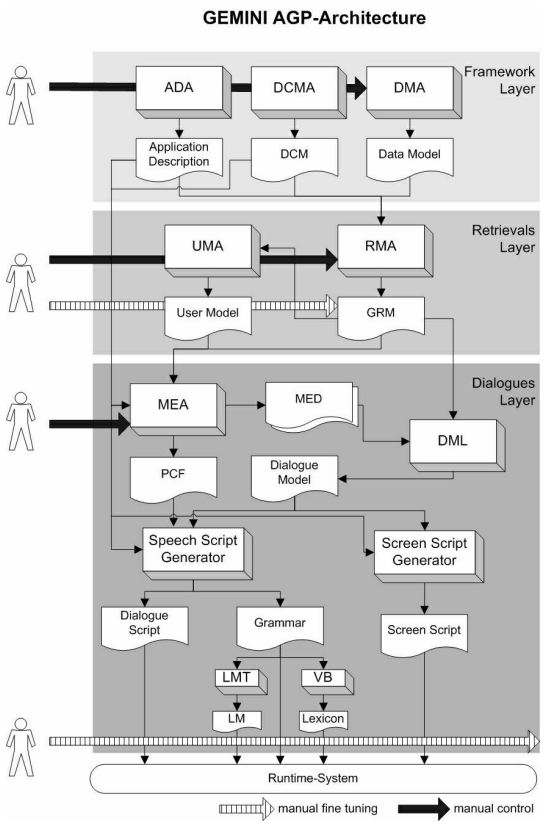

Figure 2: *Schematic view of the AGP architecture.*

way. This is mainly done in the Retrieval Modelling Assistant (RMA), which produces the Generic Retrieval Model (GRM). The GRM contains the abstract dialogue flow, as well as references to the multi-modal help concepts. Here, user-level dependent responses to erroneous input and similarly, user-level specific prompts are referenced.

The third level adds modality-specific data to the general dialogue level. Modality-specific differences in the dialogue flow will be modelled here, as well as the language-dependent parts like prompts, grammars, and error messages. The modality-specific data, called Modality Extension Description (MED) is automatically linked with the GRM, resulting in the so-called Dialogue Model. For each target modality, a separate dialogue model is created, including a complete dialogue description with prompt concepts, help concepts and grammar references. The verbalisation of the prompts is done in the prompt concept file (PCF). The dialogue model, having been merged from the GRM and MED, is completely implemented in GDialogXML, which is described in more detail in the next section.
For speech, a VoiceXML script is finally generated automatically from (1) the dialogue model, (2) the prompt concept file, (3) the application description, and (4) the database connector. Analogously, a xHTML script will be generated for the web modality.

2.3. GDialogXML

GDialogXML is an XML based object-oriented, abstract dialogue description language, which was newly defined for use in

the GEMINI AGP.

In GDialogXML, each dialogue has the following sections:

variables here all local variables of the current dialogue are declared. In GDialogXML, there are the following types of variables: local variables (for any local operation), input variables (for containing user input), arguments (containing parameters for this dialogue), and return values (containing values which are given back from the current dialogue)

presentation this is used for calling the presentation for the current modality, which means calling a prompt for speech and calling an editor for web modality

filling here user input is collected, for speech modality this is done by calling the recogniser with a specified grammar (rule)

reaction specifies the reaction that will be initiated after getting some user input

help means the reference to a multi-modal help concept, which will be defined in the PCF.

```
<xPresentation>
  <PromptCall>
    <xPrompt>
      <PromptConcept refr="ask4City"/>
    </xPrompt>
  </PromptCall>
</xPresentation>
<xFilling>
  <RecognizerCall>
    <xGrammar>
      <Grammar refr="cities" type="jsgf"/>
    </xGrammar>
    <xAcceptors>
      <Var refr="city"/>
    </xAcceptors>
    <xNomatchReaction>
      <PromptCall>
        <xPrompt>
          <PromptConcept refr="CityNomatch"/>
        </xPrompt>
      </PromptCall>
      <DoFilling/>
    </xNomatchReaction>
  </RecognizerCall>
</xFilling>
```

Figure 3: Clipping from MED example.

Since for the speech modality the most important factors are presentation and filling, a short clipping from these parts in a MED is given in Figure 3. It can be seen that modality specific error handling is part of GDialogXML. In the figure only the nomatch error is displayed, which means that in the current utterance no match for the grammar could be found. In general, GDialogXML allows the handling of the following recognition errors:

- ”noinput”: no recognition after exceeding the timeout threshold
- ”nomatch”: the recognised utterance could not be matched with any item of the specified grammar
- ”confidence_too_low”: the confidence value of the recognised utterance is below the lower threshold

3. Error Handling Capabilities

The dualistic approach allows for modality independent as well as for modality specific error handling. The first error type will be handled in the GRM, while the second one will be dealt with in the respective MEDs.

3.1. Modality Independent Error Handling

An example for an error which might occur independent of the modality is a database connection failure, i.e. the server does not respond after some specified timeout value. Its handling involves (1) receiving the error message from the assistant that calls the database server, and (2) giving an appropriate feedback message to the user.

Here the idea of using prompt references instead of full verbalisations proves useful: the system reaction can be expressed in a modality independent way while its modality dependent realisation has to be presented via different output channels.

3.2. Error Handling for Speech

In this section our concepts for error handling in the speech modality are presented.

Underlying all concepts for handling errors is a careful design of the prompts, which significantly contributes to dialogue success. For example, prompts determine the syntactic category of the user response so both items should be consistent. For example, asking ”where do you want to go to” will trigger a to-PP (e.g. ”to Paris”), while the question ”what is your destination” will trigger a simple noun phrase (e.g. ”Paris”). We also have to take into account the advantages of directed prompts as compared to open-ended prompts, see [9, 10] for further details.
As has already been described in section 1, prompts are also relevant for typical error handling contexts, i.e. for offering help or informing the user on some malfunctions of the system.

3.2.1. Incremental Help on Noinput and Nomatch

Instead of always prompting ”I did not understand” or ”I could not hear you” we will we offer incremental help. At each stage more information is provided. For example:

- 1st time noinput: Pardon?
- 2nd time noinput: Sorry. Please name the destination city.
- 3rd time noinput: [automatic transfer to operator]

This strategy avoids users to be annoyed by lengthy prompts. On the other hand, users which do not know what to say, are guided by the system to answer according to the current dialogue state.

3.2.2. User Level Dependent Error Handling

The wordings of the prompts also depend on the current user level. For the GEMINI AGP we will support at least two different user levels, ’novice’ and ’expert’. While the former user type will obtain extensive help prompts, short help prompts are regarded necessary for the latter in order not to annoy the user.

3.2.3. Verification Status of Filled Slots

Verification is indispensable for the system to complete its task. For this purpose, it is necessary to know the verification status of the current dialogue field variable. However VoiceXML does not offer an inbuilt mechanism for the verification of slot variables. Therefore we simulate this property by adding an extra variable to each slot containing the verification status of the variable. This is a temporary solution since we would appreciate more enhanced and integrated methods for VoiceXML.

3.2.4. Confidence-Driven Choice of Implicit or Explicit Verification

Confidence scores for speech recogniser output are often used for dialogue management, refer e.g. to [5, 6].

Although explicit verification is easy to understand for the user, it has the disadvantage of unnecessarily lengthening the dialogue. While implicit verification seems to be an elegant solution in terms of speeding up the dialogue and being "more natural", it does have a well-known disadvantage: If the utterance is incorrectly recognised and prompted back, the user can be severely confused (see [7, 8]). Consider the dialogue fragment in Figure 4.

sys:	what is your **destination**?
usr:	Rome please
sys:	**when** do you want to leave from Rome?
usr:	Wednesday
sys:	[mistaking Wednesday as Venice] when do you want to leave from Venice?

Figure 4: Example dialogue with classification error.

There are several possibilities to avoid implicit verification questions and classification errors to co-occur in one system prompt:

- to provide for an explicit "no" in the grammar to recognise the user rejection of an implicit verification. This way, the difference between a disconfirmation and a normal continuation of the dialogue would just be identified by a preceding "no" in the first case ("no, Wednesday"). However, this is an unreliable indicator since the corrected utterance could also just be prosodically marked ("WEDNES-day").
- make use of language models that give priority to the 'new' word which are also in the focus. So, for the example above, a time expression is more likely to occur than a city name.

Language models will in fact be used in GEMINI. Independent of those or other statistics-based methods (s. [11]), we have chosen a simple, confidence-based solution to combine the advantages of both strategies while making their disadvantages less likely to occur. A basic design decision is that even with an excellent confidence score, the user utterance must be confirmed before leaving the dialogue, because this still does not guarantee correct recognition. We distinguish three confidence intervals:

1. In the case of very high confidence scores, implicit verification is used. This is because in the upper confidence interval, correct recognition is most probable and will make confusing verification questions less likely to occur.
2. For a medium-range score, explicit verification is applied to verify the recognition result. So, even if a classification error occurs, it is much easier for the user to react on the system prompt, see section 5.
3. In the remaining cases (i.e. the confidence value of the utterance is under the low-confidence-threshold), incremental help is offered to support the dialogue.

Applying this, the example from section 4 changes by replacing the last sentence which contains the implicit verification question by the clearer explicit verification question:

sys:	...
usr:	Wednesday
sys:	[mistaking Wednesday as Venice] Did you say Venice?

Figure 5: Clipping of example dialogue using a dialogue strategy based on confidence values.

For the AGP, we are planning to integrate this method of choosing among verification strategies and measure the number of correct recognitions for each strategy during the evaluation phase.

4. Evaluation

The project will be evaluated with respect to the two main objectives named in chapter 2.1. For this evaluation the standards are provided by the EAGLES project, e.g. refer to [12].

1. Evaluation of the AGP: Compared to implementing a speech or web application without any automatic generation environment, does GEMINI really provide a time-saving and comfortable tool? How satisfied are system developers with it?
2. Evaluation of the applications generated from it: How good is the overall quality of the resulting application?

4.1. Evaluation of the AGP

The evaluation will be performed by having experienced system designers use the GEMINI platform. It will involve the following steps:

1. Training Phase: The developers will be instructed on how to use the platform. They will write test applications in order to get familiar with it.
2. Implementation Phase: Complex applications are created using the AGP.
3. Evaluation Phase: The developers will fill out an evaluation form that, for each property of the GEMINI platform, asks two questions: Was the property available at all, and how important does the developer regard the feature? Objective criteria will be the absolute development time for a base application, for the application in another modality, and in another language. It will be measured how fast these derived applications (additional modalities or languages) were set up.

In the evaluation form, we are mainly interested in the following questions:

- Which properties are considered most important, e.g. speed up of application development, reusability of dialogue modules?
- Which features are important to be included in such a platform, e.g. multi-modality, multilinguality?

4.2. Evaluation of the AGP Applications

We evaluate the applications in order to get feedback on their overall quality. The results will be used for constant improvement and tuning of the system. Evaluation will take place in three stages. At each stage, the respective user group will be less experienced with the system and the number of users will be larger, ranging from GEMINI developers to pre-selected users from the public. While the first phase serves mainly as quality assessment of the system modules, measuring e.g. speech recognition and grammar accuracy, the second and third stage focus on the dialogue behaviour. We will use objective and subjective parameters.

4.2.1. Objective Parameters

There are the following objective parameters:

- Duration parameters: dialogue duration, turn duration. We will measure the average time for a routine operation and relate them to problematic dialogues. Ideally, the duration values are indicators whether a dialogue was problematic even before analysing it.
- Task completion: task success rate, i.e. did the user get the expected information? Percentage of tasks that supplied the user with the required information.
- Counts of negative cues: number of hang-ups, number of dialogue turns, number of calls to the operator.
- Correction capabilities: was the system able to accept the user correction? This will be investigated for both explicit and implicit verification questions.
- Classification errors: how many times did classification errors occur? These numbers can be used to adjust the confidence threshold between explicit and implicit verification. If the number of errors is high, the system might have asked many confusing implicit verification questions (so raise the threshold). On the other hand if the number of errors is low, there might have been too many annoying explicit verifications (so lower the threshold). Moreover, the classification errors will tell us about how well the language models fit to the respective application.

With regard to error handling as described in section 3, the following points will be relevant:

- Database failure: Does a (simulated) database connection failure trigger the appropriate error message?
- Noinput, nomatch errors: Number of noinput and nomatch errors in each dialogue and in the overall corpus. This will give feedback on the speech recognition quality and grammar accuracy.
- Incremental help: Number of occurrences of each help level. How often did level 1 occur (”Sorry I did not hear you. What did you say?”), how often level 2, etc. In average, did the users wait until level 3 to be transferred to a human operator, or did they hang up before the dialogue ended?
- Verification: Number of occurrences of explicit and implicit verification questions per dialogue and in the overall corpus; number of classification errors for explicit and implicit verification

4.2.2. Subjective Parameters

For subjective evaluation, the user will have to answer a questionnaire to express their opinions and feelings about the system. We are interested in the following aspects:

- User satisfaction: E.g. did you get the information that you asked for? Did the system understand you at first go? Did the system work as expected?
- Questions regarding prior experience: Have you got prior experience with such systems?
- System performance: Are there systematic errors? Are there words not recognised by the system?

With regard to error handling as described in section 3, we will ask questions as follows:

- Help: Did you find the help provided sufficient? Did you find the incremental help concept useful or is there a step that you find superfluous (e.g. ”please repeat”). What would you improve?
- Explicit verification: Was it always clear to you what the system wanted?
- Implicit verification: Was it always clear to you what the system wanted?
- Verification: Did you feel more comfortable with explicit or implicit verification questions? If this differed from case to case, please give examples.

5. Conclusion and Future Work

The main task of the GEMINI project is the design and implementation of an application generation platform, which generates state of the art speech and web applications. For this platform, an abstract dialogue description language, called GDialogXML, was defined. Apart from other features, it allows for both multi-modal and modality specific error handling.

We introduced the error handling capabilities of the AGP and its applications. This includes user-level dependent and confidence driven error handling strategies. For evaluation both objective and subjective criteria will be applied.

We will use the platform’s ability to easily generate several applications for the evaluation of different dialogue and error handling strategies. Based on this evaluation the platform will be extended by additional features. Among others the integration of speaker and language identification capabilities are the next steps.

6. References

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

[2] Ehrlich, U. et al., ”ACCeSS - Automated Call Center through Speech Understanding System”, in Proc. EUROSPEECH, Rhodes, Greece, 1997, p. 1819-1822.

[3] Brøndsted, T. et al., ”The IntelliMedia Workbench – a generic Environment for multimodal Systems”, in Proc. ICSLP, Sydney, Australia, 1998.

[4] Lehtinen, G. et al., ”IDAS: Interactive Directory Assistance Service”, in Proc. of the international Workshop ’Voice Operated Telecom Services’, COST 249, Ghent, Belgium, 2000, p. 51-54.

[5] Komatani, K. and Kawahara, T., ”Flexible Mixed-Initiative Dialogue Management using Concept-Level Confidence Measures of Speech Recognizer Output”, in Proc. COLING, Saarbrücken, Germany, 2000, p. 467-473.

[6] San-Segundo, S. et al., ”Confidence Measures for Dialogue Management in the CU Communicator System”, in Proc. ICASSP, Istanbul, Turkey, 2000.

[7] Lavalle, S. et al., ”Dialogue and Prompting Strategies Evaluation in the DEMON System”, in Proc. LREC, Athens, Greece, 2000

[8] Weegels, M., ”Users’ Misconceptions of a Voice-Operated Train Travel Information System”, in IPO Annual Progress Report, Eindhoven, The Netherlands,1999

[9] Litman, D. J. et. al. ”Evaluating Response Strategies in a Web-Based Spoken Dialogue Agent.” in Proc. ACL and COLING, Montreal, Canada, 1998, p. 780-786.

[10] Balentine, B.; Morgan, D., ”How to build a speech recognition application. A style guide for telephony dialogues”, Enterprise Integration Group, San Ramon, USA, 1999.

[11] Krahmer, E. et al. ”Problem Spotting in Human-Machine Interaction”, in Proc. EUROSPEECH, Budapest, Hungary, 1999.

[12] Hanrieder, G. et al. ”Fly with the EAGLES: Evaluation of the ’ACCeSS’ Spoken Language Dialogue System”, in Proc. ICSLP, Sydney, Australia, 1998.