

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**Speed Up Strategies for the Creation of
Multimodal and Multilingual
Dialogue Systems**

TESIS DOCTORAL

LUIS FERNANDO D'HARO

Ingeniero Electrónico

2009

2009

LUIS FERNANDO D'HARO

TESIS DOCTORAL

9

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE TELECOMUNICACIÓN



**Speed Up Strategies for the Creation of
Multimodal and Multilingual
Dialogue Systems**

Autor

LUIS FERNANDO D'HARO

Ingeniero Electrónico

Director

RICARDO DE CÓRDOBA

Doctor Ingeniero de Telecomunicación

2009



Tribunal nombrado por el Sr. Rector Magfco. de la Universidad Politécnica de Madrid, el día.....de.....de 2009...

Presidente: _____

Vocal: _____

Vocal: _____

Vocal: _____

Secretario: _____

Suplente: _____

Suplente: _____

Realizado el acto de defensa y lectura de la Tesis el día..... de..... de 2009 en la E.T.S.I. de Telecomunicación.

Calificación:.....

EL PRESIDENTE

LOS VOCALES

EL SECRETARIO

“Commit your way to Jehovah, trust also
in Him, and He will do it”

Psalm 37:5

“Encomienda a Jehová tu camino,
Confía en Él, y Él hará”

Salmos 37:5

"Bear in mind that the wonderful things you learn in your schools are the work of many generations, produced by enthusiastic effort and infinite labour in every country of the world. All this is put into your hands as your inheritance in order that you may receive it, honour it, add to it, and one day faithfully hand it to your children. Thus do we mortals achieve immortality in the permanent things which we create in common"

Albert Einstein

“Pensad que las cosas maravillosas que aprendéis en vuestras escuelas son el trabajo de muchas generaciones, logrado con mucho esfuerzo y mucha fatiga en todos los países del mundo. Las ponemos en vuestras manos como herencia, para que las respetéis, desarrolléis y fielmente las entreguéis a vuestros hijos. Así es cómo nosotros, los mortales, nos hacemos inmortales, transmitiendo el trabajo hecho por todos”

Albert Einstein

ABSTRACT

Nowadays, most of the commercial and research call center applications are created using sophisticated and complete development platforms that allow the specification of all the details related to the design, deploying, and debugging of such services. In spite of all the features and utilities included in them, most of them propose the same kind of accelerations and present limitations when designing simultaneously the same service for different modalities and kinds of users.

In this thesis, we propose different innovative, dynamic, and intelligent acceleration strategies that allow the prediction of the information required to complete the different aspects of the design. In our proposal, the accelerations are based on using the data model structure and database contents, as well as cumulative information obtained from the previous and sequential steps in the design. Thanks to these accelerations, the design is reduced, most of the times, to simple confirmations from the designer to the “proposals” that the platform automatically provides.

In detail, we propose the semi-automatic generation of different kinds of proposals that can be used to complete the application flow, the actions that make up each dialogue, or to solve specific modality problems such as user confirmations and the presentation of the lists of results retrieved after querying the backend database. Additionally, we propose the creation of different assistants that contribute to accelerate the process of creating speech grammars and the definition of the functions used to access the database. The results that we have obtained in objective and subjective evaluations have shown the viability, relevance, and functionality of the platform and the proposed accelerations presented in this thesis.

On the other hand, the wide variability of the final users of the service raises different challenges such as the possibility of correctly identify the language to be used to interact with the users, as well as the possibility of providing the same service using different modalities according to the user preferences or needs or to the current conditions of the dialogue.

In relation with the improvements applied to the language identification module, we have implemented a new technique based on using a discriminative ranking of n-grams that allow the incorporation of contextual longer-span information into the language models used by the system. The proposed technique has been evaluated in the identification of spoken sentences in English and Spanish obtaining better language recognition rates than a PPRLM based system, probably because the technique copes better with the classical problem of obtaining reliable estimates with a reduced training set, so we can use higher order language models.

Finally, we have incorporated several improvements into an automatic speech-to-sign language machine translation system that extends the multimodal capabilities of the platform, so we can offer the same service, developed with the design platform, to deaf people. In this case, the translation system is used to automatically translate system prompts into an animated sequence played by a 3-D avatar. In this thesis, we propose an innovative adaptation technique that improves the quality of the translated sentences in situations when there is not enough training data to obtain reliable language models used by the translation system. The adaptation is done at the count level, using the Maximum A-Posteriori (MAP) technique. We use in this case the original occurrence counts of the n-grams that appear in the target language and the frequency counts of the equivalent n-grams in the source language retrieved from the Web and previously “translated” into counts in the target language using an independently trained phrase-base translation model.

RESUMEN

Hoy por hoy, la mayoría de los sistemas comerciales y de investigación de atención telefónica se realizan mediante el uso de sofisticadas y completas plataformas que permiten especificar todos los detalles relacionados con el diseño, ejecución, y depuración de tales servicios. Pese a todas las funcionalidades y utilidades incluidas para acelerar el diseño y permitir servicios avanzados a los usuarios, la mayoría de ellas proponen el mismo tipo de aceleraciones y presentan limitaciones al desarrollo simultáneo del servicio para múltiples modalidades y perfiles de usuario.

En esta tesis se proponen diferentes estrategias de aceleración innovadoras, dinámicas e inteligentes que permiten predecir la información necesaria requerida para completar los diferentes aspectos del diseño, usando para ello información de la estructura del modelo de datos y del contenido de la base de datos del servicio, así como de la información acumulada a lo largo de todos los pasos ya realizados durante el diseño. Gracias a estas aceleraciones, la mayor parte del diseño del diálogo se reduce a confirmaciones por parte del diseñador de las “ofertas” que le hace la plataforma.

En concreto, se propone la generación semi-automática de diversos tipos de propuestas que pueden ser utilizadas para completar el flujo de la aplicación, las acciones que componen cada diálogo, o para solucionar problemas específicos de cada modalidad tales como la confirmación de datos al usuario y la presentación de las listas de resultados devueltos después de hacer una consulta a la base de datos del servicio. Así mismo, se propone la creación de diferentes asistentes que permiten acelerar la creación de las gramáticas usadas por el sistema de reconocimiento y la definición de las funciones de acceso a la base de datos. Los resultados obtenidos en sendas evaluaciones objetiva y subjetiva han permitido demostrar la viabilidad, relevancia y funcionalidad de estas aceleraciones y de la plataforma presentada.

Por otro parte, la amplia variedad de usuarios finales del servicio plantea diversos retos tales como la capacidad de identificar adecuadamente el idioma con el cual dirigirse a los usuarios, así como la posibilidad de proporcionar el servicio utilizando una u otra modalidad según las preferencias/necesidades de los usuarios o las condiciones actuales del diálogo.

En relación con las mejoras aplicadas al módulo de reconocimiento de idioma se ha implementado una nueva técnica para la incorporación de información contextual de más largo alcance en los modelos de lenguaje utilizados por el sistema basada en un ranking de n-gramas discriminativos. La técnica propuesta ha sido evaluada en la identificación de frases habladas en inglés y castellano obteniendo mejores tasas de reconocimiento que un sistema basado en PPRLM que usa modelos de lenguaje tradicionales gracias a la reducción del problema de falta de datos para el entrenamiento de los modelos de lenguaje de orden elevado lo que permite la utilización de modelos de mayor orden.

Finalmente, se han incorporado diversas mejoras a un módulo de traducción automática de voz a lengua de signos que permite ampliar las capacidades multimodales de la plataforma al permitir la prestación del mismo servicio, desarrollado con la plataforma de diálogo, a personas con discapacidad auditiva, permitiendo la traducción de los prompts del sistema en una secuencia animada reproducida por un avatar. En esta tesis se propone una técnica de adaptación innovadora que permite mejorar la calidad de las frases traducidas en situaciones en las que no hay suficientes datos para entrenar adecuadamente el modelo de lenguaje usado por el sistema de traducción. La adaptación se realiza a nivel de cuentas, mediante la técnica de Maximum-A-Posteriori (MAP), usando las cuentas de los n-gramas originales en el

idioma destino y las cuentas de ocurrencia de los n-gramas equivalentes en el idioma origen consultadas en la Web previamente y “traducidas” posteriormente a cuentas en el idioma destino usando un modelo de traducción basado en frases.

ACKNOWLEDGMENTS

When we think in a thesis, we are not just talking about a document that is written at the end of several years of research, thousands of experiments, and many nights with insomnia. But, and above all, it is a road that without the support and orientation of many persons, it would not be possible to start and, even worse, to finish. For this reason, and with the limitation of using just some few words, I want to express my gratitude to all those people that had contributed to finish successfully this stage of my life.

In first place, if there is somebody to whom I have a duty for all that I am, all that I have been, and all that I will be, is to my Heavenly Father, to Jesus my Saviour, and to the Holy Spirit my Comforter. To You be the Glory, the Honour, and all my gratitude. Thanks, because your love and wisdom expressed in the tiny atom, in every star in the immensurable cosmos, and in every instant of my life make me possible to feel that this world, this thesis, and my whole life, is the unquestionable result of Their constant direction and protection.

In second place, I want to thank my family, my wife Ana Lucía, my little son Samuel Fernando, my fathers Luis and Yolanda, and my sisters Angélica and Viviana. Thanks for your support, love, and happiness that every day you give to me, and for trusting in me. Thanks, because each minute we shared together have given me the strength to keep on the way and to be a better person. Thanks for teaching me and helping me to build the values that, like the stars on the sky, have made me possible to navigate for the life.

I want to thank all the members of the Speech Technology Group. I have not words to thank you for all the things you have made for me. Thanks for allowing me to come to this country to make my dream true, for guiding me and advising me at every step in my research, and for giving me the opportunity of growing professionally and personally. Thanks for your confidence, patient and friendship. I also want to thank Javier Morante, Ignacio Ibarz, Rosalía Ramos, and José Ramón Jiménez for their contribution in the development of the accelerations presented in this thesis, as well as to all the members of the GEMINI project for making possible the creation of the platform used in this thesis.

I want to make a special mention and thank to my advisor. Rick, thanks for all you have done for me. Thanks for being not just an excellent advisor, directing every detail in my research, but because you have become like a brother to me. Thanks for listening to me and advising me so many times. Thanks for every challenge and opportunity you have given me. God bless you and continue guiding your life.

Finally, I cannot forget the big group of friends that have been always close to comfort me with a pray, an advice, opening the doors of their homes and hearts, or offering to me something as simple, but very important, as a smile. A special mention to Willy, Alfredo, and Samuel. Thanks guys for teaching me the meaning of the true friendship, and for sharing that goal that united us: to be AIBAS[®] for ever.

AGRADECIMIENTOS

Una tesis no es sólo un libro que se escribe al final de varios años de investigación, multitud de experimentos y varias noches sin dormir sino, y por sobretodo, es un camino a recorrer en el que sin el apoyo y la orientación de muchas personas no sería posible iniciar y mucho menos terminar. Es por eso que quiero, aunque sea con estas pocas palabras, dar mi agradecimiento a todos aquellos que han contribuido de una u otra forma a que pueda culminar con éxito esta etapa de mi vida.

En primer lugar, si hay alguien a quien debo todo lo que tengo, todo lo que soy, y todo lo que llegaré a ser, es a mi Padre Celestial, a Jesús mi Salvador, y al Espíritu Santo mi Consolador. A ellos sea la Honra y la Gloria, y todo mi agradecimiento, porque su amor y sabiduría manifestada en cada diminuto átomo, en cada estrella del inmensurable cosmos, y en cada instante de mi vida me hacen sentir que este mundo, este camino, y mi vida entera, es el indiscutible resultado de su constante dirección y protección.

En segundo lugar quiero agradecer a mi familia: a mi esposa Ana Lucía, mi pequeño hijo Samuel Fernando, mis padres Luis y Yolanda, y a mis hermanas Angélica y Viviana. Gracias por todo el apoyo que me han dado, por el amor y felicidad que cada día me dan y por creer en mí. Gracias porque todos y cada uno de los minutos que he compartido con ustedes me han hecho ser mejor, y me han dado fuerzas para continuar. Gracias por enseñarme y ayudarme a construir los valores que, a modo de estrellas, me han permitido navegar por la vida.

También quiero agradecer a todos los miembros del Grupo de Tecnología del Habla. No tengo palabras para agradecerles todo lo que han hecho por mí. Gracias por permitirme venir a este país a cumplir un sueño, por guiarme y aconsejarme en cada paso de la investigación y por darme la oportunidad de crecer profesional y personalmente. Gracias por su confianza, paciencia y amistad. También agradezco a Javier Morante, Ignacio Ibarz, Rosalía Ramos y José Ramón Jiménez por su contribución en el desarrollo de los asistentes, así como a todos los miembros del proyecto GEMINI por haber hecho posible la creación de la plataforma usada en esta tesis.

Mención aparte, quiero expresar mi gratitud al director de esta tesis. Rick, gracias por todo lo que has hecho por mí. Gracias por haber sido no sólo un excelente tutor al dirigirme en cada detalle de esta investigación, sino porque has llegado a ser como un hermano para mí. Gracias por escucharme y aconsejarme tantas veces. Gracias por todos los retos y oportunidades que me has dado. Que Dios te bendiga y siga guiando tu vida.

Finalmente, no puedo dejar de recordar a todo ese grupo grande de amigos que siempre han estado allí para apoyarme con una oración, con un consejo, abriéndome las puertas de sus hogares y sus corazones, o incluso compartiendo conmigo algo tan sencillo, pero tan importante, como es una sonrisa. Mención especial quiero dar a Willy, Alfredo y Samuel. Gracias chicos por enseñarme el valor de la verdadera amistad y por poder compartir con ustedes ese ideal que nos une de ser AIBAS[®] por siempre.

INDEX

ABSTRACT	I
RESUMEN	II
ACKNOWLEDGMENTS	IV
AGRADECIMIENTOS	V
INDEX	VI
INDEX OF FIGURES	X
INDEX OF TABLES	XIV
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Objectives	3
1.2.1 Design platform	3
1.2.2 Language Identification System	3
1.2.3 Machine Translation	4
1.2.4 Relevant Definitions	4
1.3 Organization	5
2 STATE-OF-THE-ART	7
2.1 Development Platforms and Acceleration Strategies for Designing Multimodal Dialogue Systems	7
2.1.1 Commercial Platforms	8
2.1.2 Academic and Research Platforms.....	16
2.1.3 Research Platforms that Provide an Assisted Dialogue Design	19
2.1.4 Weaknesses of Commercial and Academic Platforms.....	24
2.2 Language Modelling.....	24
2.2.1 Statistical Language Models	25
2.2.2 Context-Free-Grammars (CFG's)	35
2.3 Language Identification (LID)	36
2.3.1 Description of the PPRLM Technique: Advantages and Disadvantages.....	39
2.4 Machine Translation.....	41
2.4.1 Current Approaches for Machine Translation.....	41
2.4.2 Word-based and Phrase-based Translation.....	45
2.4.3 Current Metrics for the Automatic Evaluation of Machine Translation Quality...	47
2.4.4 Speech to Sign Language Translation.....	50
3 PLATFORM ARCHITECTURE	57
3.1 GDialogXML: Internal Descriptive Language for the Generated Models.....	58
3.2 FrameWork Layer	59
3.2.1 Application Description Assistant (ADA).....	59
3.2.2 Data Model Assistant (DMA)	60
3.2.3 Data Connector Modelling Assistant (DCMA)	61
3.3 Retrieval Layer	62
3.3.1 State Flow Modelling Assistant (SFMA)	62
3.3.2 Retrieval Modelling Assistant (RMA).....	63
3.4 Dialogues Layer.....	64

3.4.1	<i>User Modelling Assistant (UMA)</i>	65
3.4.2	<i>Modality Extension Retrieval Assistant for Speech (MERA-Speech)</i>	65
3.4.3	<i>Modality and Language Extension Assistant (MEA)</i>	66
3.4.4	<i>Dialogue Model Linker (DML)</i>	69
3.4.5	<i>Script Generators</i>	69
3.4.6	<i>Auxiliary Assistants</i>	71
3.5	<i>Runtime System</i>	74
3.5.1	<i>Speech Recognizer and Synthesizer</i>	74
3.5.2	<i>Animated Agent Used by the Sign Language Translation System</i>	75
3.5.3	<i>Distributed Platform and VoiceXML Interpreter (OpenVXI)</i>	77
3.5.4	<i>Portability and Use of Standards</i>	82
3.6	<i>Scope and Limitations</i>	83
4	SPEED UP STRATEGIES APPLIED IN THE DIALOGUE DESIGN	85
4.1	<i>Heuristics</i>	86
4.2	<i>Strategies Applied to the Data Model Assistant (DMA)</i>	88
4.2.1	<i>Semi-automatic Classes Proposals</i>	89
4.2.2	<i>Common Accelerations</i>	90
4.3	<i>Strategies Applied to the Data Connector Model Assistant (DCMA)</i>	91
4.3.1	<i>Definition of Relations between the Function Arguments and the Data Model</i>	91
4.3.2	<i>Automatic Generation of SQL Queries</i>	93
4.4	<i>Strategies Applied to the State Flow Model Assistant (SFMA)</i>	94
4.4.1	<i>Functionalities Included in the Graphical User Interface</i>	94
4.4.2	<i>Automatic State Proposals for Defining the Dialogue Flow</i>	97
4.4.3	<i>Automatic Unification of Slots for Mixed-Initiative Dialogues</i>	99
4.5	<i>Strategies Applied to the Retrieval Model Assistant (RMA)</i>	101
4.5.1	<i>Automatically Proposed Dialogues</i>	101
4.5.2	<i>Automatic Generation of Action Proposals in Each State</i>	103
4.5.3	<i>Automated Passing of Arguments between Actions</i>	105
4.5.4	<i>Mixed-Initiative and Over-Answering</i>	106
4.5.5	<i>Other Functionalities</i>	109
4.6	<i>Strategies Applied to the Modality Extension Retrieval Assistant for Speech (MERA-Speech)</i>	109
4.6.1	<i>Presentation of Object Lists</i>	110
4.6.2	<i>Confirmation Handling</i>	113
4.7	<i>Strategies Applied to Other Assistants</i>	115
4.7.1	<i>Modality and Language Extension Assistant (MEA)</i>	116
4.8	<i>Conclusions</i>	122
5	EVALUATION OF THE ACCELERATION TECHNIQUES	125
5.1	<i>Subjective Evaluation</i>	125
5.1.1	<i>Experimental setup</i>	125
5.1.2	<i>Evaluation results</i>	127
5.2	<i>Objective Evaluation</i>	132
5.2.1	<i>Experimental setup</i>	132
5.2.2	<i>Description of the evaluated tasks and results</i>	135
5.2.3	<i>Subjective survey</i>	144
5.3	<i>Conclusions</i>	147
6	DEVELOPMENTS AND IMPROVEMENTS APPLIED TO THE RUNTIME SYSTEM	149

6.1	Language Identification System	150
6.1.1	<i>System Description</i>	150
6.1.2	<i>Proposed Technique: n-gram Frequency Ranking</i>	154
6.1.3	<i>Incorporation of additional information</i>	163
6.1.4	<i>Conclusions</i>	169
6.2	Automatic Translation of Dialogue Prompts into the Sign Language	170
6.2.1	<i>Runtime System for the Speech-to-Sign Language Translation System</i>	171
6.2.2	<i>Bilingual Corpus</i>	172
6.2.3	<i>Speech Recognition Results</i>	174
6.2.4	<i>Statistical Machine Translation System</i>	174
6.2.5	<i>Proposed Adaptation Technique</i>	177
6.2.6	<i>Language Model Experiments</i>	181
6.2.7	<i>Machine Translation Experiments</i>	181
6.2.8	<i>Conclusions</i>	183
7	CONCLUSIONS AND FUTURE WORK	185
7.1	CONCLUSIONS	185
7.1.1	<i>Dialogue Platform</i>	185
7.1.2	<i>LID System</i>	187
7.1.3	<i>Machine Translation System</i>	187
7.2	FUTURE WORK	188
7.2.1	<i>Dialogue Platform</i>	188
7.2.2	<i>LID System</i>	190
7.2.3	<i>Machine Translation System</i>	190
	BIBLIOGRAPHY	193
	APPENDIX A. LIST OF ABBREVIATIONS	209
	APPENDIX B. ADDITIONAL INFORMATION ABOUT CURRENT COMMERCIAL AND WEB-BASED PLATFORMS	211
B.1	Commercial Platforms	211
B.2	Web-Based Development Platforms	219
	APPENDIX C. TEMPLATES FOR THE CREATION OF AUTOMATIC DIALOGUES IN THE MERA-SPEECH ASSISTANT	225
C.1	Template for the Presentation of Lists of Objects	225
C.2	One Slot Confirmation	229
C.3	Mixed-Initiative Confirmation	230
C.4	One Slot Plus Over-Answering Confirmation	232
C.5	Mixed-Initiative Plus Over-Answering Confirmation	235
C.6	Simple Confirmation and Basic Dialogues	237
	APPENDIX D. QUESTIONNAIRE FOR EVALUATING THE APPLICATION GENERATION PLATFORM	239
D.1	Specific Questions by Assistant	239
D.1.1	<i>Questions regarding the assistant:</i>	239
D.2	General Questions about the AGP	240
D.2.1	<i>Advantages of using the AGP</i>	240
D.2.2	<i>Do you learn quickly how to make applications with the AGP?</i>	241
D.2.3	<i>How do you rate the overall appearance of the AGP (consistent, transparent, and intuitive)?</i>	242

<i>D.2.4 Do you find the various assistants of the AGP are well integrated?</i>	242
<i>D.2.5 Do you think non-experts could use the AGP efficiently?</i>	242
<i>D.2.6 Would you use this system in the future or recommend it to develop speech/Web applications?</i>	242
<i>D.2.7 If yes, how much would you be willing to pay for its use?</i>	242
APPENDIX E. DETAILED RESULTS OF THE OBJECTIVE EVALUATION OF THE PLATFORM	243

INDEX OF FIGURES

Figure 2.1. Appearance of some tools provided by the IBM WebSphere Toolkit. (Source: IBM WebSphere home page).....	10
Figure 2.2. Example of creation of a Speech Grammar using the Microsoft Speech Application SDK.....	15
Figure 2.3.CSLU's RAD Toolkit: a) Example of the main canvas, available objects, and dialogue states definition. b) Example of possibilities using the included animated agent: Baldi. (Source: CSLU Toolkit home page).....	16
Figure 2.4. Detailed graphical presentations of a dialogue model using DialogDesigner (Source: DialogDesigner Web page)	17
Figure 2.5. Diagram of a PPRLM LID system (Source: [Zissman, 1996]).....	39
Figure 2.6. The Vauquois triangle (Source: [Jurafsky and Martin, 2008]).....	42
Figure 2.7. Translation process based on Bayes decision rule	44
Figure 2.8. Example of SignWriting notation for the Spanish sign: Book (Source: [Parkhurst and Parkhurst, 2007]).....	55
Figure 2.9. Example of HamNoSys notation and its representation using the avatar.	55
Figure 3.1. Platform architecture	57
Figure 3.2. Graphical details of a class and its attributes, and code fragment generated for the Transaction class.....	60
Figure 3.3. Form used to define the prototype of a database access function	61
Figure 3.4. GDialogXML code generated by the SFMA.....	62
Figure 3.5. Code generated by the RMA for the bank transfer example.	64
Figure 3.6. Example of the definition of a TTS prompt using SSML tags.....	67
Figure 3.7. GDialogXML code generated by the MEA for the speech modality.....	68
Figure 3.8. Process for creating a dialogue in GDialogXML using the Diagen assistant	73
Figure 3.9. Offline and Online process for creating and using sign language prompts.....	75
Figure 3.10. Example of process to design and play a sign with VGuido.....	76
Figure 3.11. Distributed architecture for the runtime system	79
Figure 3.12. Procedure to retrieve information from the database using the runtime system .	80
Figure 3.13. Example of a SRGS grammar file used by the NLU module in the run-time system	81
Figure 4.1. Form fill-in window that allows the creation of custom classes (from the database and classes from the current model) in the DMA.	90
Figure 4.2. Example of the automatic creation of a referenced class	91
Figure 4.3. GDialogXML code generated by the DCMA for the bank transfer.....	92
Figure 4.4. Form fill-in window for the automatic creation and testing of SQL queries for database access functions.....	93

Figure 4.5. Appearance of the SFMA main window	95
Figure 4.6. Process for the creation of a 1:N transitions in the SFMA.....	96
Figure 4.7. Pop-up window with states proposals from classes defined in the data model structure (DMA).....	97
Figure 4.8. Example of a proposed state from a defined database access function. The GDialogXML code corresponds with the definition of the function in the DCMA.....	98
Figure 4.9 Configuration window for creating or editing rules for automatic detection of directed or mixed-initiative dialogues	99
Figure 4.10. Example and GDialogXML code for two slots automatically unified for mixed-initiative	100
Figure 4.11. Auxiliary screen of the RMA and popup window for dialogue configuration..	102
Figure 4.12. Example with automatic dialogues and database access function proposals	104
Figure 4.13. Form fill-in windows that automate the process of passing arguments between actions	105
Figure 4.14. Example of the GDialogXML syntax for a mixed-initiative dialogue created in the RMA.....	107
Figure 4.15. Example of the creation of a mixed-initiative dialogue	108
Figure 4.16. GDialogXML code for a dialogue to ask for a single slot and define another one as optional using over-answering.....	108
Figure 4.17. Tooltips functionality for a quick description of all internal actions	109
Figure 4.18. Example of the assistant window for configuring a DSay dialogue for the presentation of objects lists (case 3).	111
Figure 4.19. Assistant window for copying prompts.....	116
Figure 4.20. Examples of wording and prompt library files.....	117
Figure 4.21. Additional languages prompts setting window.	118
Figure 4.22. Example of the definition of a grammar rule using the Language Modelling Toolkit.....	119
Figure 4.23. Example of full generation of possible sentences from a JSGF file.....	120
Figure 4.24. Assistant for the creation of stochastic language models.....	121
Figure 5.1. Final distribution of experience status for the evaluation participants of the subjective test.....	126
Figure 5.2. Evaluation results for the question about how quick the participants learnt to use the platform assistants.....	128
Figure 5.3. Evaluation results for the question about how easy and intuitive were the platform assistants	128
Figure 5.4. Evaluation results for the question about how sufficient was the functionality of each platform assistant.....	129
Figure 5.5. Evaluation results for the question about how consistent, transparent, and intuitive the users rated each platform assistant.....	129

Figure 5.6. Average results of the subjective evaluation for general questions about the assistants	130
Figure 5.7. Interface used to start the evaluation of the different assistants in the AGP and using the Diagen assistant.....	133
Figure 5.8. Interface to record the mouse and keyboard events during the evaluation	135
Figure 5.9. Chart with the improvements obtained when evaluating the Data Model Assistant for the creation of a class structure with database dependency	136
Figure 5.10. Chart with the improvements obtained when evaluating the Data Model Assistant for the creation of a mixed class.....	137
Figure 5.11. Chart with the improvements obtained when evaluating the Data Connector Model Assistant for the definition of the prototype of a database function.....	137
Figure 5.12. Chart with the improvements obtained when evaluating the State Flow Model Assistant for the creation of a state with one slot	138
Figure 5.13. Chart with the improvements obtained when evaluating the State Flow Model Assistant for the creation of a state with mixed initiative slots and one transition.....	139
Figure 5.14. Chart with the improvements obtained when evaluating the State Flow Model Assistant for connecting two states.....	139
Figure 5.15. Proposed flow for the evaluation of a menu-based dialogue in the RMA	140
Figure 5.16. Chart with the improvements obtained when evaluating the Retrieval Model Assistant for creating a menu-based dialogue	140
Figure 5.17. Proposed flow for evaluating a dialogue with aver-answering and conditional actions	141
Figure 5.18. Chart with the improvements obtained when evaluating the Retrieval Model Assistant for creating a dialogue with over-answering and a conditional structure	142
Figure 5.19. Proposed flow for a dialogue with mixed-initiative and the creation of a global variable.....	143
Figure 5.20. Chart with the average improvement by assistant considering all tasks	144
Figure 5.21. Chart with the results of the subjective evaluation for general questions about the assistants	145
Figure 6.1. PPRLM scores used for the LID system	151
Figure 6.2. Example and calculation of distance score using a ranking of n-grams as proposed by [Cavnar and Trenkle, 1994].....	155
Figure 6.3. Example of the modification of a ranking template using the “golf score”	156
Figure 6.4. LID error rate results varying the ranking size and using the ‘golf’ ranking	156
Figure 6.5. Accumulative LID error rates reductions over the original ranking technique... ..	162
Figure 6.6. Example of the procedure to create the vector with the acoustic score for each phoneme.....	164
Figure 6.7. LID error rate results and confidence intervals considering the $PPRLM_{NG}$, $PPRLM_{RANK}$, and the fusion of both systems	169
Figure 6.8. Spoken Language to Sign Language translation system	171

Figure 6.9. Example of an alignment template and phrase alignments for a Spanish to Spanish Sign Language sentence pair	175
Figure 6.10. Flow diagram of the proposed adaptation technique.....	178
Figure B.1. Audium Builder main window. (Source: [Scholz, 2006]).....	212
Figure B.2. Avaya Dialogue Designer. (Source: Avaya product brochure available at the corporate website).....	213
Figure B.3. Example of dialogue design using the OpenVXML toolkit. (Source: OpenVXML Web page).....	215
Figure B.4. Example of wizards included in the OptimTalk Professional Edition Toolkit. (Source: OptimTalk Desktop Suite Web page)	216
Figure B.5. Example of dialogue design using the Vocalocity App Center. (Source: Vocalocity Web page).....	218
Figure B.6. Appearance of the main work area of the VoiceObjects Desktop. (Source: VoiceObjects Web page)	219
Figure B.7. Appearance of different TellMe Studio tools. (Source: TellMe Studio Web site and TellMe Voice Studio user guide)	221
Figure B.8. Examples of Voxeo Evolution Web-based tools. (Source: Voxeo Studio Web page).....	223

INDEX OF TABLES

Table 4.1. Confusion matrix for automatic field types detection comparing the human classification (real type) and the proposed type by the system (RE type).....	87
Table 4.2. Confusion matrix comparing the human classification (real type) and the driver classification (driver type)	88
Table 5.1. Distribution of the evaluation participants for the subjective test	126
Table 5.2. Subjective evaluation of the platform.....	131
Table 5.3. Subjective evaluation results for specific questions about the accelerations.....	146
Table 6.1. Differential score vector	153
Table 6.2. LID error rate results for PPRLM _{NG}	153
Table 6.3. Example of an n-gram specific count-based ranking for English.....	157
Table 6.4. Ranking size for the different n-grams and languages	158
Table 6.5. LID error rate results with n-gram specific ranking	158
Table 6.6. Comparison of feature discrimination between PPRLM _{NG} and PPRLM _{RANK}	159
Table 6.7. Average feature discrimination (several formulas)	160
Table 6.8. Comparison of feature discrimination between PPRLM _{NG} and discriminative PPRLM _{RANK}	161
Table 6.9. LID error rate results for PPRLM _{NG} versus discriminative PPRLM _{RANK}	162
Table 6.10. LID error rate results for including incrementally long-span information	162
Table 6.11. Comparison of LID error rate results for including the sentence acoustic score to the PPRLM _{NG} and the discriminative PPRLM _{RANK} systems.....	163
Table 6.12. LID error rate results for individual feature vectors.....	166
Table 6.13. LID error rate results for feature vector combinations	166
Table 6.14. LID error rate results using the Multi-Gaussian classifier.....	168
Table 6.15. Corpus statistics summary	173
Table 6.16. Speech recognition results.	174
Table 6.17. Example of n-grams in the phrase translation table.....	180
Table 6.18. Perplexity (PPL) results using the corresponding LM.....	181
Table 6.19. Average Machine translation results for the test set (Exp 1-3).....	182
Table C.1. Proposed actions for the automatic filling of one slot and over-answering DGet dialogues in the MERA-Speech assistant	232
Table C.2. Proposed actions for the automatic filling of mixed-initiative and over-answering DGet dialogues in the MERA-Speech assistant.....	235
Table E.1. Quantitative measures obtained during the objective evaluation.....	243
Table E.2. Results for general questions about the assistants in the AGP during the subjective evaluation.....	246

1 INTRODUCTION

1.1 Motivation

The growing interest from companies in using new information technologies as a means to getting closer to the final users has led to the quick growth and improvement of automatic dialogue systems for database search tasks. In these systems, users interact with an automatic system to retrieve or exchange information that is available in a backend database. In this way, it is possible to provide services such as reservations [San-Segundo et al, 2001a][López-Cozar and Granell, 2004][Lamel et al, 2000][Levin et al, 2000], customer care [Strik et al, 1997], information retrieval [Zue et al, 2000][Seneff and Polifroni, 2000], interactive voice response systems, etc. 24 hours a day 7 days a week.

One of the main difficulties of these systems is the process of designing them in a fast and flexible way, so that the time needed by the designer to design the service, and the time that it will take to the user to obtain the desired information in the real-time system can be both reduced. In addition, given the different characteristics and requisites of the final users, the service is expected to be available for several languages [Turunen et al, 2004][Uebler, 2001] and input/output modalities such as speech, Web, interactive maps, gestures, tactile screens, animated agents, etc.([Almeida et al, 2002][Gustafson et al, 2000][Oviatt et al, 2000]) for allowing users from different nationalities and physical abilities to have access to the service. Besides, it is expected that the same design can be reused for new languages and modalities with minimum modifications and without requiring too much expert knowledge.

Fortunately, the increasing demand of automatic dialogue systems have resulted in several companies and academic institutions working in the development of fully integrated platforms that necessarily have to provide the maximum number of features to the designer and the final users, a high level of portability, standardization and scalability in order to minimize design time and costs. Moreover, these platforms have to enable the rapid development, maintenance, and deployment of automatic dialogue services, as well as to be flexible enough to allow the creation of a wide range of services and to be adapted to the special characteristics of each one. In general, these platforms are made up of different and independent modules allowing collaborative role-based development, so that different teams of developers can work on the same project at the same time. Through these modules, the designer can specify for instance: the application flow, grammars and system's prompts, actions for error handling, integration with backend databases, etc. Besides, they also include debugging and edition modules to improve the service (for instance loggers, call/flow analyzers, grammars and vocabularies generators, etc.), built-in components such as dialogue libraries, grammars, and prompts for common situations (e.g. for requesting a phone number, an address, names, etc.), etc. Finally, the usability of such platforms is increased thanks to a clear and fully integrated graphical user interface, as well as to built-in libraries and dialogue components that accelerate the design and simplifies to reuse previous knowledge.

However, in spite of all the advantages provided by current platforms, it is surprising to observe that, in general, all of them share the same kind of accelerations to the design. For instance, most of the accelerations offered by these platforms rely on the possibility of using configurable built-in libraries or dialogues modules for common situations. Unfortunately, they lack of some kind of accelerations based on basic business intelligence and data mining

methodologies applied to the contents of the task database and from the data model structure (i.e. the set of object-oriented classes and attributes that model the database tables and fields and their relationships). To cope with this, our objective is to use these dynamic and intelligent acceleration strategies so that we can, among other things, predict the necessary information required to complete the definition of a dialogue state, accelerate the specification of the application flow and the definition of the database access functions, and help designers with built-in solutions for presenting lists of objects (generated after executing a database query) for the speech modality, not forcing designers to define this information from scratch. Additionally, most of these platforms do not offer the possibility of creating the same service for users with disabilities, e.g. deaf users, or present reduced capabilities for designing the same service for different modalities at the same time.

Taking into account the limitations of the best commercial platforms and the limited number of research projects for creating, accelerating, and improving such design platforms, and based on the results and experience obtained in previous projects [Cordoba et al, 2001][Lehtinen et al, 2000], we undertook the European Project GEMINI¹ (Generic Environment for Multilingual Interactive Natural Interfaces) developed from 2002 to 2004. The result was a complete, flexible, and highly automated development platform that consists of a set of tools and agents that guide the design process and allow the definition of the different levels of knowledge needed to complete and run state-of-the-art speech and Web-based services. Then, after finishing the project, we decided to continue working on the development platform in order to propose new accelerations strategies and improving the capabilities of the final version generated during the project.

In this thesis, we will describe in detail the main strategies applied to the different assistants that make up the platform in order to speed up the design process, as the possibility of handling mixed-initiative and over-answering dialogues using the same framework. Detailed procedures to handle the presentation of lists of objects and confirmation handling for the speech modality will be presented too. Features like user modelling, speaker verification, language identification can also be included easily through runtime modules included in the platform, although several accelerations to these assistants and modules are left for future work. On the other hand, the generation of the runtime scripts of the dialogue using standard languages like VoiceXML and xHTML kept the platform open for further development and to the possibility of extending its capabilities with third party tools and technology. However, and even most important, our work also allowed us to contribute to the development of these standard languages and to overcome some of its limitations. Another important result of the GEMINI project was a newly designed abstract dialogue description language called GDialogXML, to which we have also contributed during this thesis. Finally, we carried out an objective and subjective evaluation that demonstrated the user and designer-friendliness and robustness of the platform, as well as the fulfilment of all the objectives initially planned, together with a proposal for improvements and plans for the platform.

On the other hand, the wide range of final users of the service creates diverse challenges such as the capability of the system to properly identify the language to be used to communicate with the users, or the possibility of providing the service using different modalities according to the user preferences or needs, or to the state of the dialogue. In general, this kind of functionalities are not totally considered in most of the current platforms

¹ <http://www-gth.die.upm.es/projects/gemini/>

although they are, in real world applications, essential for the correct execution of the service and to provide a better user experience.

In this thesis, we will describe the improvements made to a language identification system through the integration of an interesting technique based on long-span language models and to an automatic speech-to-sign-language translation system through an innovative adaptation technique that improves the quality of the translated sentences. In the former case, we will describe the creation and use of a discriminative ranking of phone-based n-grams that allows the LID system to use higher-order models and reduces frequent problems that appear in the estimation of most statistical-based language models. In the latter case, we will explain the process of accelerating and improving the translations of the written/spoken prompts of a given service into an animated representation in the sign language for deaf people. In this case, our main contribution is the adaptation of the language models used to increase the quality of the translated sentences when there is not enough training data to obtain a reliable statistical-based language model. Finally, we will also show the evaluation results of both techniques.

1.2 Objectives

The most important objectives of this thesis are focused on the study and integration of innovative strategies applied to simplify and speed up the design process in a unified environment with multimodal and multilingual capabilities, as well as the development and improvements on the different modules that make up the runtime system. In detail, we will pursue the following goals:

1.2.1 Design platform

The main objective in relation with the design platform is to propose different acceleration strategies to the main assistants of the development platform, through the incorporation of heuristic information extracted from the backend database and the data model structure, and by sharing information among assistants. This objective involves the fulfilment of the following sub-objectives:

- To propose accelerations that can help in reducing the overhead produced by performing repetitive or common procedures in the design. In this case, we propose to accelerate the passing of arguments between actions, the definition of prompts/grammars, the semi-automatic generation of SQL statements to access the backend database, and the definition of the system behaviour for error handling in the speech modality.
- To evaluate the different proposed accelerations and acceptability of the platform through a subjective and objective evaluation. In this case, the evaluations have to demonstrate the contribution of the proposed accelerations to reduce the design time and to simplify the design process, as well as to demonstrate the designer-friendliness of the proposed platform and assistants.

1.2.2 Language Identification System

In this case, the objectives we will pursue will be the following ones:

- To study of the incorporation of a long-span language model as phonotactic constraints for a language identification system based on PPRLM, which reduces the problem of having a reduced training set to obtain reliable high-order language models.
- To study the integration of long-span, acoustic, and duration information as input features for a Gaussian classifier, evaluating the contribution and discriminative power of each one.

1.2.3 Machine Translation

Regarding the automatic machine translation system, we propose the following sub-objectives:

- To include a new modality to the runtime platform extending the functionalities of the platform allowing that the same service can be provided to a handicapped user.
- To study the viability of a new language model adaptation methodology that can be used to improve the quality of the sentences generated by a machine translation system.

1.2.4 Relevant Definitions

Throughout this thesis, we are going to use some terms that do not necessarily have the same meaning as the ones used in common literature or that do not present a general accepted definition. To clarify them and avoid confusions we want to define them here from the perspective of our platform.

- **Designer and user:** The term designer will refer to the person that uses the platform to build the service, and user will refer to the final client of the developed service.
- **Mixed-initiative and over-answering:** It is well known that the concept of mixed-initiative includes over-answering, as mixed-initiative is a generic term used to refer to a flexible interaction between the user and the system to get together to reach a common final solution [Allen et al, 1999]. However, we preferred to differentiate them to maintain the consistence with the specifications and implementation of the VoiceXML² standard. In this sense, we will use the term mixed-initiative to indicate the system's ability to ask simultaneously for two or more compulsory data from the user, and, if the user's answer is incomplete—or the recognizer fails—new subdialogues are started to obtain the missing data. With over-answering, we indicate the user's ability to provide additional data—not compulsory at that state—to the system.

² <http://www.w3.org/TR/voicexml20/>

- **Multiple modalities:** The common usage of the term multimodality in dialogue applications refers to the ability to support the communication with the user through several channels to obtain and provide information [Nigay and Coutaz, 1993]. The most widely used modalities are voice, gestures, mouse, images, or writing, which can be combined simultaneously or otherwise during the dialogue. In our platform, we have focused on applying this term from the designer point of view, referring to the platform's ability to generate the service for two modalities in a unified and simultaneous way: Web and voice. Right now, these modalities work apart from each other instead of being combined (synchronized) in the real-time system.
- **Dialogue and state and action:** From the terminology established by the W3C for an event-driven model of dialogue interaction³, we can find the following definitions:
 - **Dialogue:** “a model of interactive behaviour underlying the interpretation of the markup language. The model consists of states, variables, events, event handlers, inputs and outputs”
 - **State:** “the basic interactional unit defined in the markup language . . . A state can specify variables, event handlers, outputs, and inputs”.

In spite of the differences in these definitions, throughout the thesis we will use both terms with very little difference, as they will refer, from the perspective of a finite state machine, to each interaction with the user—or a set of them—needed to fulfil a service task. Nevertheless, the term dialogue will be more associated to the interaction with the user, whereas state will mostly refer to a set of interactions and other additional actions, such as a database access.

- **Action:** This term will refer to each procedure needed to complete a state or a dialogue, for example calls to other dialogues, arithmetic, or string operations, programming constructs, variable assignments, etc.
- **Slot:** This term will refer to each piece of compulsory information that the system has to ask the user in order to offer the service.
- **Acceleration:** This term will refer to any methodology implemented into the different assistants of the platform in order to reduce the design time and to make easy the definition of the different dialogues, actions, and elements required to design and run the service.

1.3 Organization

The thesis is organized as follows. Chapter 2 is divided into four sections. The first one presents the state-of-the-art on development of dialogue applications, including descriptions of several commercial and academic platforms, as well as different kinds of acceleration strategies included in them for designing the service. The second section describes the state-of-the-art on language modelling, including information about the most common strategies to train and improve the statistical-based language models. The third section describes the state-

³ <http://www.w3.org/TR/voice-dialogue-reqs/>

of-the-art on language identification techniques with special emphasis on the PPRLM system. Finally, the fourth section describes the state-of-the-art on machine translation systems; In this case, several strategies are described, with emphasis on the statistical approaches. This section also provides a brief description of the phrase-based translation approach and current metrics for the evaluation of machine translation systems. Finally, an overview on important aspects and research about speech-to-sign language translation are also presented.

Chapter 3 describes the overall architecture of the development platform proposed in this thesis, including detailed information about the assistants that make it up, the runtime system, the internal XML language used to share information among the assistants and create the service, as well as a description of the platform scope and main limitations.

Chapter 4 describes the main acceleration strategies applied to the different assistants in the platform. In chapter 5, we will show the results of an objective and subjective evaluation of the whole platform, the assistants, and the proposed acceleration strategies.

Chapter 6 shows the development and improvements applied to the runtime system in order to allow the multimodality and multilinguality capabilities of the platform. The first improvement consists of the creation and incorporation of a new long-span language model based on using a N-gram frequency ranking, as well as the study of combining the proposed technique with additional information, mainly acoustical and durations, to a state-of-the-art language identification system based on the PPRLM technique and using as final backend a Gaussian classifier. The second improvement is the creation of a new online adaptation technique that improves the quality of the translated sentences of an automatic machine translation system that translates system prompts into an animated representation in the Spanish sign language in order to provide the service to users with hearing disabilities.

Chapter 7 presents a list of future improvements to the platform and the main conclusions, followed by the complete bibliography used in the thesis. Appendix A provides a small list of abbreviations used in the thesis. Appendix B provides detailed information about the characteristics and accelerations included in several commercial and Web-based development tools. Appendix C describes the templates used in the assistant that creates the flow for the presentation of lists of objects after retrieving information from the backend database and for handling user confirmations. Appendix D describes the questionnaire used for the subjective evaluation. Finally, Appendix E includes detailed tables with the results of the objective and subjective evaluation done to the development platform and the accelerations described in this thesis.

2 STATE-OF-THE-ART

This chapter describes the platforms, tools, algorithms, and methodologies we have studied and used to accomplish the objectives of the thesis. The chapter is divided into four sections representing the different topics tackled in this dissertation.

The first section presents detailed information about current commercial and non-commercial platforms that allow the design, debugging, and execution of automatic dialogue services. Information about different kind of methodologies used for accelerating the design process is also presented.

The second section describes the main algorithms and methodologies for training, adapting, and improving language models. From the perspective of the present thesis, this information is relevant since they establish the foundations to explain the techniques we have applied to improve the language models used by the language identification system (LID) and the automatic translation system. The former is used to detect the language to be used by the system to communicate with the user, and the latter is used to translate system prompts (i.e. text or speech messages presented to the final user) into an animated representation in the sign language in order to allow deaf users to use the developed service.

The third section shows the most important algorithms and research lines for language identification. Among the reported algorithms, the Parallel Phone Recognition followed by Language Modelling (PPRLM) is the most successful and widespread technique. For that reason, and because it was used as our baseline system, we will describe it in more detail.

Finally, the last section presents the main methodologies and algorithms used for training, evaluating, and using statistical machine translation systems. The section makes emphasis on current research systems for translating text/speech into Sign Language, and in the phrase-based translation methodology used by most research systems.

2.1 Development Platforms and Acceleration Strategies for Designing Multimodal Dialogue Systems

This section describes the most important platforms and tools that were studied and compared with our design platform. In this study, a distinction between systems developed for research and for commercial purposes was done. This classification is important because, in general, platforms developed for commercial purposes present a clear and elegant interface that reflects the big effort companies usually make in this respect. In addition, these platforms make use of several standards in order to provide flexibility and to simplify sharing files across platforms. However, their big disadvantage is that most of them just allow the creation of services for one or two modalities such as speech and Dual-Tone Multi-Frequency (DTMF), but do not provide support for other modalities such as animated agents, Web, touch screens, remote controls, etc. On the other hand, non-commercial platforms do not provide many of these characteristics but present more capabilities for integrating different modalities and languages at the expense of making the design process slower and difficult to mimic for other systems. In addition, this distinction is also important because if not taken

into account it would be a difficult task to compare and to extract conclusions from all these platforms.

The following sub-sections provide a brief description of the main features included in most of the commercial and research platforms. Since the number of modalities, language specification formats, and architectures is too high, we have focused in platforms that allow the creation of VoiceXML-based applications and speech grammars. For further information, about these and other development tools, environments, and modalities, please check the Web address mentioned for each one, or refer to [López-Cozar and Araki, 2005] and [McTear, 2004].

2.1.1 Commercial Platforms

In order to summarize the main efforts done by most of the current commercial platforms to accelerate the design of multimodal and multilingual dialogue applications, we can say that they include state-of-the-art modules such as speech recognizers, high quality speech synthesizers, language and speaker identification capabilities, and several other high-level tools, that allow the creation of very complex and advanced dialogue services. In addition, these platforms support the creation of the service using widespread standard languages and protocols such as VoiceXML, SALT [Wang, 2002], X+V⁴, J2EE, xHTML, Voice Browser Call Control XML (CCXML)⁵, etc, to guarantee the integration between different vendors and platforms. Besides, these platforms are often supported by advanced hardware modules, which can be used with minimum programming effort and adapted easily to the runtime system. These platforms also include a high number of predefined libraries for typical dialogue states such as requesting card or social security numbers. In addition, they incorporate assistants for debugging, logging, and simulate the service. Finally, they present a very friendly graphical user interface that simplifies the development of very complex dialogues.

Below, we provide an overview of the main features and accelerations included in three of the most widely known commercial platforms. In this case, we describe the IBM Websphere, Nuance, and SpeechDraw development platforms. In Appendix B we have included detailed information regarding other platforms such as the ones offered by Audium, Avaya, Genesys, Envoy, Vocalocity, VoiceObjects, among others.

IBM Websphere Voice Platform⁶: This platform is a complete commercial application for developing, setting up, and debugging VoiceXML and CCXML applications. The platform requires the installation of several packages in order to design and run the service, for instance IBM WebSphere Application Server, IBM Rational Application Developer (RAD), IBM WebSphere Voice Server, and IBM WebSphere Voice Toolkit⁷.

The IBM Application Server allows the system administration of the WebSphere Voice Server. The IBM RAD is an integrated development environment (IDE) for designing, testing, and deploying Web or speech services, with support for backend database connections, among others. The development of dynamic VoiceXML applications is accelerated using the J2EE platform and through two basic wizards for creating JSP/Servlets:

⁴ <http://www.voicexml.org/specs/multimodal/x+v/12/>

⁵ <http://www.w3.org/TR/ccxml/>

⁶ <http://www-01.ibm.com/software/voice/>

⁷ We want to thank IBM for letting us use their platform for evaluation purposes

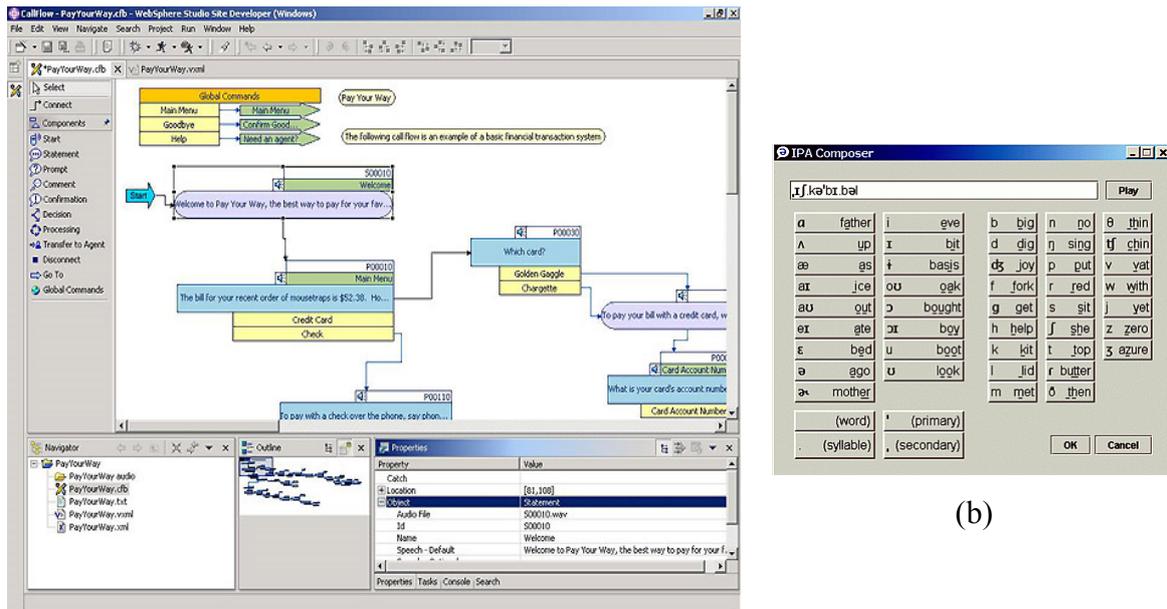
one for Database Web pages and another for Java Bean Web Pages. In addition, the IBM Voice Server provides the middleware to allow the service to be accessed from a telephone, cell phone, or Web browser. The Voice Server includes software for speech recognition and Text-To-Speech in several languages, as well as other development tools to support applications written in VoiceXML. The platform supports the dynamic modification of the TTS using the SSML (Speech Synthesis Markup Language) specification and Speaker Verification features.

In addition, the IBM Voice Toolkit includes several tools to build, debug, and deploy the dialogue flow through an intuitive graphical environment. The Voice Toolkit allows the creation and testing of grammars, pronunciation dictionaries, and natural language understanding models (NLU). In addition, the toolkit makes possible to generate reports and obtain service metrics based on the analysis of the call flow.

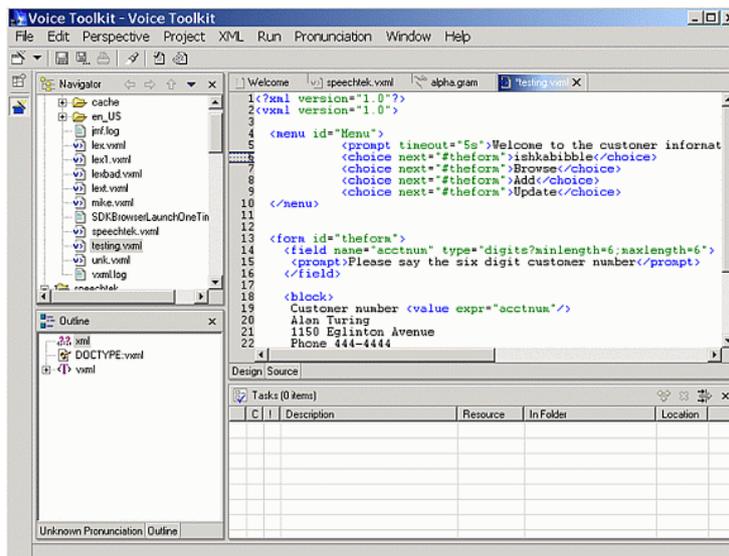
In detail, the IBM Voice Toolkit provides the following tools and features:

- The Communication Flow Builder that makes possible to simulate and debug the service using the graphical environment, or using a SIP-phone and a MRCP server. The graphical interface allows traditional debugging features such as breakpoints, step-by-step walkthrough, variable inspection, and modification of any variable on the fly as the program is debugged.
- The Visual Grammar Builder is a MRCP-based tool that allows the creation and testing of JSGF, BNF or XML-based grammars and pronunciation lexicons using the IBM WebSphere Voice Server. This tool can also be used to detect words that cannot be recognized within the grammar, to convert between grammar formats, or to generate all the sentences that it is possible to recognize given a grammar file (similar to the assistant described in section 4.7.1.3, page 120). It is also useful to test grammar files using text/speech based utterances, providing debugging information such as semantic interpretation results, n-best matches, and confidence scores. Finally, it also includes a tool for CCXML edition, validation, formatting, and preferences management.
- The Prompt Manager tool allows the organization, edition, and recording of audio files.
- The Voice Trace Analyzer allows the analysis of log files from the Voice Server.
- The toolkit includes tools for the development of Natural Language Understanding (NLU) models. In this case, this tool supports the creation of different types of statistical models and grammars in SRGS (Speech Recognition Grammar Specification) format, the classification of the data used to train the models, and the possibility of allowing multiple developers to work in parallel with the same training data. The toolkit allows setting up a DB2 database for the development of the NLU models. This feature is used to import/export XML-formatted data into/out-of the database, as well as the possibility of making searches in the database and navigating through the results, the possibility to reclassify large training data, and the validation of NLU models.
- Finally, the toolkit includes pre-written and working code, called Reusable Dialogue Components (RDC), which can be copied, edited, and incorporated into the VoiceXML file through a configuration wizard that helps in selecting and customizing these RDC. The use of these components is highly recommended since they reduce development time and contribute to the learning process. Among the

RDCs included are spelling, browsable selection list, confirmation, currency, date info, postal code, telephone number, URL, airports, stocks, time or date range, billing application, shopping cart shipping information, etc. These libraries include predefined prompt messages, error-handling actions, suggestions to spell an input or discard it, SRGS grammars, and predefined help information, which can be used with/without modifications. Two types of RDC components are allowed: Subdialogues and Templates. Subdialogues allow the designer to create dialogues that request a single slot to the final user (e.g. the name of a person), whereas templates are predefined dialogues where several slots are requested to the user (e.g. the name, the address, the zip code, etc.).



(a)



(c)

Figure 2.1. Appearance of some tools provided by the IBM WebSphere Toolkit. (Source: IBM WebSphere home page)

Figure 2.1 shows a) the appearance of the Communication Flow Builder, included in the Voice Toolkit, which is used to develop the dialogue flow represented as a flowchart diagram, with nodes, transition and decision points. b) a pop-up window for including pronunciations using International Phonetic Alphabet (IPA) symbols. c) the built-in VoiceXML and Call Control Extensible Markup Language (CCXML) editor, which features a context-sensitive auto-completion, colour coding, drop-down menus, and validation capabilities, for allowing fine-tuning of the applications.

Nuance Voice Platform⁸: Compared to other commercial platforms, this is one of the most complete development environments currently available. In addition to the big number of features included in the platform, it also includes some of the most advanced speech recognition, text-to-speech synthesiser, and speaker identification engines available at the market. The platform provides an off-the-shelf solution to design, deploy, and monitor the service. The Nuance Voice Platform consists of four main components: Nuance Conversation Server, Nuance Management Station, Nuance Application Environment, and Nuance CTI Gateway⁹.

The Conversation Server enables the caller to interact with the application using a built-in VoiceXML browser, the Nuance speech recognition, text-to-speech, and verification engines. These engines run as services and can be started, stopped, and monitored separately.

The Management Station allows the designer to remotely manage, analyze, and tune the service, as well as to check the status of the servers that run the service, through a centralized Web-based graphical interface.

The Nuance CTI (Computer Telephony Integration) Gateway provides a Web-based interface that allows the integration and control of the Nuance platform with third-party CTI servers, from leading vendors such as Cisco, Avaya, or Genesys. The CTI Gateway includes different plug-ins to translate API requests into commands supported by the server.

Finally, the Application Environment provides the graphical user interface for the development and deployment of the voice service. It consists of two applications: Nuance V-Server and Nuance V-Builder. The former allows the integration with backend databases and CTI servers, controls the execution of transition rules that allow an application to switch between dialogue states, and acts as gateway interface with the Management Station in order to monitor and control the service. On the other hand, the V-Builder is managed and executed at runtime using the V-Server, and it is used to create the VoiceXML application graphically. To do so, the V-Builder allows the creation of the application flow using a palette of objects and specifying their properties. After creating the flow, the V-Builder automatically generates the underlying VoiceXML code, without requiring the developer to know VoiceXML. This is also the aim of our assistants. In addition, it includes a complete set of tools for project management, document edition, grammars and prompts edition, application debugging, prompt playback, and a large number of ready-to-use grammar libraries to accelerate the design.

As previously stated, Nuance Inc. is the owner of some of the most advanced commercial engines for speech recognition, synthesis, language identification, and verification, the platform takes advantages of the main key features allowed for them, in order to improve the quality of the service and increasing the user satisfaction in the

⁸ <http://www.nuance.com>

⁹ We want to thank Nuance for letting us use their platform for evaluation purposes

interaction. Among the most relevant features included by these modules we can mention the following ones:

- The automatic detection of start and end of speech with echo cancellation to improve barge-in and recording of messages.
- The recognition of Hot-Words (i.e. word or phrase spotting), which are useful to improve the barge-in feature because the prompt is not stopped until a successful recognition occurs (i.e. prompts are not stopped due to noises or sentences with non-sense).
- The incorporation of Skip-lists, i.e. an array of previous bad recognised words/sentences, is used to avoid the system to confirm a previous misrecognized word or sentence with an inadequate format.
- Call-logs allow the creation of customizable reports of the service, including information about task completion, latency, dialogue status and returning values, CPU load, speed of the speech recognition, baseline recognition, accuracy, etc., which can be used to improve and tune the service.
- Context-files are XML documents that allow an expert to tune the VoiceXML service without requiring a complex knowledge of the specification. These files are used to set confidence values, timeouts, n-best properties, etc. specific to each dialogue state.
- Voice-enrolled grammars allow the creation of a grammar file and pronunciation dictionary using the ASR engine. This way, the process of adding sentences to the grammar or words to the vocabulary is accelerated.

The V-Builder also includes more than 100 different pre-built and ready-to-use grammars, as well as a grammar debugging tool that is useful to determine the grammar coverage, interpretation results, ambiguity (i.e. with multiple interpretations), detection of words with unknown pronunciations or misspelling, and the verification that the grammar does not accept the recognition of unwanted sentences. Finally, V-Builder includes a set of pre-defined java-classes called SpeechObjects that provide reusable dialogues to accelerate the design. A typical SpeechObject includes pre-recorded prompts, error management, and default confirmation handling, as well as some default actions such as playing a prompt, recognizing speaker input, interpreting and processing the recognition result, and returning a result. These actions and default properties are configurable by the designer according to the requirements of the service.

Speechdraw¹⁰: It is another interesting IDE that can be used for developing from basic up to very complex dialogue applications such as “How may I help you”. The responsible of the platform development claims that it is a zero programming interface, i.e. designers are not required to be experts on VoiceXML at all. The platform allows, among others, the creation of mixed-initiative speech applications, the design of speech recognition grammars, the automatic generation of documentation reports including detailed information about the flow, prompts, and grammars. The platform automatically checks the application looking for design errors, and points out where the errors appear and suggests solutions for them. In addition, the platform provides mechanisms to enable different designer profiles (i.e. flow

¹⁰ <http://www.speechvillage.com/home/>

design, grammar and backend design) to work on the same project, this way the development is centralized and the mismatch between designer teams is minimized.

On the other hand, the platform allows database emulation simply entering data into precompiled tables and defining the parameters that must be sent and returned when communicating with the backend. In this way, the platform avoids access delays and a strong interdependence between the designer interface and the integration layer when debugging the application. The platform incorporates a graphical flow editor that consists of two layers: application logic and error recovery logic. Using the application logic the designer specifies the dialogue flow with its corresponding states, actions, and transitions. On the other hand, in the error recovery layer the designer specifies the actions and prompts that control the system behaviour against the different kind of errors that could appear when interacting with the user or providing the service. An important acceleration included in this assistant is that the error recovery is automatically drawn using pre-defined rules specified by the designer. Another contribution of the platform is the use of sub-diagramming in order to reduce and keep readable the dialogue flow view readable when developing very complex services.

Finally, the platform incorporates an interesting acceleration for the design of prompts and grammars. In this case, the platform uses a pane that shows all the prompts and pre-recording messages reflecting the call flow structure. Through this pane, the designer can edit and visualize the prompts without wasting time going through the different states and actions of the dialogue. In addition, the pane also provides access to syntax checking, grammar and pronunciation development, debugging and grammar compilation, parsing of input sentences, and the visualization of keys and values of semantic tags.

2.1.1.1 Web-based development tools and portals

In addition to the previously described platforms, currently there are different Web portals that provide similar functionalities as the ones offered by the PC-based interfaces, allowing in addition access to different kinds of resources, documentation, and development tools. According to [Beasley et al, 2001], these portals present several advantages over the PC-based simulator environments provided by most of the toolkits described in the previous section. For instance, Web-based environments are comprehensive and relatively easy to use, and allow starting the development of the service inexpensively without setting up complicated platforms or complex simulated VoiceXML networks. Another advantage of Web portals is that they allow developers to deploy the VoiceXML code using a Web server and accessing the service through a voice service provider (VSP). This way, the service can be tested without requiring any investment on proprietary hardware, and using the same interface as the real users, allowing the designer to detect network latency and other interface issues that are not possible to detect when using simulated environments. Finally, most of the Web-based environments are supported by leading companies that allow these portals to offer sophisticated and state-of-the-art modules such as speech synthesizers, recognizers, speaker or language identification modules, etc., which can be used to improve the quality of the service without requiring companies to buy them. The main disadvantages are that these portals restrict the control over the platform and the creation of dynamic VoiceXML applications, reduce the possibility of obtaining knowledge about how the VoiceXML interpreter and gateway work, and make difficult the transition and sharing of information between the Web-based applications and PC-based applications.

In general, designers can use the Web-based platform after applying for a developer account, which is most of the times free or inexpensive, and can be done through the Web site. Additional services or support for more complex applications will require a professional

or enterprise account, which can be obtained after paying an additional fee that, in any case, is low in comparison to the price of the licenses for using PC-based platforms. In the first versions of these systems, after the service is created and deployed, it could be accessed through conventional Public Switched Telephone Network (PSTN) lines using a toll-free number and providing an ID of the service. Unfortunately, this kind of access limited the possibility of providing advanced multimedia services and turned out to be frustrating for developers living in other countries (since international calls are not free). In order to reduce these problems, most Web sites offer now the possibility of accessing the service using a SIP (Session Initiation Protocol) phone number and a VoIP (Voice over IP) phone. Currently there are many hard (e.g. 3Com, Avaya, Siemens, etc¹¹.) and soft phones (e.g. sipXphone, SightSpeed, SJPhone, etc¹².), which can be used to make calls using an Internet connection. However, the most common and cheapest solution is to use soft phones since they are simple computer programs that do not require dedicated hardware and are easily downloadable from many internet sites. An advantage of using these soft phones is that they add new features to standard telephony like video and wideband audio, providing new services and allowing interactions with the final users using other modalities. In this thesis, when it was required, we used X-Lite¹³ for testing the runtime platform. Appendix B includes detailed information about the most important Web portals (e.g., Bevocal Café, TellMe Studio, Voxeo Evolution, VoiceGenie) and their main features. For further information, please refer to the corresponding Web page or read [López-Cozar and Araki, 2005] [Beasley et al, 2001].

2.1.1.2 Grammar development

One of the most important aspects of a well-designed and user-friendly speech-enabled service is the capability of the dialogue manager of being able to understand and correctly parse the unlimited range of users' utterances for a given system prompt or request. In order to do this, the dialogue manager uses static or dynamic speech grammars which are responsible of modelling the set of possible and valid recognized sentences, as well as the semantic interpretation (i.e. the values to be returned by the grammars) of the user responses. It is important to highlight that an incomplete or bad-designed speech grammar will make the system fails in the process of understanding the user and completing the dialogue goal.

Given the complexity of developing speech grammars, the most of the above-mentioned commercial platforms include assistants for debugging and testing grammars, as well as built-in grammars for common situations (e.g. for requesting card numbers, phone numbers, dates, currencies, airports, cities, social security numbers, etc.). Below we provide some examples of the more sophisticated wizards or specialized development tools for creating speech grammars.

Grammar Studio¹⁴: This tool provides a complete GUI for creating and debugging complex speech grammars through a clear and easy to use workspace layout, even for designers with little knowledge on speech grammars and format languages. The toolkit uses grammar icons and connecting lines in order to create a visual representation of the grammar. The graphical representation is automatically parsed in order to create the final definition of the grammar file in SRGS format. Besides, the toolkit includes the possibility of importing or exporting partial or complete grammar files. Finally, the designer can also write directly the

¹¹ <http://www.voip-info.org/wiki-VOIP+Phones>

¹² http://en.wikipedia.org/wiki/List_of_SIP_software

¹³ <http://www.counterpath.com/>

¹⁴ http://www.voicewebsolutions.net/grammar_speech_tools.html

grammar in SRGS format and the system will automatically converted it into its graphical representation. The platform accelerates the process of writing the grammars using auto-complete capabilities and auto-tagging in order to avoid typing errors. The main window can be split into two main sections, one for the graphical definition of the grammar, and another one for the written representation of the grammar defined in the graphical view.

Microsoft Speech Grammar Editor (Visual Studio)¹⁵: This tool is included in the freeware Microsoft Speech Application SDK, which can be downloaded from the Microsoft Web site. The tool uses a basic graphical interface (see Figure 2.2) with drag-and-drop capabilities, allowing the creation of lists of words, references to other rules or grammar libraries, groups of words, creation of semantic tags, wildcard rules, etc. The toolkit also features a grammar checker and a sentence tester for debugging the grammar.

Semantic Grammar Studio (SGStudio): Reported by [Wang and Acero, 2006], they present an innovative Microsoft tool that allows the rapid creation of grammars through a supervised algorithm with examples provided by the designer. Besides, the tool allows the definition of rules that can be applied for different situations accelerating, in this way, the design and taking advantage of previous knowledge (i.e. previously generated grammars).

Visual JSGF¹⁶: This tool is included in the Matrubhasha platform that provides a text-to-speech and speech recognition system for Indian languages. The main features included in this tool are a point-and-click interface, automatic grammar and sentence generation, reusable templates, and an assistant for the creation of pronunciation dictionaries for the speech recognition and TTS engines. The toolkit allows the designer to define and save groups of words, i.e. optional words in a grammar rule, and import them later as templates in other grammars. Finally, the designer uses the graphical interface for creating the grammar rules through a hybrid combination of words and group of words, and setting the respective connection between them.

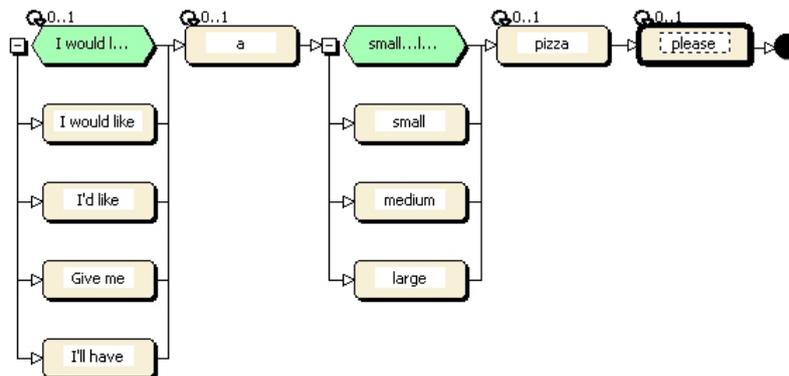


Figure 2.2. Example of creation of a Speech Grammar using the Microsoft Speech Application SDK

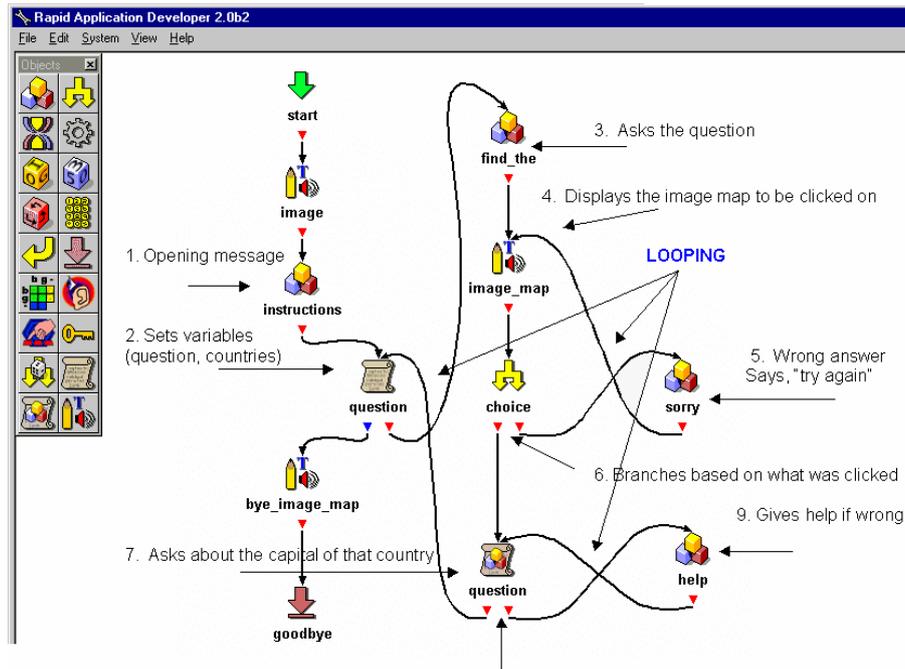
¹⁵ <http://msdn.microsoft.com/>

¹⁶ <http://www.ncb.emet.in/matrubhasha/visualjsgf.shtml>

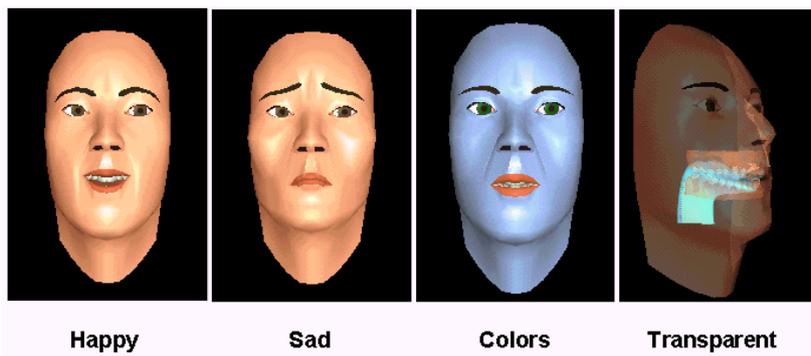
2.1.2 Academic and Research Platforms

In contrast to most of the commercial platforms, academic and research platforms allow designers to create more complex dialogue interactions providing features not included by any standard description language. They also allow the creation of more complex multimodal dialogues, some are freely available as open source, and their functionalities can be extended using proprietary or third party modules.

The following are noteworthy examples of tools developed in academic environments:



(a)



(b)

Figure 2.3.CSLU’s RAD Toolkit: a) Example of the main canvas, available objects, and dialogue states definition. b) Example of possibilities using the included animated agent: Baldi. (Source: CSLU Toolkit home page)

CSLU's RAD Toolkit¹⁷: Created at the Center of Spoken Language Understanding (CSLU) at Oregon Graduate Institute [McTear, 1999][Cole, 1999], it allows the development of multimodal system initiative dialogues (combining voice, DTMF, interactive images, graphs, text, and animated agents), using a representation based on state-transition networks [McTear, 1998] that describe the different functions and actions in the dialogue. The states and transitions of the dialogue flow are created using a toolbar with objects from where they can be dragged and dropped into the canvas and connected with arrows to other objects. The toolkit reduces the size of the information displayed in the canvas using sub-dialogues, which group repetitive or common actions.

In addition, the toolkit includes an embedded speech recognition system that can be configured in different ways; for instance, it is possible to define task-dependent or independent models for recognizing alphanumeric or digit strings, to use continuous or isolated models, or to use a recognizer based on Hidden Markov Models (HMMs) or on Artificial Neural Networks (ANNs). At the same time, the toolkit incorporates a free Text-To-Speech (TTS) engine based on Festival¹⁸, and a module called Baldi Sync that it is used to align speech files or synthesised sentences with the face/lips movements of the animated agent. Another tool, called CU Animate [Ma et al, 2002], allows selecting the animated agent to use in the application, and configure facial expressions such as blink frequency, head movements, colours, emotions, eyebrows, etc. of the animated agent. The tool controls and renders the animated agents in real-time. Finally, the toolkit includes an interface to run Tcl/Tk scripts that can be used to perform better dialogue analysis or grammar development, as well as to allow a higher interaction of the platform with backend databases and internet resources, among others. Figure 2.3 provides an example of the toolkit environment and capabilities of the animated agent.

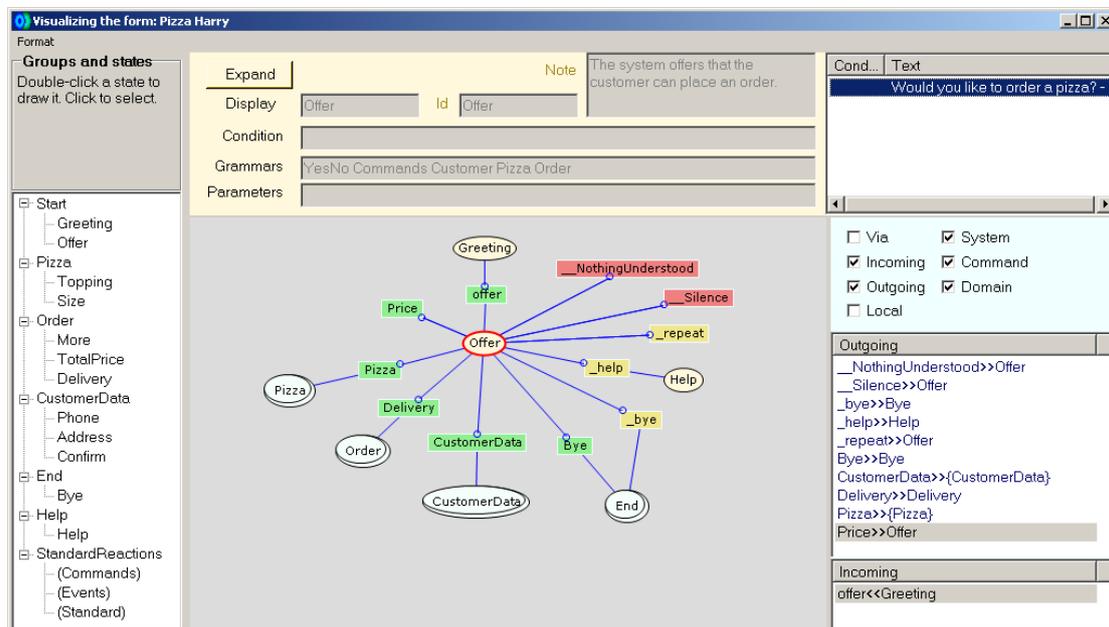


Figure 2.4. Detailed graphical presentations of a dialogue model using DialogDesigner (Source: DialogDesigner Web page)

¹⁷ <http://cslu.cse.ogi.edu/toolkit/>

¹⁸ <http://www.cstr.ed.ac.uk/projects/festival/>

DialogDesigner¹⁹: [Dybkjær and Dybkjær, 2005] present a tool for designing and evaluating dialogue models. The platform allows the definition of the dialogue flow through a single window where the designer can specify states, groups of states, transitions between states, conditions, prompts and grammars (see Figure 2.4). The platform also includes functionalities for running a Wizard of Oz simulation through a text-based interface, the possibility of providing detailed information about the dialogue flow using the graphical view options, and the possibility of converting the flow into a set of linked HTML pages that the designer can also use to debug the service.

DialogStudio: Described in detail in [Jung et al, 2008], it is a recent platform for developing data-driven spoken dialogue systems that integrates several tools that cover the different steps of a dialogue design, i.e. from preparing data to testing the service. One of the main objectives of the platform is to provide a complete set of functionalities for preparing the input files to be used by the speech recognizer, language understanding, and dialogue manager modules. The platform also provides an annotation environment for tagging semantic and knowledge information, as well as dialogue examples; in this case, the platform uses a meta-model language that allows the quick definition and adaptation of semantic and dialogue structures to domain specific knowledge. Other included accelerations are the possibility of creating and adding new words into the ASR's dictionary and language models, and the generation of new sentences from other domains with similar semantic structure. Finally, the platform provides an average time reduction of 30% when compared with other dialogue platforms on the creation and annotation of three different domains, i.e. an electronic program guide, an immigration simulation, and weather information domains.

EVITA-RAD: Reported by [Chen, 2004], it is a Web interface tool for building VoiceXML applications using a set of predefined system modules. These modules correspond to dynamic Web pages consisting of forms, checkboxes, or drop down menus that the designers use to define different actions such as dialogue states (including mixed-initiative capabilities), database queries and updates, variable confirmations, and call transfers. Besides, the tool also includes other assistants for developing grammars, vocabularies, prompts, and other VoiceXML actions.

GULAN: Reported by [Gustafson et al, 1998], it is a research platform used to build multimodal dialogue services using speech and interactive maps. The platform has been successfully used for creating a yellow pages system to make searches for different services in Stockholm. The dialogue flow is defined using a tree representation whose nodes model the structure, focus, and actions that are executed inside each dialogue state.

SpeechBuilder: Described in [Glass and Weinstein, 2001], this platform has been developed at the Spoken Language Systems Group from the Massachusetts Institute of Technology (MIT). In this application, the design is made through a Web interface that allows, using an action definition language based on examples, the specification of the relevant semantic concepts and actions that are allowed in the application. Then, the generated dialogue model is executed using available modules from the Galaxy architecture [Polifroni et al, 2000]. The communication and parameter passing between the Web site and the runtime platform is achieved using the HTTP protocol and a predefined CGI script (similar to the one we have implemented in our runtime platform).

¹⁹ <http://spokendialogue.dk/DialogDesigner/DialogDesigner.html>

Trindikit²⁰: Initially described in [Larsson and Traum, 2000], it was developed and improved during the projects TRINDI²¹, SIRIDUS²², and TALK²³. This toolkit allows the creation and evaluation of the dialogue manager and information states. Designers can create a complete dialogue application through the definition of the information states (i.e. dialogue history), dialogue actions, update rules, dialogue grammars, inference engines, and planners. The toolkit proposes a general system architecture and allows experimenting with different algorithms, rules, and implementations of information states for the dialogue manager; it also includes a GUI for inspecting information states, speech recognition and synthesis modules, a debugger module, and ready-made module interfaces to databases, input/output devices, interpretation, generation, etc.

VoiceComposer: described in detail in [Li and Lin, 2006], it is a development tool that allows the visual programming of the dialogue flow. The platform architecture consists of five main modules: a dialogue flow editor, a dialogue component builder, a database integrator, a script generator, and a service simulator. These modules let the designer build the dialogue flow using built-in dialogue components (i.e. predefined templates for different kind of VoiceXML functionalities and actions), simulate the application, and the specification and evaluation of SQL commands to be used to communicate with the backend database. In addition, the dialogue component builder allows the designer to edit or create new dialogue components allowing the possibility of creating global variables and events, to execute a sequence of form or menu elements, or to create returning dialogues. The paper reports that the toolkit has been used in a research project for converting part of a Web site for life education into a voice-enabled Web service.

2.1.3 Research Platforms that Provide an Assisted Dialogue Design

As it has been described, surprisingly most of the above-mentioned commercial and academic platforms do not include any kind of acceleration based on the contents and structure of the backend database, which can provide important information to accelerate the design when developing the dialogue service. However, in the literature we can find some examples of how this information is used to accelerate the design of the dialogue flow and other aspects of the design. This section describes the most relevant systems and strategies included in them. It is important to mention that most of these strategies are based on using predefined templates for different kind of dialogues and situations, or on using the contents of the database.

In [Denecke, 2002], a complete three-layer architecture for rapid prototyping of dialogue applications is presented. In the first layer, language and domain-independent algorithms are provided to describe the dialogue objectives, discourse history, and the semantic representation of the speech recognizer output. In the second layer, the interaction mechanism between user and system is described (e.g., variables used by the recognizer, database access variables and methods, dialogue states, etc.) Finally, the third layer contains the dialogue controller that uses the information from the other two layers and interacts with the final user. This system is similar to our proposal in some aspects, as the handling of concepts to facilitate multilingual interaction, the use of special variables related to the

²⁰ <http://www.ling.gu.se/projekt/trindi/trindikit/>

²¹ <http://www.ling.gu.se/projekt/trindi/>

²² <http://www.ling.gu.se/projekt/siridus/>

²³ <http://www.talk-project.org/>

system and dialogue status, and the use of automatic preconfigured templates for each dialogue state. Nevertheless, as the author admits, the built-in templates fail when not all the states of the dialogue can be covered. In our system, we have tried to avoid this problem by using more flexible and general templates, although less automatic.

Another platform that follows a very similar approach to the platform developed in this thesis is the Agenda system (now called RavenClaw) from Carnegie Mellon University (CMU) described in [Rudnicky and Xu, 1999] and [Bohus and Rudnicky, 2003]. For instance, both platforms are similar in that the designer can create the service using a hierarchical representation of the task and its subcomponents, facilitating maintenance and scalability, and that each state is described using a set of forms with information regarding its restrictions and optional slots.

In [Polifroni et al, 2003] a rapid development environment for creating spoken dialogue applications using online content is described. The development process is started extracting knowledge from various Web applications and composing a dynamic database from it. Then, the dialogue flow is determined at runtime depending on the contents of such database. In this platform, they propose a methodology for creating automatic clusters that group and organize numeric data into symbolic data. For instance, the symbolic concept of cheap/medium/expensive in the domain of a hotel reservation is automatically created according to the information in the database (i.e. hotel rates vary depending on the city). Then, at runtime, the system summarizes the partial retrieved information and creates the prompts to present to the users, determining also the order in which they appear, based on the most useful set of attributes to narrow down the current data subset. Although this algorithm is very interesting, it is more limited than ours since we also use the database structure, not only its contents, to extract knowledge for the design process. Because of this, the design is more domain-independent, as it is more feasible to find data structures that are similar between several services and, therefore, can be applied in several applications. Besides, there are databases whose contents cannot be easily used for research purposes for security reasons as in banking or medical databases where confidential information exists. Another important difference is that the speech dialogue applications generated by our platform will be implemented in VoiceXML, which allows the generated dialogues to be executed with any VoiceXML interpreter. However, the idea of using the database content when it is available was also explored in this thesis.

In [Pargellis et al, 2004] a complete platform to build voice-enabled applications is described. The dialogue structure can be modified using a set of templates adapted to the final user of the system, as well as several resources and service features. As in the proposal of this thesis, the platform automates the generation of the final script in VoiceXML, the grammars and prompts, and the application flow; nevertheless, their proposal differs from ours in that the automation efforts, in a similar way as in [Polifroni et al, 2003], are more focused on the dynamic contents of the database than on its structure, so it could be more domain dependent.

[Tsai, 2006] presents a multimodal dialogue system that allows users to access the final service using a voice or visual interface. In this way, the users can use traditional telephony devices (e.g. phones or cell phones) and VoIP-based phones, and at the same time use any Web browser to interact with the application. The paper describes a proprietary mark-up language that is used to allow the interaction between the VoIP and the Web platform, as well as detailed information about the modules that generate and interpret the dialogue scripts, and the runtime platform used to access the service through the different devices and modalities. Finally, the paper describes some demo services created using the platform, as well as

evaluation results of the speech recognizer considering the user level (i.e. novice and expert) and the number of attempts required to be successfully recognized.

[Polifroni and Walker, 2006] describes an interesting technique that allows the dynamic creation of system prompts based on partial retrieved database results and the automatic selection of the most relevant information to request to the user in order to restrict future retrieved results. Besides, they propose to use data mining techniques in order to dynamically create system messages with summarized information. The technique is a two-steps procedure: the first step consists of calculating the entropy, or information gain, of the data in focus at each turn in the dialogue (i.e. retrieved database results using partial information provided by the user according to the current dialogue history). The second step consists on using a decision tree induction for inferring association rules among the database attributes using the entropy calculated in the previous step. From the inferred association rules, the system selects an appropriate set of them in order to create the intentional summary messages and data-specific queries for the current dialogue state. A preliminary evaluation showed that users prefer this system when they are unfamiliar with the knowledge contained in the database, but if they are familiar with the data then they prefer direct dialogues (i.e. traditional system's initiative dialogue systems).

In [Chung, 2004], the database contents is used together with a simulation system in order to generate thousands of unique dialogues that can be used to train the speech recognizer and understanding module, and to diagnose the system behaviour against problematic user's interactions or unimaginable user's answers, etc. In [Wang and Acero, 2006] the database contents are used to accelerate the creation of grammars for the speech recognition and spoken language understanding modules. In this case, the system uses the database to generate a large number of artificial sentences that are integrated into semantic frames in order to create customized grammars for different scenarios.

[Feng et al, 2003] propose a very different approach. In this case, they do not extract information from a backend database but they apply data mining techniques to the contents of corporate websites for automatically creating spoken and text-based dialogue applications for custom care. The process is carried out through a Website analyzer that exploits the contents and structure of the site in order to generate structured and semi-structured task data. Then, the generated data is classified according to some predefined information units (e.g. menu, question-answer, topic-explanation, etc.). With this information, the dialogue manager, at runtime system, will identify the focus or expectation of the user's question and will provide a concise answer. Although the dialogue flow is not defined using any GUI application, it is interesting to observe that important knowledge for the different modules of a dialogue system can also be extracted from well-designed contents.

It is important to highlight the growing interest in using XML-based languages in order to exchange input/output data between different modules and to allow the specification of multimodal dialogue systems [Flippo et al, 2003][Katsurada et al, 2002] and [Araki and Tachibana, 2006]. Examples of this kind of languages are VoiceXML, SALT, XISL [Katsurada et al, 2003], or X+V, etc., which offer portability and flexibility, they also allow the quick definition of the dialogue flow, interaction between modalities, management of system and user errors for complete voice-enabled services [Komatani et al, 2003][Bennett et al, 2002]. Considering also that the selected description language can contribute to make independent the generated service from the execution platform, and the big number of tools for parsing and checking tag based languages, we decided to create our own language (called GDIALOGXML) to allow the internal communication between platform assistants, and to use the standard VoiceXML and XHTML as output scripts for the runtime system.

In relation to multimodal systems, we should especially mention the work in [Johnston et al, 2002], where a multimodal architecture for a dialogue system based on finite states is described. This architecture allows synchronous multimodal input/output data allowing users to use speech and/or gestures/images with a pen on a PDA. The authors make emphasis on the methodology used to guarantee the multimodal interaction, the features provided for each modality, and the architecture that supports the multimodality. In a recent improvement over this previous work, [Johnston et al, 2007] extend the proposed architecture to a new domain, i.e. multimodal access to contents in the home environment, including new interesting modalities such as handwriting, remote control, and dynamic combination of these modalities and speech. Even though our present system does not provide this kind of interaction right now, future work will be oriented towards the creation of a similar mechanism using a multimodal specification language like X+V, SALT [Wang, 2002], XISL [Katsurada et al, 2003], or MILM [Araki and Tachibana, 2006].

Another interesting application is presented in [López-Cozar et al, 2005]. In this case, they describe a multimodal dialogue system, based on the X+V language specification, that helps professors and students in some common academic activities such as obtaining information about available books for a specific subject at the library or turning-on/off lights at professor's office. The system also uses user and ubiquitous information for adapting its behaviour and capabilities. Besides, the application handles explicit confirmation and mixed-initiative interaction.

[Katsurada et al, 2002] describe a modality-independent system architecture. The architecture is divided into three main modules: the document server module, the dialogue manager, and the front-end module. The first one handles the scripts of the service, the second one controls the dialogue flow, and the third one the user's input and output. An interesting contribution of this paper is that the applications are written in a XML-based modality-independent language called XISL. This language provides all the required information for controlling different user inputs and system actions, output messages to the user, arithmetic operations, flow control, etc. Finally, the language also includes tags for allowing a basic synchronization of the different modalities.

We also need to mention the SmartKom²⁴ project. Described in detail in [Wahlster (Ed.), 2006], the main result of this project was the creation of a robust multimodal dialogue system that supported symmetric multimodality (i.e. the system allows all input modes, e.g. gesture, speech, and facial expressions, to be also available for output) and mixed-initiative dialogues. The platform includes an embodied anthropomorphic conversational agent called Smartakus, which features coordinated speech, facial expression, and emotional gestures allowing face-to-face dialogue interactions between the system and final users. One of the main research problems tackled in this project was the integration and mutual disambiguation of all the symmetrical modes, including also the resolution of back-channelling, cross-modal references, multimodal anaphora resolution, turn-taking, meta-communicative interaction, and other discourse phenomena such as ellipsis and deixis resolution/generation. In addition, the dialogue manager exploits predefined models of the user, task, context, domain, and modalities to adapt the service and provide better interactions. The SmartKom architecture also supports multiple parallel recognizers such as emotional prosody, boundary prosody and speech recognition, which can help, for instance, to detect user's emotions or to check if the information provided by the system fulfils the expectations of the user or not.

²⁴ <http://www.smartkom.org/>

Among the main contributions of the SmartKom project was the complete specification of a XML-based language called M3L (Multimodal Markup Language) designed for the exchange and representation of the multimodal content. The big advantage of this specification is that it covers all data interfaces, avoiding using several different languages formats for each component of the platform. For instance, the word hypothesis graph, gesture hypothesis graph, media fusion results, hypothesis about facial expressions, information about segmentation, synchronization, confidences, etc., were all encoded using M3L. Besides, the specification was decomposed into about 40 different schemas in order to provide a thematic organization of the language that makes it manageable and to allow verifications during information exchange. Finally, this language also allows the definition of the appearance and contents of the information that has to be provided to the final user according to different parameters such as user preferences, current scenario, language, and output device. Then a set of XSLT stylesheets transform the M3L files into the format required specifically by each output device. In some ways, this language can be compared to the GDialogXML (see section 3.1, page 58) language used by our platform, although M3L goes beyond in that it is extended to the runtime system, and not only to the design of the service.

Another contribution of this project was the proposal of a distributed run-time architecture called MULTIPLATFORM (Multiple Language Target Integration Platform for Modules). This architecture differs from the widely used Galaxy Communicator Architecture [Polifroni et al, 2000] in that there is not a central hub in order to avoid any information bottleneck and to allow more complex interactions between servers. Although this new architecture supports more complicated multimodal dialogue architectures, we decided to use a similar architecture as Galaxy because it is simple, the complexity of our system is lower, and because the messages between our modules were less complicated.

Finally, regarding tools that can be used to debug and evaluate the application off-line, we should mention the SUEDE platform [Klemmer et al, 2000]. This application offers a graphical interface to design, test, and analyze a dialogue system using the Wizard of Oz (WOZ) technique. The objective is to provide the designer a controlled environment for running an electronic WOZ. SUEDE allows designers to test system prompts and user responses, simulate speech recognition errors, timeouts, barge-in prompts, analyze logs, review responses across participants, etc. This way, the toolkit allows an easy evaluation and improvement of a dialogue system without requiring a complete runtime system.

[Dybkjær and Dybkjær, 2006] also describe a debugging tool for the DialogDesigner environment (see section 2.1.2, page 16). In this case, the debugger allows testing the dialogue model by selecting transitions and listening to the activated prompts. The simulation is logged which allows its posterior analysis. Most commercial applications and Web portals provide similar debugging tools. In our case, we have tried to minimize some of these design problems using automatic and configurable templates for the treatment of common system errors.

[Ito et al, 2006] present an interesting user simulator tool based on VoiceXML files that can be used to evaluate the behaviour of the designed service. The application uses synthesized voice in order to evaluate the dialogue flow without requiring real users. Although this system is able to predict human-machine interactions, the program has to be improved to accept out-of-task behaviours or out-of-vocabulary utterances. In our current system we have not implemented this kind of assistant, but because we generate VoiceXML files we could take advantage of this program or similar ones provided by third parties.

2.1.4 Weaknesses of Commercial and Academic Platforms

In spite of all the features included in most commercial platforms, a large drawback they present is that the runtime platform depends on the underlying technology (speech recognizer, text to speech systems, dialogue managers, etc.) therefore the behaviour of the service may vary across different platforms. In addition, it is difficult to integrate proprietary modules, most of the assistants do not take advantage of the contents of the database or data model structure, and they do not offer any proposal for completing or defining the service or common modalities issues. Finally, they may present difficulties in integrating new modalities, creating the service in multiple languages, adapting the service according to predefined user profiles, or for obtaining the same functionalities using the same platform but on different operative systems.

Regarding academic and research platforms, although most of them are easy to use, they may show serious limitations such as a low portability level as they are tied to specific running platforms being difficult to integrate them with other systems and/or architectures. Besides, they require the designer to know several programming languages and non-standard formats reducing this way their usability. In addition, they may present limitations when trying to implement dialogue strategies that take into account the user level and different modalities, or when simultaneously building the application for several languages.

Despite all the advantages and accelerations included in the commercial and academic platforms, most of them do not provide any kind of acceleration based on using data mining techniques applied to the contents of the task database and using information from the data model structure. In this thesis, we have solved this limitation incorporating successfully heuristic information into the different assistants and allowing these assistants to collaborate between each other in several ways, as they collect the information already provided in the first stages of the design to improve and accelerate the design in the last stages. This way, the platform assistants classify which fields of the database could be relevant for the design, generate different kinds of automatic proposals according to the design step, reduce the information displayed to the designer, and accelerate different procedures required to define the service.

In addition, during the design of our platform we also provided solutions to other limitations of the commercial and academic platforms mentioned above. For instance, the platform was structured into three layers allowing a separation between the general and high-level definition of the dialogue flow and the specific details imposed by each modality, language, and user profile. Besides, we also made a big effort for making independent the platform from the operating system and the runtime platform by using several standard languages such as VoiceXML, JSGF, SSML, xHTML, etc. In addition, we have incorporated several tools and assistants that provide access to new kind of user interactions such as animated agents, automatic machine translation, and language identification.

2.2 Language Modelling

Nowadays, language modelling is a general term that refers to the attempt of capturing the properties of a given language as accurate as possible in order to improve the performance of a wide range of natural language processing applications such as speech recognition, information retrieval, machine translation, language identification, Part-Of-Speech tagging, text-to-speech, parsing, spelling correction, document classification,

handwritten recognition, etc. Depending on the application a different definition and purpose is given to the language models. For instance, in speech recognition a language model (LM) is mainly used to predict the next word in a speech sequence or to define the set of allowed sentences that a user can use when communicating with the system in order to be successfully recognized. In machine translation, LMs are used for choosing among different candidate hypothesis or to generate a score that measures the quality of a translated sentence.

In general, there are two different kind of grammars widely used in most spoken dialogue applications: a) stochastic grammars, and b) Context Free Grammars (CFG). The former is appropriate for recognizing free-style speech and in applications where it is important to avoid writing complex grammar rules. Besides, it is the preferred one for research purposes. On the other hand, the latter is useful for applications with a restricted phraseology such in most automatic dialogue services. Besides, this kind of grammars provides good results when there is not enough training data to obtain reliable statistical-based models.

In this thesis, three different applications and types of language models were used. The first one is used in one of the assistants included in the development platform, allowing both the creation of CFG grammars in JSGF format and word-based language models to be used by the runtime platform to recognize user utterances and return semantic information to the dialogue manager. The second one was used for language identification, allowing the correct identification of the language uttered by a user. In this case, the proposed statistical phone-based language model provides local and long-span information that is integrated into the front-end feature vector used for a Gaussian Mixture Model classifier. The third one is a statistical word-based language model that has been adapted and used to improve the quality of a machine translation system that translates previously defined system prompts into a sign language representation in order to provide speech-based dialogue services to deaf people.

In this section, stochastic and finite-state grammar approaches will be studied in detail, explaining, in each case, its advantages and disadvantages, and the most common strategies to improve their performance. However, taking into account that the most important contributions of this thesis have been obtained using statistical language models for the LID and machine translation systems, in this section we will emphasize this kind of grammars, describing in detail the main solutions to the problem of insufficient training data and the management of long-span information.

2.2.1 Statistical Language Models

As mentioned above, statistical-based language models (SLM) are the most widely used kind of grammars in most natural language applications. In simple terms, a SLM is a probability distribution $P(s)$ over all possible strings S (or documents, spoken utterances, phone sequences, or any other linguistic unit, etc.) that tries to reflect how frequently a string s occurs as a sentence. In general, it is common to decompose the probability distribution into a product of conditional probabilities using Eq. 2.1. Here, w_i is the i^{th} word in the sentence, and h_i is the history.

$$P(S) = P(w_1 \dots w_n) = \prod_{i=1}^n P(w_i | h_i)$$

Eq. 2.1

Several SLM techniques have been proposed [Rosenfeld, 2000], but the most widely used are the n-gram models. In this kind of models, the next word is predicted using only the n-1 more recent words in the history (i.e. Markov assumption) using Eq. 2.2.

$$P(w_i | h_i) \approx P(w_i | w_{i-n+1} \dots w_{i-1})$$

Eq. 2.2

N-gram conditional probabilities are trained using Maximum Likelihood Estimations (MLE), i.e. the ratio between the observed frequency of occurrence for a given n-gram divided by the observed frequency of occurrence of the n-1 gram in the training corpus using Eq. 2.3.

$$P(w_i | w_{i-n+1} \dots w_{i-1}) = \frac{\text{Count}(w_{i-n+1}, \dots, w_{i-1}, w_i)}{\text{Count}(w_{i-n+1}, \dots, w_{i-1})}$$

Eq. 2.3

The great advantage of n-gram based LMs lays in that they are easy to train, provide good results very quickly, they are robust (when there is enough text data for training), they are widely accepted, and there is a big number of software and algorithms to train and test them [Rosenfeld, 2000][Goodman, 2001][Bellegarda, 2004]. However, they present three main disadvantages that reduce its predicting and adaptability capabilities:

1. In theory, it is possible, and desirable, to use a long n-gram order model since they will use a larger context that increases the prediction power. However, in practice, the value of the order is a trade off between the generalization and stability of the estimations of the model. In addition, the size of the training corpus is an important factor to choose the order of the model. The biggest the corpus the highest the order. Unfortunately, even with a big corpus most of the n-grams will occur just once or twice or will not occur at all. In these cases, the ML estimation will result in that those n-grams will obtain a high, but poorly estimated, probability or, even worse, a zero probability when they do not occur at all. In order to avoid these problems, several smoothing and interpolation techniques have been proposed in the literature. Some of these solutions will be presented in section 2.2.1.1.
2. As stated above, the n-gram based models assume some kind of independence among different portions of the same document predicting the next word based only on a reduced context given by the n-1 more recent words in the history. This simplification is useful to keep the model trainable; however, it does not provide a long span neither semantic information. The direct consequence is that the model does not take full advantage of the dynamic characteristics of the domain or from the dialogue history. Moreover, since the n-grams just model the local information of the sentences, it is possible that the model assigns a high probability to sentences that do not have a correct syntax. In order to solve these problems, it is common to interpolate word n-gram models with dynamic models (e.g. cache, trigger pairs) or with topic specific models. Besides, in order to provide syntactic and semantic information the preferred solutions is to use other kind of models as POS (Part-Of-Speech), class based n-grams, hybrid grammars such probabilistic Context Free Grammars (see section 2.2.2, page 35), or to rescore n-best lists results with semantic parsers. Sections 2.2.1.2 and 2.2.1.3 (pages 30 and 31) will provide more information about solutions for this problem.

3. The remarked dependency of n-gram based models to the domain of the training data. For instance, [Rosenfeld, 2000] and [Rosenfeld, 1996] report that a model trained with 2 million words from the same domain as the testing data is better, in terms of perplexity and Word Error Rate (WER), than another model trained with 140 million words from a different domain. Although this problem is well known, most of the times it is not possible to have such kind of large in-domain training data, especially for new domains or minority languages. A proposed solution is to generate automatically new sentences using templates learnt or defined from the in-domain data that capture the phraseology used in the available training data. Then the new sentences are filtered and included in the corpus in order to re-estimate the model. Another possibility is to adapt the poorly estimated model with more reliable models trained with texts from other domains or retrieving online sentences closer to the in-domain data. In sections 2.2.1.4 and 2.2.1.5 (pages 33 and 34), we will provide more details about the proposed solutions for this problem.

Finally, in order to evaluate the performance of n-gram based models several metrics have been proposed [Chen et al, 1998]. However, the preferred one is the perplexity. In the context of language modelling, the perplexity gives a rough idea of the average number of different possible words that can follow a given context and how much information is provided by the grammar. This metric is interesting since it can be used to compare different language models, to measure the complexity of a speech recognition task, or to estimate how well a language model is to match a given corpus. Perplexity is often calculated using Eq. 2.4. Here, $P(x_i)$ is the probability assigned by the language model to the sequence of words in the test set, and N is the number of words in the test set.

$$2^{-\frac{1}{N} \sum_{i=1}^N \log_2 P(x_i)}$$

Eq. 2.4

Although perplexity is easy to calculate and provides a simple mechanism for evaluating different language models, it is often combined with other metrics in order to measure its effect on the application where the language models are applied. Typically, the quality of a language model is measured by its effect on the error rate (e.g. word error rate, language identification rate, translation rate, etc.). Unfortunately, error rates are difficult to calculate, are commonly non-linear, and do not correlate completely with the results provided by the perplexity. Several attempts to find correlations between perplexity and error rates [Klakow and Peters, 2002], or to find better correlated and easier to optimize metrics have met with limited success [Chen et al, 1998]. However, according to [Rosenfeld, 2000] there is a rule of thumb to estimate the effect of the improvements on perplexity over the error rate. For instance, a reduction of 5% in perplexity is not significant in practice. A 10%-20% improvement is usually, but not always, translated into some improvement in performance. More than 30% is quite significant but it is too difficult to obtain.

In the following sub-sections, we will explain in more detail some of the proposed solutions to the problems that affect n-gram models. For more information please consult [Jurafsky and Martin, 2008], [Manning and Schütze, 1999], and [Rosenfeld, 2000].

2.2.1.1 Smoothing techniques

As mentioned above, one of the weaknesses of the n-gram based models is that they underestimate the probability of words that do not occur in the training corpus, assigning them a zero or low probability. In order to avoid this problem, it is possible to re-estimate these zero and low-probability n-grams using smoothing techniques. The smoothing is based on saving a probability mass, discounted from more frequent or better-estimated events, to be distributed among all low frequency events. The most frequent types of smoothing techniques are Witten-Bell, Good-Turing, and Kneser-Ney.

Briefly, Witten-Bell, [Witten and Bell, 1991], introduces the concept of using information regarding the counts of n-grams seen just once for estimating the counts of n-grams that do not appear in the training; this is a relevant contribution since several smoothing methods rely on using the same concept. In this method, see Eq. 2.5, the existing counts are modified by a factor that depends on the number of n-gram tokens, $C(h_i)$, and the different n-gram types or contexts, $T(h_i)$, seen in the training text for the given n-1-gram context. The idea is that words that tend to occur in a smaller number of contexts will contribute with a lower probability mass than words that appear in more contexts. On the other hand, the probability of unseen events is calculated distributing the discounted probability mass considering the number of n-gram tokens, $C(h_i)$, the different n-gram types, $T(h_i)$, and the number of n-gram tokens that do not occur at all in the training text, $Z(h_i)$.

$$p^*(w_i | h_i) = \begin{cases} \frac{T(h_i)}{Z(h_i)(c(h_i) + T(h_i))}, & \text{if } c(w_i | h_i) = 0 \\ \frac{c(h_i w_i)}{c(h_i) + T(h_i)}, & \text{if } c(w_i | h_i) > 0 \end{cases}$$

Eq. 2.5

In Good-Turing, [Good, 1953], the smoothed counts, c^* in Eq. 2.6, are estimated using the concept of frequency of frequency (i.e. the sum of different n-grams, with the same order, that occur c times). Here, the smoothed count is estimated using the original count, c , plus one multiplied by the division between the number of n-grams that occurred $c+1$ times, N_{c+1} , by the number of n-grams that occur c times, N_c . Therefore, the smoothed counts for n-grams with zero counts are estimated by dividing the number of n-grams that occurred once by the number of n-grams that never occurred. [Katz, 1987] was the first one on applying this equation to the smoothing of n-gram based grammars, introducing at the same time the concept of applying the smoothing only to n-grams whose frequency of frequency is lower than a given threshold, generally from one to seven, since for more frequent n-grams the probability estimation can be considered as reliable.

$$c^* = (c + 1) \frac{N_{c+1}}{N_c}$$

Eq. 2.6

Finally, in Kneser-Ney smoothing, [Kneser and Ney, 1995], the probabilities for all non-zero n-grams are discounted with a constant amount δ (see Eq. 2.7). For all zero n-grams, the probability is calculated taking into account the number of different contexts in which a word occurs. A modified version of this technique is also proposed in [Chen and Goodman, 1998] where different values for the discounting parameter are used instead of a single constant parameter. [Goodman, 2001] presents detailed information for each technique besides an extensive comparison between these and other proposed techniques.

$$P_{KN}(w_i | h_i) = \begin{cases} \frac{\max(C(h_i w_i) - \delta, 0)}{C(h_i)} & \text{if } C(h_i w_i) > 0 \\ \alpha(h_i) P_{KN}(w_i) & \text{Otherwise} \end{cases}$$

Eq. 2.7

An interesting characteristic shared by all these smoothing methods is that they are frequently combined using two well-known techniques: back-off and deleted interpolation. Both also contribute to solve the problem of zero frequency n-grams. In this case, when a particular n-gram does not exist it is possible to estimate its probability by using lower order n-grams (which tend to be better estimate than the very sparse high-order models).

$$P_{backoff}(w_i | w_{i-n+1} \dots w_{i-1}) = \begin{cases} (1 - d_{w_{i-n+1} \dots w_{i-1}}) P(w_i | w_{i-n+1} \dots w_{i-1}) & \text{if } N(w_{i-n+1} \dots w_i) > k \\ \alpha(w_{i-n+1} \dots w_{i-1}) P_{backoff}(w_i | w_{i-n+2} \dots w_{i-1}) & \text{else} \end{cases}$$

$$\text{Where } \alpha(w_{i-n+1} \dots w_{i-1}) = \frac{1 - \sum_{w_i: c(w_{i-n+1}^i) > 0} P(w_i | w_{i-n+1}^{i-1})}{1 - \sum_{w_i: c(w_{i-n+1}^i) > 0} P(w_i | w_{i-n+2}^{i-1})}$$

Eq. 2.8

A back-off n-gram model is a nonlinear method proposed by [Katz, 1987]. According to Eq. 2.8, in the Back-off technique, the probability of an existing n-gram is calculated using the Maximum Likelihood Estimation (MLE, computed directly by dividing counts) or through a recursive utilisation of lower level conditional distributions if it does not exist. In this way, when a given n-gram is not available (i.e. its count is zero or below a given threshold in the training text) its probability is calculated using the occurrence count of a lower order model (n-1 gram) instead. In this equation, $\alpha(w_{i-n+1} \dots w_{i-1})$ is a normalization factor, called back-off weight, that is calculated offline and represents how much probability mass has to be distributed from the high order model into the n-1 gram model. In addition, the equation shows that the MLE estimates are discounted with a certain amount, d , that is distributed among the unseen n-grams whose probability is calculated using the back-off.

The other technique is the deleted interpolation or Jelinek-Mercer smoothing [Jelinek and Mercer, 1980]. Here, the probability for a given n-gram is obtained using a linear interpolation of high-order models with lower-order distributions (see Eq. 2.9). Where the λ values are calculated using held-out data and the final model in the recursion correspond to a uniform model. In this case, the probability of the n-gram relies always on the probabilities given by the high order and low order models, even if there is zero evidence for the high-order model. In [Goodman, 2001] the conclusion is that this kind of models performs better on small training sets. For this reason, in this thesis we used this smoothing algorithm in the experiments for speech-to-sign language translation (section 6.2.3, page 174).

$$p_{J-M}(w_i | w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} p(w_i | w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) p_{J-M}(w_i | w_{i-n+2}^{i-1})$$

Eq. 2.9

Finally, in addition to the techniques described above, in the literature we can find other strategies that also try to reduce the effect of low frequency n-grams. Among them, we can

mention variable length n-grams [Kneser, 1996], or skip-gram [Ney et al, 1994]. In variable length n-grams the number of words in the context is not fixed but depends on the context itself. The algorithm implements a pruning strategy where n-grams that have a small impact on the model are discarded. Then, using the remaining set of n-grams the smoothing parameters are recalculated. The main idea here is to provide a better distribution of discounted mass among the remaining n-grams. Skip-gram models are based on the intuition that when using high-order n-grams the probability of having seen the exact context is low, however the chance of having seen a similar context (i.e. most of the words occurring) can be high. In this method, several combinations of skipping words can be done whose probabilities are linearly interpolated with the full n-gram probability. Finally, a completely different approach for fighting the data sparseness is to calculate n-gram probabilities in a continuous space using neural networks. In order to do it, [Schwenk, 2007] proposes a linear interpolation between a traditional n-gram model with the probability given by a neural network trained with highly efficient algorithms.

2.2.1.2 Long span and syntactic information

One of the simplest ways to include some sort of semantic information in a language model is the so-called class-based language models. In this kind of models, statistically related words or phrases are clustered together in classes in order to train a new n-gram model with the indexes of the classes. The big advantage of these models is its robustness against infrequent events and its generalization capability about words used in contexts that have not appeared explicitly in the training data. The reason is that several words, some more frequent than others, can belong to the same class and each one contributes to the estimation of the final probability (see Eq. 2.10). Besides, they allow the system to dynamically modify the vocabulary without retraining or recompiling the entire language model.

In general, the classes can be created by hand using linguistic information [Jelinek, 1990](i.e. Part-Of-Speech models) or automatically inferred. The former is the preferred for narrow discourse domains or when the size of the training data is not big. The latter is preferred for unconstrained tasks and when the size of the training corpus allows the creation of reliable models. Several automatic and iterative algorithms have been proposed. The most frequent ones are the agglomerative hierarchical clustering (bottom-up) proposed in [Brown et al, 1992] or the divisive algorithm (top-down) proposed in [Kneser and Ney, 1993]. In both algorithms, the clustering process is measured using the information gain distance and the process is stopped when the desired number of clusters is reached or the information gain is below a given threshold.

$$P_{\text{class}}(w_i | w_{i-n+1} \dots w_{i-1}) = P(w_i | C(w_i)) P(C(w_i) | C(w_{i-n+1}) \dots C(w_{i-1}))$$

Eq. 2.10

Eq. 2.10 shows one of the several alternative formulas used to calculate the word n-gram probability using class-based LMs. In this case, the conditional probability of a word w_i given the history $w_{i-n+1} \dots w_{i-1}$ is calculated as the product of two factors: the probability of the given word (w_i) belonging to class $C(w_i)$ and the probability of the class given the preceding classes.

In spite of its robustness, the main disadvantage is the loss in the ability to distinguish between different histories, although a quick solution is to increase the order of the model at the expense of increasing the number of infrequent events. In practice, this kind of models are usually interpolated with a word n-gram model. Another important characteristic that affects the performance of the model is the quality of the classes. When there is not enough training

data, good results can be obtained using hand-made classes (i.e. linguistically motivated); however, when the size of the corpus is increased, the automatically inferred classes perform better because they are more dependent to the domain.

As mentioned above, the set of classes can be created by hand using POS tags (i.e. verbs, nouns, adjectives, etc). Since the number of tags is reduced, in comparison with the number of words in a traditional n-gram model, it is possible to train a high order n-gram model where the words in the original text are replaced by its POS tags according to its grammatical function given a recent history. However, the main disadvantage is that it is very time consuming and expensive to manually parse a full corpus. A possible solution is to use an automatic tagger, though it may introduce mistakes in the labelling process mainly due to the reduced context used to predict the tag (it is common to use just the last two words in the history), and from the fact that the tagger could have been trained with text from a different domain. In spite of these problems, [Heeman, 1999] presents experiments where the POS model produces better results when compared to an automatic class-based model or a back-off model, because the probabilities are better estimated.

Finally, a more complex approach, called structured language modelling, is presented in [Chelba and Jelinek, 2000]. In this approach, the hierarchical nature of the language is taken into account through syntactic information at the sentence level. In this model, a grammar parser is used in order to obtain the syntactic structure of the sentence. The reported results show improvements on perplexity and WER (Word Error Rate) when the model is interpolated with a word-based n-gram model. The big advantage of this kind of models is that the generated sentences are more grammatically consistent. However, since the quality of the results highly depends on the quality of the parser, this approach is restricted to well known languages (i.e. languages with enough data to train the parser and with a properly defined grammar).

2.2.1.3 Dynamic and topic dependent models

These models try to take advantage of the dynamic nature of the human language and its highly heterogeneity, with varying topics, genres, and styles. In general, these kind of dynamic models are linearly interpolated with a more robust static n-gram model in order to obtain improvements. Additionally, the interpolation parameter could be modified according to the most relevant topic for a given sentence or dialogue state.

The simplest technique is Cache Models [Kuhn, 1988]. In this technique, a dynamic language model is created with the N most recent words in the history. Then it is interpolated with a static language model trained with the entire corpus. The premise is that a word that has appeared before has more probability to occur later on. [Jelinek et al, 1991] report reductions on WER using this method. Additional improvements are reported in [Clarkson and Robinson, 1997], where the contribution of each word to the cache probability decays exponentially over time. In this way, words that are more recent contribute more to the cache probability. A similar method, called trigger pairs, is presented in [Lau et al, 1993]. In this model, the presence of a particular word in the history is tied to the appearance of another one; therefore, the probability for that triggered word can be increased. However, results presented in [Rosenfeld, 1996] show that in most of the cases, the inducted trigger pairs are self-triggers that result only in a generalization of the cache model.

Finally, another kind of dynamic model is described in [Iyer and Ostendorf, 1999]. Here, the training texts are partitioned automatically in clusters representing the different topics in which the texts can be classified. Then, different LMs are created one for each cluster and combined later on using different interpolation weights and formulas. For creating

the clusters, they propose a similarity distance where the *tf-idf* weight (term frequency–inverse document frequency) is used. The *tf-idf* evaluates the importance of a word to a document given a collection of documents. The importance is proportional to the number of times a word appears in a given document taken into account the total number of times it appears in the whole corpus or collection of documents. The *tf-idf* is calculated as the product of the *tf* and *idf* terms. Since in this thesis we also applied a similar concept for creating the ranking of n-grams for the LID system (see section 6.1.2.4, page 159), below we will provide more details regarding the *tf-idf* formulation.

The *tf* term is calculated using Eq. 2.11. In the equation, n_{ij} is the number of occurrences of term i in document j , and the denominator is the total number of terms in document j . In this equation, the normalization is useful to reduce the effect of very frequent terms without considering the actual importance of those terms in the whole document; this is especially relevant in longer documents.

$$tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}$$

Eq. 2.11

The *idf* term, see Eq. 2.12, provides a measure of the general importance of a term across all documents. It is calculated applying the logarithm of the quotient of dividing the total number of documents in a corpus by the number of documents containing a specific term.

$$idf_i = \log \frac{Num_Docs}{\sum_{\forall D, t_i \in d_i} 1}$$

Eq. 2.12

Another possibility for creating the clusters is based on using the concept of bag-of-words, in which a text or document is represented as a disordered collection of words, i.e. the order of the words or any grammar information is discarded. This model has been widely used in several different tasks such as document classification and information retrieval. In relation to language modelling, the most relevant algorithms that use this paradigm are: latent semantic analysis clustering (LSA)[Bellegarda, 2000a][Deerwester et al, 1990], probabilistic LSA (pLSA)[Hofmann, 1999], and Latent Dirichlet Allocation (LDA)[Blei et al, 2003]. In LSA, for instance, hidden semantic relations between words can be obtained using a term-document matrix (i.e. the occurrence of different and relevant terms along different documents). LSA applies a singular value decomposition to find a low-rank approximation of the term-document matrix. The goal of reducing the rank is to decrease synonymy, polysemy, sparsity and noise, and to discover new dependencies among different terms and documents. In [Bellegarda, 2000b], LSA is used for generating dynamic n-grams that are interpolated with a conventional word-based n-gram model in order to improve a speech recognition system. pLSA is the probabilistic version of LSA that provides a more intuitive model together with slight improvements since it can be used directly on the speech recognition system as it provides normalized probabilities. Finally, LDA is similar to pLSA except in that the topic distribution is assumed to have a Dirichlet prior, instead of a uniform distribution. This way, LDA is more robust than pLSA, and does not present problems of over-fitting and it is able to generalize when used with unseen documents. It has been successfully applied,

although with discrete reductions on perplexities and WER, to different kinds of tasks for language modelling such as dynamic interpolation with traditional n-grams [Tam and Schultz, 2005], for unsupervised adaptation as marginal constraints [Tam and Schultz, 2006], and for clustering of topic sentences to train different language models [Heidel et al, 2007]. In this thesis, we have not applied any of these techniques leaving this task for future developments and research.

2.2.1.4 Adaptation and interpolation

As mentioned in section 2.2.1 (page 25), one of the main problems when training n-gram based LMs is to have enough training data to obtain reliable models, especially when the order of the model is high. This fact is relevant since the model will fail due to the poor estimation of unknown events and because there will be too many low frequent n-grams that will obtain a high probability mass to distribute between them. This way, the probability of the good n-grams, i.e. the most frequent and reliable ones, will be discounted and its discriminative power and estimation will be diminished. In [Bellegarda, 2004] a complete survey of adaptation techniques to overcome these problems is described. Below we describe some of the most relevant methodologies.

In general, the idea is to build two LMs, one trained from the in-domain text and another one from out-of-domain data or a background corpus (i.e. bigger than the in-domain but probably less specific), and then to apply an adaptation formula that tries to modify dynamically the well estimated background model using the information from the in-domain model. This way, the probabilities of the new model will be more robust and better estimated. One of the most common adaptation techniques are the linear or the log-linear interpolation [Broman and Kurimo, 2005] that operate at the probabilities given by each model at sentence level. Another solution, when the size of the adaptation/in-domain data is small, is to use the estimations of the unigrams to adapt the probabilities given by the general model as a marginal constraint. Examples of this technique are unigram rescaling [Gildea and Hofmann, 1999], fast-marginal adaptation [Kneser et al, 1997] or fill-up models [Besling and Meier, 1995]. A more complex, but promising technique for combining several sources of language models is the Maximum Entropy framework [Rosenfeld, 1996]. This kind of framework has been successfully applied for combining trigram, class-based, cache and trigger pairs models with good results but requiring a lot of time to train.

[Galescu et al, 1998] report another strategy that uses texts from other domains by using phrase templates. The idea is to find sentences with similar structure in both domains to make the corresponding word replacements and take advantage of the knowledge present in the original domain (i.e. new contexts, new vocabulary, more training data, etc.). In this thesis, we did not use this strategy for improving the language models for the machine translation system since the Spanish sign language has a different grammatical structure than the written Spanish language, so it is difficult to use or to find any templates from a similar domain.

Another kind of adaptation, widely reported in the literature, is the so called Maximum A-Posteriori (MAP)[Federico, 1996]. In this technique, the adaptation is performed at the counts level; the original n-gram counts of the in-domain model are modified by the n-gram counts of a background corpus. The adaptation is made using Eq. 2.13.

$$p(w_q | h_q) = \frac{\alpha \cdot C^I(h_q w_q) + \beta \cdot C^O(h_q w_q)}{\alpha \cdot C^I(h_q) + \beta \cdot C^O(h_q)}$$

Eq. 2.13

Here, C^I and C^O are the frequency counts for the in domain and out-of-domain corpora for history h_q and n-gram $h_q w_q$ respectively; α and β are weight factors, estimated empirically on a development set to reduce the bias of the estimators and to provide more or less importance to each corpus. [Bacchiani et al, 2006] report improvements on WER using a combination of supervised and unsupervised adaptation of n-gram language models to a new domain and with a small corpus using MAP. Their results show that using MAP it is possible to obtain an absolute improvement of 7.7% (from a baseline WER of 28%) when using the supervised adaptation, or 3.9% absolute improvement when using the unsupervised adaptation. In addition, it is also possible to obtain new improvements using iterative adaptations. In [Wang and Stolcke, 2007], supervised MAP and marginal adaptation [Kneser et al, 1997] are successfully combined with unsupervised language models for transcription of broadcast conversations. In this case, the proposed method is applied to adapt a 5-gram language model providing reductions on character error rate (CER) up to 20.7% over the baseline system, i.e. a static model, with a 22.4% CER, and over the traditional linear interpolation that obtained a 21.5% reduction. Given the good results obtained with this technique, we decided to use it to adapt the original language model used by the machine translation system described in section 6.2 (page 170).

2.2.1.5 Gathering of new training data

Finally, as we have seen, most adaptation techniques, including MAP, require the existence of a big background corpus to provide the general distribution of the n-grams in other domains. However, considering that it is possible that such corpus is not available or is not big enough to provide reliable estimations, a proposed solution is to artificially generate it by using generic templates and applying then filtering algorithms [Bellegarda, 2004]. Another interesting solution, proposed in [Sarikaya et al, 2005] and [Zhu and Rosenfeld, 2001], is to dynamically collect new sentences from other domains or corpora using information retrieval (IR) techniques, i.e. collecting texts from online resources or from other existing databases. Then, the new sentences are incorporated, using a supervised or unsupervised procedure, into the available texts of the original domain to be adapted.

On the other hand, the adaptation using online resources and IR techniques has been previously used in different applications with successful and promising results. For instance, it has been used to reduce the number of out-of-vocabulary words (OOV) [Bigi et al, 2004], for rescoring a n-best list in a speech recognition system through the selective adaptation of discriminative high-order n-grams occurring in the initial n-best list [Zhu and Rosenfeld, 2001], and to obtain frequencies for unseen n-grams [Keller and Lapata, 2003]. In this kind of systems, two important topics of research are the creation of good queries [Zhao et al, 2004], in order to reduce the number of times the system queries the Web, and the posterior filtering of the retrieved texts since the inclusion of non-relevant text could affect negatively the adaptation and therefore the recognition results [Yu et al, 2005].

The basic procedure is to create a list of specific n-grams (i.e. content-words, very frequent words, or poorly estimated n-grams) that the system searches on the internet. Then, from a limited number of Web pages returned by the search engine a Web crawler extracts all the sentences in the retrieved URLs. In the next step, the system first cleans the retrieved texts and extracts relevant sentences from them (e.g. those containing the terms used in the query), which are then used to create or complement the background corpus. Although, this method is relatively easy to implement and provides good results, the process of retrieving all the text sentences introduces a considerable latency and too many unnecessary sentences and words. In [Keller and Lapata, 2003] and [Zhu and Rosenfeld, 2001], these problems are reduced using another approach: instead of retrieving full sentences from a Web page they

retrieve the number of different Web pages where all the terms of the query (i.e. n-grams) appear, i.e. frequency counts. The reported results confirm that the estimations using Web frequency counts correlates well with estimations made using the crawling method. Even more, the Web frequency method has proved to provide comparable or better results for adaptation purposes on different tasks, even with different techniques (e.g. linear interpolation or entropy models).

In this thesis, we have followed a similar approach, using the Web frequency counts instead of retrieving full Web pages. However, we have implemented a new algorithm that introduces some differences in relation to the ones reported in the literature: a) in the mechanism for creating the list of n-grams to query the Web, b) in the process of converting the Web frequency counts from the source language to the target language in a Machine Translation system, and c) in the adaptation framework used (i.e. MAP) to modify the background counts with the converted Web frequency counts.

2.2.2 Context-Free-Grammars (CFG's)

A context free grammar is a mathematical model for modelling the structure of a natural language using a lexicon of terminal (words) and non-terminal (high level tags expressing generalizations) symbols, and a set of production or transition rules that defines how the symbols of the lexicon can be grouped and ordered together. In this formalism, sentences are generated starting from a non-terminal symbol and applying several times the conversion rules until all the non-terminal symbols are finally converted into a sequence of terminal symbols (words).

The main advantage of these grammars is that they are easy to create and maintain, present a low perplexity, and generate grammatically correct sentences. The main disadvantage is that they may restrict the final user to use, when addressing the system, only the defined set of sentences of the training data. Any other sentence uttered by the user would be misrecognized or not allowed; besides, these grammars are only useful when the vocabulary size is reduced [Pereira and Riley, 1997]. Therefore, in order to obtain good results with this kind of grammars a compromise between the size of the grammar and the speed and accuracy of the system has to be taken. For instance, a big grammar may introduce too many recognition errors, then degrading the system accuracy; on the other hand, a small grammar will produce too many out-of-grammar recognitions, producing too restrictive or non-natural interactions with the final users. On the other hand, in a conventional CFG all the transitions have equal probability, however it is possible to include probabilistic information to the transition rules in order to define some alternatives (the sequence of rule expansions) as more probable than others. This way, it is possible to adapt the grammar to new domains or to a particular dialogue state [Mohri, 2000].

Despite its drawbacks, CFGs are the most extended type of grammars for spoken dialogue systems with system or mixed initiative capabilities. Besides, most of the current development platforms allow the creation and debugging of this kind of grammars, supporting several specification languages such as JSGF (Java Speech Grammar Format), GSL (Nuance Grammar Specification Language), and SRGS (Speech Recognition Grammar Specification), and including tools to convert grammar files from one format to others.

In relation with our development platform, during the GEMINI project an assistant was created that allows the creation and edition of this kind of grammars for compatibility with the VoiceXML specification (see section 3.4.6.2, page 72). This assistant supports JSGF and XML-based formats. In addition, new assistants, developed in this thesis, allow the

possibility of debugging JSGF grammar files and the automatic creation of stochastic grammars from a JSGF file (see section 4.7.1.3, page 120). Finally, our runtime platform also supports speech grammar files in SRGS and JSGF format (see section 3.5.3, page 77).

2.3 Language Identification (LID)

In order to allow the runtime platform to automatically detect the language to be used by the system to communicate with the user, we have worked in improving a LID system. Specifically, we have worked on a PPRLM-based system where we have introduced a new kind of long-span language model described in detail in section 6.1 (page 150). In this section, we want to present an overview of the most recent and widely used techniques for LID, as well as the reasons to select the PPRLM technique. Afterwards, we will provide a detailed description of this technique including also information about its strengths and weaknesses.

Several techniques have been suggested in the last years for LID. Probably the most extended technique is the Parallel Phone Recognition followed by Language Modelling (PPRLM) [Zissman, 1996][Zissman and Berkling, 2001]. In PPRLM, the language is classified based on statistical characteristics extracted from the sequence of recognized allophones. The idea is to use N phone parallel recognisers followed by M language models, one for each language to be identified, trained with the phoneme sequence obtained for each recognizer during the training step. During the classification step, the unknown utterance is transcribed using each of the N recognisers. Then a score is calculated for each of the N transcription using the M language models. Finally, a backend classifier selects as target language the one with the higher score.

Although PPRLM provides a high LID performance, it may require a heavy computational demand that limits its use in real-time or low cost applications. An alternative approach, and may be the most simple and popular technique, is the Gaussian Mixture Model (GMM) [Zissman, 1996]. In this technique, a model of multiple Gaussians is trained for each language using the cepstral and delta MFCC coefficients obtained from all the audio files available for each language. During the recognition, the system calculates the cepstral and delta coefficients for the unknown utterance, selecting as best hypothesis language the one that maximises the log-likelihood between the new vector and each trained model. This technique is interesting because the identification is performed very quickly, which is useful for real-time systems, and because it does not require orthographic or phonetically labelled corpus. However, for these reasons, it could not get the same accuracy rate obtained with more complex techniques such as PPRLM.

[Torres-Carrasquillo et al, 2002a] present a variation to the GMM technique called GMM-Tokenizer. In this case, the classifier output (i.e. the indexes of the GMM models) is used to train a “language model”. This technique uses both acoustic information and sequence information, so it seems to be suitable and has the same advantages as the GMM alone: labelled data is unneeded and it is faster than the phone-based approaches. However, this technique is not as good as PPRLM, but can outperform it if both techniques are combined. Therefore, it offers complementary information to the task, but increasing the CPU time due to PPRLM.

[Torres-Carrasquillo et al, 2002b] describes a variation on the GMM technique where instead of using MFCC coefficients they use a technique proposed by [Bielefeld, 1994] called shifted delta cepstrum (SDC). In this technique, the final vector used as input to the

GMM classifier at time t is given by a concatenation of k blocks of size N , with a time advance and delay d , and a p time shift between consecutive blocks. Eq. 2.14 shows the formula used to calculate the value for the i^{th} block at time t in function of the values for p and d . [Torres-Carrasquillo et al, 2002b] describe experiments where using these coefficients a GMM based LID system provides similar results than a PPRLM based system with the advantage of not requiring any orthographically or phonetically transcribed speech data and with a greatly reduced computational cost required for real-time systems. [Yin et al, 2006] also propose using SDC as a mean to introduce prosodic information such as intensity and pitch with very good results. In our current system, we have not incorporated these coefficients yet but we propose it as future work.

$$\Delta c(t) = c(t + iP + d) - c(t + iP - d)$$

Eq. 2.14

[Navratil, 2001] presents an interesting variation to PPRLM contributing with different ways to combine the information of the allophone sequence with language dependent acoustic models. In this system, the language model score is provided through a linear interpolation between an n -gram based LM and a tree-based LM in order to capture long-span information. On the other hand, the acoustic component consists of a set of language-dependent Gaussian Mixture Models (GMM) trained for each allophone using information about energy, cepstral and delta coefficients, and duration (i.e., prosodic information). Finally, the backend classifier is an ANN that integrates all the information provided by the two components. In order to reduce the computational load produced by using parallel recognizers in PPRLM, they propose to use a single recognizer with a set of multilingual phonemes including all the possible units from the languages to be recognized. Unfortunately, this approach requires labelled transcriptions in order to train language dependent n -gram LMs that are integrated into the recognizer in order to produce the N different phoneme sequences as in PPRLM. The reported results show that the acoustic information provides relevant information that improves the results considerably. In our system we have also arrived to similar conclusions after including acoustic information at sentence and phoneme level. Although we do not use tree-based LMs we agree that capturing long-span information is also useful for improving system results.

[Gauvain et al, 2004] describe an extension to the PPRLM framework by using phone lattices, both for training and testing, instead of using only the most likely phone sequence. One of the interesting contributions of using this approach is that the phone lattices offer more accurate n -gram frequencies estimates in order to train the n -gram based LMs. Finally, an artificial neural network (ANN) is used as backend classifier instead of using the average score estimated for each phone recognizer, obtaining the largest improvements when the audio segment is large (i.e. more than 30s). In our case, since most of our audio files are short we decided not to use the ANN but a Gaussian Classifier.

[López-Moreno et al, 2008] propose a new machine learning fusion scheme, called Anchor-model, to create the final feature vector used for the backend classifier. In this approach, the system exploits the relative behaviour of all the different sub-systems (e.g. phonotactics models) to a given speech utterance. Then, the relative behaviour of each language in comparison with the competing ones is modelled by using a SVM model, one for each language to be recognized. The idea is interesting since the system uses the information of the relative behaviour of the scores produced by each sub-system for a given input language instead of using a single vector. This way, the system learns how the scores given by the sub-system that models a given language tend to be higher than the scores produced by the non-target subsystems. Then, at recognition, the system uses similarity functions between

the new composite vector of all anchor models and the trained anchor model space. The proposed technique outperforms other fusion schemes based on different kernels applied to a SVM model. In our current system, we have not used this kind of fusion, but we propose a simplified model using instead the differential scores between competing languages as input vector to the Gaussian classifier.

[Ramasubramaniam et al, 2003] propose a slight variation of PPRLM called Parallel Phone Recognition (PPR). The main difference between PPRLM and PPR is that the sequence of allophones generated by each phone recognizer is used as input to only the corresponding language-dependent LM instead of making them to go through a bank of M different n -gram language models. This way, it is possible to evaluate the contribution of the acoustic and phonotactic information separately, or it is also possible to integrate the phonotactic and acoustic models into one-step allowing that the phone recognizer can use the language model for constraining the Viterbi decoding rather than applying the constraints after the phone recognition is complete. As a consequence of this approach, the phone recognizer produces the most likely phone sequence that is also optimal regarding the combination of the acoustic and phonotactic information. In this paper, they also propose the bias removal procedure to improve the classification results. In addition, a mono Gaussian classifier (GC) and a K-Nearest Neighbour Classifier (KNNC) are also compared. According to their results, the bias removal performs better than the GC, and the GC better than the KNNC. However, probably they could obtain better results with the GC using a mixture of Gaussians instead of using just one Gaussian.

[Sai-Jayram et al, 2003] propose a modification of PPR, called Parallel Sub-Word Recognition (PSWR), where instead of using phone-based HMMs for the recognizer they use sub-word units created by automatic segmentation, segment clustering and segment HMM models. This idea is interesting since labelled data is not required. Briefly, the technique consists of splitting each utterance into segments with a minimum duration, and then applying an agglomerative clustering of those segments to train HMM models from the final clusters. They compare the results provided by the acoustic, language models, and mixed scores but do not provide results integrating all this information. According to their results, the proposed framework performs similar to PPR with the advantage of requiring not any labelled data.

[Nagarajan and Murthy, 2004] present a similar technique to PSWR but in this case using an HMM of syllables-like units. The training process does not require transcriptions and the syllabic units are automatically defined using an unsupervised incremental clustering. The results are good but the improvement is low. However, it is interesting that a better improvement can be obtained when the more discriminative (language specific) units are used. A similar behaviour we have observed in our experiments.

[Gleason and Zissman, 2001] present comparative results between using an ANN and a single Gaussian classifier as the backend. In this case, the ANN performs slightly better. However, they do not provide results using a multi-Gaussian classifier as the one we use in our system.

A recently technique proposed in [Ma et al, 2005] is based in a new concept called Bag-Of-Sounds models (BOS) of phone-like units (e.g. acoustic segments). The BOS concept is inspired in the corresponding Bag-Of-Word (BOW) model used in information retrieval (see section 2.2.1.3, page 31). Their proposal is to convert the utterances into a count vector matrix, similar to the document-word matrix used in BOW, where “documents” correspond to the different languages to recognize, and “words” correspond to the set of all possible unigrams and bigrams generated using a universal inventory of phone models. Then, they

apply a normalized entropy quantity to the count vector matrix in order to model the discriminative power of each phone considering the entire set of training documents. Finally, a Support Vector Machine (SVM) is used as backend classifier. In relation with our system, we have experimented with different entropy quantities for creating a ranking of discriminative n-grams; in our case, we tried the *tf-idf* term and proposed several other alternatives.

Finally, [Li et al, 2006] present a new line of research where several sources of information are combined. In this case, they propose the combination of a PPRLM-based system and a Bag-Of-Sounds model to obtain corresponding scores for all target languages and then concatenating them to form an utterance-level score vector. Then this composite score vector is fed into an ANN and a simple linear discriminant function (LDF) in order to generate two confidence scores for each language which are then fused and sent to the backend classifier to make the final selection. An important conclusion from this study is that the confidence scores from both classifiers (ANN and LDF) and the information provided by the PPRLM and BOS system exhibit large diversity that is ideal for score fusion. In our work, we have also focused on this kind of characteristics using information from the PPRLM system and from an n-gram frequency ranking that provides utterance-level information similar to the BOS system.

2.3.1 Description of the PPRLM Technique: Advantages and Disadvantages

As mentioned above, PPRLM is the most popular approach to language identification. The main objective of PPRLM is to model the frequency of occurrence of different allophone sequences in each language. Following the diagram in Figure 2.5, this technique can be divided in two stages. First, several parallel phone recognizer take the speech utterance and outputs a sequence of allophones corresponding to the phone sets used for each one. Then, the sequence of allophones is used as input to a bank of n-gram language models (LM) in order to capture phonotactics information. In the second stage, the language model scores the probability that the sequence of allophones corresponds to a given language.

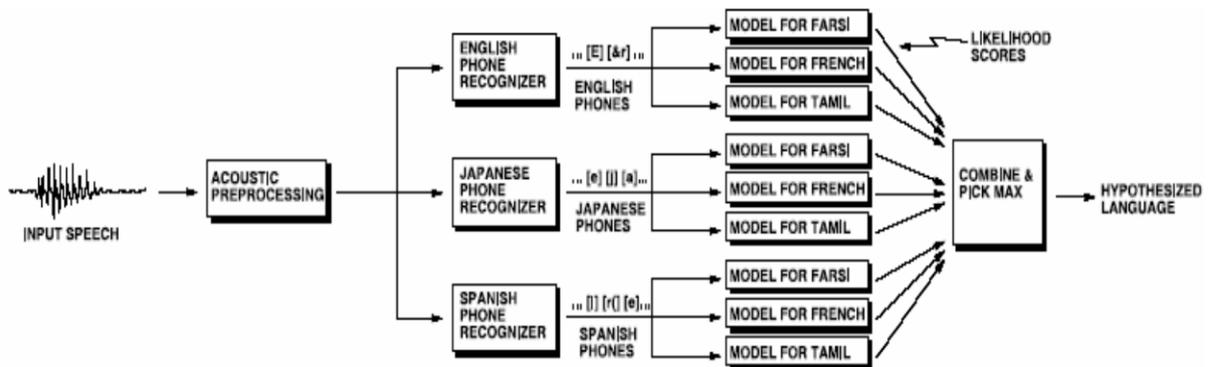


Figure 2.5. Diagram of a PPRLM LID system (Source: [Zissman, 1996])

During test, the unknown utterance, U , is hypothetically identified as language L following Eq. 2.15. Here, L is the set of all possible and equiprobable languages to identify, S is the phone sequence, and Φ is the set of allophone acoustic models usually estimated using HMM phone models. The prior probability of S , $P(S|L)$, is estimated using a phone n-gram LM. Several approximations can be applied to this equation in order to simplify the calculations, for instance to consider that the phone models are language independent by not using any phonotactic constraints for phone decoding.

$$L_{hyp} = \arg \max_L \sum_S P(U | S, L, \Phi) P(S | L)$$

Eq. 2.15

The main advantages of PPRLM are: a) As it uses many recognizers, it is possible to cover most of the phonetic realizations of every language. b) PPRLM makes possible to have phone recognizers modelled for languages different to the languages that have to be identified, which is especially useful in situations when the training data is not enough to obtain reliable language dependant models. Obviously, if there is a match between the input language and the language of the models the performance will be better, because it will model explicitly the phonetic variations of each language.

In spite of the good results obtained using PPRLM, it presents some weaknesses that in the literature have been solved in different ways.

1. The processing time is multiplied by the number of recognizers. This fact limits the use of PPRLM for real time systems that require recognizing a high number of languages. In this case, the recent incorporation of faster and multiple parallel processors in new computer machines help to alleviate this problem. Another solution, is the optimization of the algorithms using dedicated software libraries and hardware that can reduce the elapsed time required during training or testing. In our case, we have just take advantage of the new capabilities given by current computers.
2. The presence of bias in the log-likelihood scores generated by each combination of the N recognizers and M language models. This problem is mainly due to the differences between the allophone dictionaries and training data used by each recognizer [Zissman, 1996]. [Ramasubramaniam et al, 2003] describes two solutions for this problem. The first solution is called bias removal; it consists on a normalization procedure using as LM score the calculated score minus the average score in the training data. Then, the language is identified using a Maximum Likelihood Classifier. The second solution is to use another kind of classifier, such as Gaussian, K nearest-neighbour, or Support Vector Machine (SVM) classifiers. The advantage of using these classifiers is that the final decision about the recognized language is not affected by the bias, because the classification is not based on using an absolute discriminant function. In this thesis, we decided to use a Gaussian Classifier given the good results obtained in [Cordoba et al, 2006a] and [Cordoba et al, 2006b]. These classifiers also benefit from the normalization of scores (e.g., the T-norm normalization). In our system, we use what we call “differential scores”, which is a similar normalization.
3. The LMs models present the same kind of problems that occur in recognition tasks, mainly data sparcity and the limitation of the n-grams to model long-span information. In this case, it is difficult to solve the first problem because it would require new training data (i.e., obtaining new recordings or using an external corpora with the same dictionary of phonemes used in our platform) consisting of a sequence of recognized phonemes. As mentioned before, data sparcity limits the performance of the most important smoothing algorithms, although it is frequent to counteract it using the deleted interpolation algorithm or other adaptation techniques (see section 2.2.1.1, page 28). In deleted interpolation, the conditional probability of a word given its context is calculated as the linear interpolation between the individual probabilities of different order n-gram models. Regarding solutions for the problem of including long span (dynamic) information to the language models, [Navratil and Zühlke, 1997] present slight improvements on the LID rate when using the skip-gram technique

proposed by [Ney et al, 1994]. [Padró and Padró, 2004] present LID experiments on written text for six languages using three different kinds of LM: Markov models, trigram frequency vectors and n-gram text categorization [Cavnar and Trenkle, 1994].

In this thesis, we have mainly focused on providing new solutions to the third kind of problems. In our case, extending the work proposed by [Cavnar and Trenkle, 1994]. Our main interest in this technique was that it combines local information (n-grams) and long-span information (collected counts from the whole utterance). In general terms, during training the technique proposes the creation of a ranked template with the N (typically 400) most frequent n-grams (up to n-grams of order five) of the character sequences in the train corpus for each language. During the evaluation, a dynamic ranked template is created for the phoneme sequence of the recognized utterance. Then a distance measure is applied between the dynamic template and each language dependent template previously trained. The selected language is the one that presents the higher correlation between templates. This technique is very simple but provides good results for language recognition of written texts (up to 93%, depending of the length of the sentence to recognize and the size of the template), it is robust against text errors, and it does not require applying any kind of smoothing technique.

2.4 Machine Translation

Machine Translation (MT) is the name for the automatic process of translating text or speech from one language to another. One of the first MT systems was presented in 1954, but it was not until the 1970s when many governments started to be interested in MT thanks to systems like the Canadian Meteo, for translating weather forecasts, and SYSTRAN²⁵ used by the European Commission. In the 1980s the first versions of PC-based MT systems appeared. Then, in the 1990s appeared the first online MT services such as BabelFish²⁶, sponsored by Altavista. Finally, during the 2000s, MT has grown considerably thanks to the creation of new efficient hybrid algorithms and the availability of training data that allows better translations. In this section, we will show the background and the most important research approaches in this field in the last years.

2.4.1 Current Approaches for Machine Translation

Current architectures for MT can be categorized into three main approaches: direct, transfer and Interlingua. In direct translation, the text is translated word-by-word using a bilingual dictionary and then reordered using simple rules. In the transfer approach, the source text is first parsed and then syntactic and lexical rules are applied to transform the parsed structure into a target parse structure that is used to generate the final target sentence. Finally, in the Interlingua approach, the source sentence is deeply analyzed and converted into an abstract language independent meaning representation (i.e. Interlingua) that is used to generate the target sentence. For a detailed description of each approach, refer to [Jurafsky and Martin, 2008]. These three approaches are represented graphically through the Vauquois pyramid (see Figure 2.6). In this figure, the vertical axis represents the effort required for

²⁵ <http://www.systran.co.uk/>

²⁶ <http://babelfish.altavista.com/>

analysis and generation, while horizontal axis represents the amount of transfer knowledge when moving up in the triangle.

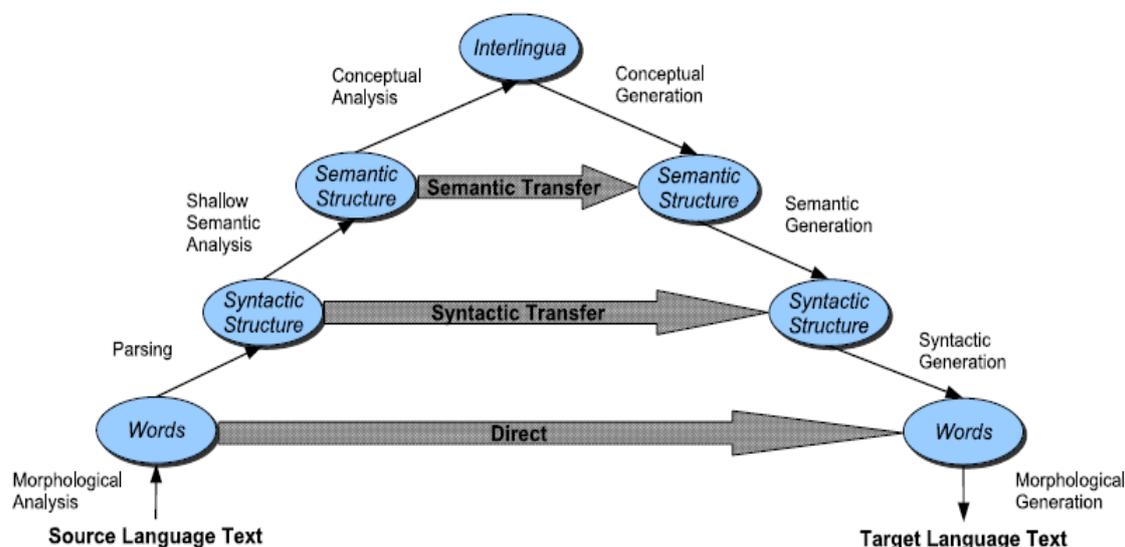


Figure 2.6. The Vauquois triangle (Source: [Jurafsky and Martin, 2008])

These three approaches constitute the classical classification for machine translation. All current systems are -in one way or another- hybrid combinations of them. On the other hand, it is also possible to classify the MT systems considering the information components applied to increase the accuracy of the translation. The most important sources of information are Knowledge-Based, Example-Based, and Statistical.

In the Knowledge-Based approach, the MT system incorporates an extensive pragmatic and semantic knowledge of the domain in the form of rules defined previously by an expert or using supervised training. Depending on the desired level of quality of the translation, the system requires a progressive in-depth understanding of the text (morphology, syntactic and semantic information), and a more complex domain model (ontology of concepts). In general, this approach provides high quality translations and is robust against recognition errors in a speech-to-speech translation system but it is restricted to small domains since the creation of the rules is very expensive and time consuming. Examples of research systems that incorporate this information are KANT²⁷ at Carnegie Mellon University and the described in [San-Segundo et al, 2008].

In the Example-Based Machine Translation (EBMT), proposed by [Sato and Nagao, 1990], the idea is to translate a source sentence by imitating a similar translation example previously trained from a parallel corpus and stored in a database that is looked up at runtime. In this kind of system, the main problems to solve are how to combine the different candidate phrases in order to obtain a coherent translation, to disambiguate when it is necessary to imitate more than one translation example, and how to solve problems with non-matching

²⁷ <http://www.lti.cs.cmu.edu/Research/Kant/>

segments of the sentence. A recent example of this kind of systems is described in [Morrissey and Way, 2005].

Finally, in Statistical Machine Translation (SMT) systems the translations are generated using statistical models where its parameters are estimated from the analysis of parallel corpus. Initially classified as a different paradigm for machine translation, in the last years the differences between the SMT and the EBMT approaches have been reduced thanks to the development of phrase-based and syntax-based models in preference to the original word-based translation. Nevertheless, some differences remain [Hutchins, 2005]. Since in the thesis we have worked on a statistical-based machine translation system, in the next section we will provide more background information about this approach and the main research work in this area.

2.4.1.1 Statistical machine translation

In any automatic language translation, the goal is to translate a text, given in some source language, into a target language. Given a source string, $f_1^J = f_1 \dots f_J$, it must be translated into a target string, $e_1^I = e_1 \dots e_J$. Among all possible target strings, the system has to choose the string with the highest probability given by Bayes decision rule, Eq. 2.16:

$$\hat{e}_1^I = \arg \max_{e_1^I} \{ \Pr(e_1^I | f_1^J) \} = \arg \max_{e_1^I} \{ \Pr(f_1^J | e_1^I) \cdot \Pr(e_1^I) \}$$

Eq. 2.16

In this equation, $\Pr(e_1^I)$ is the probability given by the target language model, whereas $\Pr(f_1^J | e_1^I)$ is the probability given by the string translation model. The argmax operation denotes the search problem, i.e. the generation of the most probable sequence of words in the target language.

The overall architecture and process for translating a sentence is summarized in Figure 2.7. The first step is to pre-process the input sentence in order to make the translation task simpler. Different kind of transformations can be done, ranging from the categorization of the words to parsing the source string, in order to obtain better and more reliable alignments that contribute to the generation of better translations. A similar process can be done to the final candidate sentence in the target language in order to re-order or improve the translation. The more complex process is the Global search where the system combines the probabilities produced by the translation model, $\Pr(f_1^J | e_1^I)$, and the target language model, $\Pr(e_1^I)$, in order to select the best candidate sentence. An interesting characteristic of the Bayes decision rule is that the language and the translation models provide independent information so that they can be trained individually.

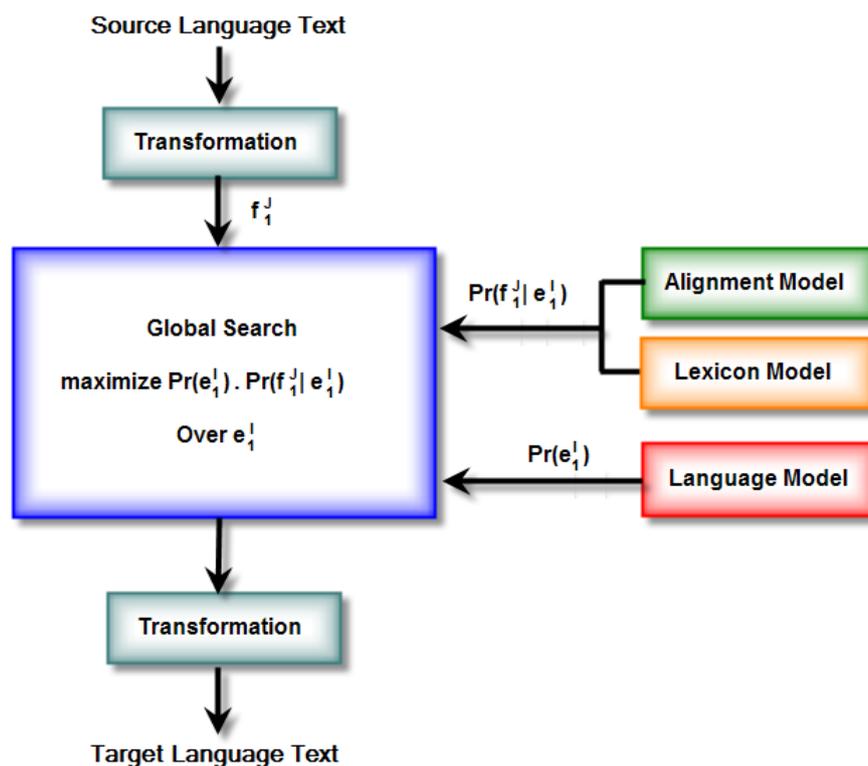


Figure 2.7. Translation process based on Bayes decision rule

Currently, most of the statistical machine translation systems are based on the Bayes decision rule, where the parameters for both the language and translation models are learnt from the analysis of bilingual corpus. In general terms, the main difference between the different proposed systems is the algorithm used to calculate the score produced by the translation model, i.e. $\Pr(f_1^J | e_1^I)$. The most important approaches are statistical finite state transducers [Casacuberta and Vidal, 2006][Bangalore and Riccardi, 2000], example-based models [Sumita, 2001], word alignment models [Brown et al, 1993][Vogel et al, 1996], and phrase-based alignments [Koehn et al, 2003]. These approaches are explained in the following paragraphs, explaining with more detail the latter two in section 2.4.2 (page 45) since they are the most relevant for this thesis.

In the statistical finite state transducers, the translation is performed using a finite state transducer where, given an input sentence, the system searches for the most likely output sentence through all possible output strings paths generated by the finite state transition network and the joint probability of sentence pairs.

In the example-based models, the system creates a dictionary with large bilingual chunks that are learnt from the parallel corpus. During the translation, the system selects the most similar source-side chunk and picks up the target-side chunk from the bilingual dictionary, applying afterwards reordering and refined rules.

In the word-based alignment models, the sentences of the parallel corpus are first aligned and then the mappings between individual words in the source language and the target language are learnt by the system by using interactive statistical methods. Afterwards, based on these alignments, the system creates a translation model that is then used to generate the most probable output sentence.

Finally, in the phrase-base alignment, the word-based alignment is conducted one-step forward by considering not individual words but word sequences, i.e. blocks or phrases, of

different lengths. Interestingly, these blocks do not correspond necessarily to linguistic phrases but to “phrases” learnt using statistical methods and that consistently appear to be corresponding translations along the corpus.

2.4.2 Word-based and Phrase-based Translation

According to Figure 2.7, the translation model can be decomposed into two models: the lexicon model and the alignment model. The first one models the probability of translating two aligned words, i.e. $p(f_j | e_i)$. The second one defines the correspondence, i.e. alignments, between the words in the source sentence and the words in the target sentence. In the basic model, some restrictions are imposed such as each source word has to be aligned to exactly one target word, and allowing that source words that are not aligned to any word in the target sentence can be modelled by assuming the existence of a null word or empty cell on the target side. Then, more complex models are allowed by introducing the concept of fertility, i.e., the possibility of aligning one target word to many source words, and a distortion penalty that penalizes implausible alignments. Eq. 2.17 shows the formula used by a basic first-order translation model based on word alignment. In this formula, e is the target sentence, f the source sentence, I and J are the lengths of the target and source sentences respectively, a_j is the position in e that f_j is aligned with, $p(J|I)$ is the length model, $p(a_j|a_{j-1}, I, J)$ is the alignment model, and $p(f_j|e_{a_j})$ is the lexicon model. Observe that the formula sums over all possible alignments considering the restrictions mentioned above. The formula also shows the the alignment and lexicon models.

$$p(f_1^J | e_1^I) = p(J | I) \sum_{a_1'} \prod_{j=1}^J p(a_j | a_{j-1}, I, J) \cdot p(f_j | e_{a_j})$$

Eq. 2.17

Currently, most word-based alignment models are based on the work presented in [Brown et al, 1993] namely the IBM 1-5 models, and the HMM models proposed by [Vogel et al, 1996]. Below we describe them briefly. For more information about these models and machine translation refer to [Manning and Schütze, 1999] chapter 13, [Jurafsky and Martin, 2008] chapter 25, [Chou and Juang, 2003] chapter 11, and the educational tutorial presented by [Knight, 1999]. During the training process, since each model presents many free parameters that have to be optimized, the algorithms only provide local minimums. In order to reduce these problems, the training procedure is initialized with a simple model that does not present local optima. Then, the parameters of the simple model are used to initialize the training process of more complex models.

In IBM-1 model, it is assumed that all the alignments are equally likely, i.e. it presents a uniform alignment probability. The big advantage of this model is that it converges to a global maximum therefore in most of the cases it is used as seed for the following models. IBM-2 makes a more realistic assumption using a zero-order model, i.e. the system only considers a dependence on the absolute position of the source word. In IBM-3, the algorithm uses a zero-order inverted model, i.e. a mapping from the target positions i to the source positions j , introducing new parameters such as the fertility model, the distortion model, and the spurious model. The fertility model gives the probability of aligning many source words to one target word, the distortion model gives the probability of aligning two word positions in the source and target sentences conditioned on the sizes of both sentences, and the spurious model gives the probability of assigning a source word to the empty word. IBM-4 is similar to IBM-3 but uses a first-order model where the algorithm introduces dependencies on the

previous alignment, the identity of the source word, and the previous target word. The idea of these dependencies is to reduce some problems introduced by the fertility model and the process of specifying the final positions of the target words. In order to reduce the dependencies on the identity of specific words that can result in an unreliable model due to scarce training data, it is possible to use corresponding word classes or part-of-speech as suggested by [Och, 1999][Kneser and Ney, 1993] and [Brown et al, 1993]. IBM-5 can be considered as an improved version of IBM-4 where the probabilities are strictly normalized, which means that the algorithm does not put probability mass on alignments or events that can never occur. This way, the alignments are better estimated but at the cost of a more complex and delaying algorithm. Finally, the HMM model can be considered as an improved version of the IBM-2 model since it uses a first order model, i.e., considering dependencies on the position of the previous alignment, which capture localities in the translations, i.e. neighbouring source words are often aligned with neighbouring target words. In the baseline HMM, the locality is captured by using absolute positions. In order to reduce the number of alignment parameters, [Vogel et al, 1996] and [Dagan et al, 1993] propose a homogeneous HMM where the alignment probabilities depend only on the jump width ($a_j - a_{j-1}$). Finally, [Och and Ney, 2000b] propose the context dependent HMM where an additional dependence on the identity of the source word of the previous alignment is introduced.

One of the main shortcomings of IBM-1, IBM-2, and HMM models is that they only allow to model correspondences between single words instead of longer span correspondences, i.e., modelling structural or syntactic aspects of the language. Although IBM-3, 4, and 5 models reduce this problem by introducing the concept of fertility, it is not enough to model more complex relationships. In order to reduce this problem, a new kind of translation model was proposed in [Och et al, 1999], [Yamada and Knight, 2001] and [Koehn et al, 2003] called phrase-based translation. In this kind of models, the system learns the phrase alignments by using a word alignment model trained using the IBM models and applying different heuristics to extract the final phrases. For instance, [Yamada and Knight, 2001] propose to extract only phrases that have a linguistic motivation, i.e., phrases that are constituents in well-formed syntactic parse trees. On the other hand, [Koehn et al, 2003] report better results when using long span phrases without imposing syntactic restrictions or using a more complex IBM model to start with.

The process for creating the phrase alignments is called symmetrising. The idea of this process, described in detail in [Och et al, 1999] and [Jurafsky and Martin, 2008], is to train two separate word alignments, one for the source-to-target language, and another for target-to-source language. These word alignments can be created using the IBM models or more complex models to obtain better results [Och and Ney, 2000a]. Then different combinations of both alignments are applied in order to produce the final alignments. For instance, it is possible to intersect both alignments in order to create an initial high-precision phrase alignment and then to apply the union in combination with other heuristics to add new points to the initial intersected phrase alignment. In order to improve the generalization of the alignments, the phrases are learnt by using bilingual word classes rather than using word identities. In addition to the translation probability, the algorithm also considers a distortion probability that penalizes large reordering in the final positions of the phrases in the target sentence considering the positions in the source sentence. In [Och and Ney, 2004] full details about the training and search process are described.

Current research on machine translation is oriented to the incorporation of new complementary information using log-linear models [Och and Ney, 2002] in the search process (see Eq. 2.18). This way, it is possible to combine the traditional models proposed by the Bayes rule, i.e. translation and language model, with different sources of information such

as POS-based models, word-classes, lemma-based LMs, likelihood of the parse tree and dictionary matching on the source sentence, etc. In the equation, h_m represents the different models to combine, and λ_m the scaling factors that are optimized on a development set using numerical optimization algorithms.

$$\hat{e} = \arg \max_{e_1^I} p(e_1^I | f_1^J) = \arg \max_{e_1^I} \exp\left[\sum_{m=1}^M \lambda_m h_m(e_1^I, f_1^J)\right]$$

Eq. 2.18

Another area of research is the use of factored translation models [Koehn et al, 2006], where instead of using only the surface form of words, the forms can be augmented with different factors such as lemmas, POS tags, morphologies, word classes, supertags [Birch et al, 2007], etc. The advantage of this model is that it can overcome some of the problems associated with phrase-based alignments, since they are able to capture long-span information, to respect linguistic phrase boundaries, and they allow the incorporation of more generalizations that are implicit when using syntactic knowledge.

Finally, another area of interest are the so called discriminative word alignments models [Moore et al, 2006] that can obtain equal or surpass the alignment accuracy obtained by the IBM models. In this case, the alignments are created by using human-annotated word alignments on a small set of the training data and adding arbitrary features that are linearly interpolated in order to create the final alignments. Several features are proposed in the literature, for instance: the relative distance between source and target words, a binary feature to indicate if both words are identical, number of unaligned words in a sentence, probability of translating one word into one, two, three, etc. words (i.e. fertility), etc.

2.4.3 Current Metrics for the Automatic Evaluation of Machine Translation Quality

Although evaluating the quality of a translation is a subjective and very difficult task, several automatic metrics have been proposed in order to avoid the necessity of performing this task by human evaluators. The task is quite difficult since natural language is ambiguous and complex. For instance, two sentences can contain different words but they can be equivalent, while another two can differ in only a word but have an entirely different meaning. In [Vilar et al, 2006] we show a descriptive list of the most relevant problems we can find when automatically translating two sentences between different languages. In contrast to machines, human beings are able to evaluate a translation according to two main factors: adequacy and fluency. The first one evaluates if the translation preserves the same meaning as the original sentence. The second one measures if the translation is grammatically correct. On the other hand, automatic measures are only able to evaluate the closeness between the translated sentence and a reference translation (or multiple translations if possible). In this case, the candidate sentence is ranked as better if it is closer to a human translation (references).

[Banerjee and Lavie, 2005] present a list of basic criteria for any useful and effective MT metric: high correlation with human judgment, sensitivity to differences between different systems, consistency among similar texts, reliability (different systems that score similarly should be expected to perform similarly) and generality (it has to work with independence of domain and scenario). In general, according to [Jurafsky and Martin, 2008], automatic MT metrics are not good for evaluating radically different architectures (e.g. an

Interlingua vs. a statistical-based machine translation systems), even when evaluating human-aided translation. However, they can be very useful when evaluating improvements made to the same MT system or with similar architectures.

This section describes the main metrics used for performing the automatic evaluation of machine translation systems, including those that were used in this thesis.

2.4.3.1 Word error rate (WER)

Similar to the one used in speech recognition, this metric works at word level and it is based on the calculation of the number of words that differ between a machine-translated sentence and a reference translation.

$$WER = \frac{S + D + I}{N}$$

Eq. 2.19

Where S is the number of substitutions, D is the number of deletions, I is the number of insertions, and N is the number of words in the reference. If there are multiple translation references, only the lowest rate is reported.

2.4.3.2 Position independent word error rate (PER)

Proposed by [Tillmann et al, 1997], this metric is similar to the WER but does not take into account the ordering of words in the matching operation. It considers the translations and the reference as bag-of-words and computes the differences between them, normalised by the reference length. Besides, it is guaranteed that the PER rate is less than or equal to the WER. The metric operates counting only the number of times that identical words occur in the translated and the reference sentence. Words that do not match are counted as substitutions, and depending on the translated sentence is longer or shorter than the reference translation, the rest of the words are considered as insertions or deletions.

2.4.3.3 Bilingual evaluation understudy (BLEU)

Proposed by [Papineni et al, 2002], BLEU is one of the most popular metrics for evaluating machine translation systems since it provides a high correlation with human judgements of quality. The metric tries to guarantee adequacy, assigning a higher score to sentences that use the same words as in the references, and it looks for fluency using longer n-gram matches. BLEU is formulated as the geometric mean of a modified form of n-gram precision, p_n , using weighted n-grams up to order N, multiplied by a decaying length penalty to impose that the best candidate matches the reference translations in length too.

Eq. 2.20 shows the formula for calculating the score. In this equation, N is the order of the n-grams calculated (typically $N = 4$), BP is the Brevity Penalty factor that penalizes translations that are shorter than their reference sentences. L_{ref} is the number of words in the reference translation that is closest in length to the translated sentence, and L_{sys} is the number of words in the translated sentence. The precision p_n is calculated for every n-gram order and weighted by the factor w_n (typically a uniform weight is applied, i.e. $w_n = 1/N$). Count is the number of n-grams found both in the candidate reference C and in the translated sentence. $Count_{sys}$ is the number of n-grams found only in the translated sentence.

$$Score = BP \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right)$$

$$BP = \exp\left(\min\left(1, 1 - \frac{L_{ref}}{L_{sys}}\right)\right)$$

$$p_n = \frac{\sum_{c \in \{Candidates\}} \sum_{ngram \in C} Count(ngram)}{\sum_{c \in \{Candidates\}} \sum_{ngram \in C} Count_{sys}(ngram)}$$

Eq. 2.20

2.4.3.4 NIST

Proposed in [Doddington, 2002], it is based on the BLEU metric but introducing slight modifications. First, BLEU uses the geometric mean of the n-gram precision, but NIST uses the arithmetic mean to reduce the impact of low co-occurrences for high order n-grams. Second, BLEU calculates n-gram precision using equal weights to each n-gram; on the contrary, NIST takes into account how informative a particular n-gram is (i.e. the rarer the n-gram is the larger weight it will obtain).

$$Score = \sum_{n=1}^N \left\{ \frac{\sum_{all\ n\text{-grams\ that\ co-occur}} Info(n\text{-gram})}{\sum_{n\text{-gram} \in s_i} 1} \right\} \cdot \exp\left\{ \beta \log_2 \left[\min\left(1, \frac{L_{sys}}{\bar{L}_{ref}}\right) \right] \right\}$$

$$Info(n\text{-gram}) = \log_2 \left(\frac{count(w_1 \dots w_{n-1})}{count(w_1 \dots w_n)} \right)$$

Eq. 2.21

Eq. 2.21 shows the formula to calculate the NIST score. In the equation, N is the order of the n-grams calculated (typically N = 5). The exponential factor is the brevity penalty, where β is such that the brevity penalty factor is equal to 0.5 when the number of words in the translated sentence is $2/3^{\text{rds}}$ of the average number of words in the reference translation. This way, small variations in the translation length do not affect too much to the overall score. L_{sys} is the number of words in the translated sentence; \bar{L}_{ref} is the average number of words in a reference translation, averaged over all reference translations. $Count(\cdot)$ is the number of occurrences for n-grams $(w_1 \dots w_n)$ and $(w_1 \dots w_{n-1})$ in all reference translations.

2.4.3.5 Metric for evaluation of translation with explicit ordering (METEOR)

Proposed by [Banerjee and Lavie, 2005], it is one of the latest proposed metrics for evaluating machine translation systems that shows a high correlation with human evaluators. The metric is based on using the harmonic mean of unigram precision and recall. An interesting characteristic of this metric compared to the previous ones is that it does not only use the matching of the different n-grams in the reference and in the evaluated sentence but also several other features such as exact word, stemming, and synonym matching. The sentence is scored based on a combination of different features: unigram precision, unigram recall, and a direct measure of how the words are out-of-order in comparison to the reference sentence.

The algorithm creates initially an alignment of unigrams between the reference and the translated sentence. The alignment is gradually created through successive stages not allowing one word to map to more than other single word in the reference string. Each stage

maps unigrams that have not been mapped previously and selects as the best alignment the one with the fewer number of crosses (i.e. number of intersections of two mappings). By default the first stage tries to make an exact match, the second stage uses a stemmer module, and the third one is based on using a WordNet synonym module.

After creating the final alignment, it is scored using Eq. 2.22. In this equation, the precision and recall measures are combined using the harmonic mean, weighting more the recall than the precision. Precision (P) is defined as the number of unigrams in the translated sentence that also appear in the reference translation (m) divided by the total number of unigrams in the translated sentence (w_t). Recall (R) is defined as the number of unigrams in the translated sentence that also appear in the reference translation (m) divided by the total number of unigrams in the reference sentence (w_r). Now, in order to take into account longer matches, METEOR computes a penalty (p) weight. The penalty is calculated as the number of chunks (i.e. a set of adjacent unigrams that appear in the hypothesis and reference sentences. Few chunks means that both sentences are almost equals) divided by the number of unigrams (u_m) that have been mapped. Using the penalty the effect of the F_{mean} can be reduced by up to 50% if there are no bigram or longer matches.

$$F_{mean} = \frac{10PR}{R + 9P} \qquad P = \frac{m}{w_t}; R = \frac{m}{w_r}$$

$$p = 0.5 \cdot \left(\frac{Num_Chunks}{u_m} \right)^3 \qquad Score = F_{mean} \cdot (1 - p)$$

Eq. 2.22

Although reported results using this metric present a high correlation with human evaluators, in this thesis we could not use it since it requires a stemmer and synonym module that are not currently available for the Sign Language.

2.4.4 Speech to Sign Language Translation

Nowadays, thanks to the significant improvements in automatic speech recognition, 3D animation, and statistical machine translation (SMT), it has been possible to face new challenges such as speech-to-speech and speech-to-sign language translation. In this section, we will describe different research projects and approaches, databases, tools, and standards that have contributed in the development of this kind of systems.

In relation with speech-to-speech translation, several research projects such as Verbmobil²⁸, Eutrans²⁹, Nespole³⁰, TC-Star³¹, MASTOR³², and GALE³³ have been undertaken in recent years, contributing significantly to the creation of new algorithms and applications for different tasks, domains, and vocabulary size. In general, the reported quality of the translation is good. For instance, in Verbmobil the goal was the creation of a mobile

²⁸ <http://verbmobil.dfki.de/overview-us.html>

²⁹ <http://cordis.europa.eu/esprit/src/30268.htm>

³⁰ <http://nespole.itc.it/>

³¹ <http://www.tc-star.org/>

³² <http://domino.watson.ibm.com/comm/research.nsf/pages/r.uit.innovation.html>

³³ http://www.darpa.mil/IPTO/programs/gale/gale_concept.asp

speech-to-speech translation system for bidirectional German/English and German/Japanese, allowing spontaneous speech and speaker independent recognition in a restricted domain (i.e. making hotel reservations, scheduling appointments, and travel planning) with a vocabulary size of about 10000 words. Reported results show around 80% correct translations and 90% of dialogue task completion. In TC-Star, the goal was the creation of a speech-to-speech translation system for unrestricted domains such as broadcast news and the European parliament speeches allowing bidirectional translation for English, Spanish, and Mandarin languages. In this case, the best result was around 70% of correct words when word positions are ignored, i.e. around 38% of WER. Besides, in most of the current research projects the translation is done using different types of statistical approaches such as phrase-based translation [Koehn et al, 2003], example-based methods [Sommer, 1999], finite-state transducers [Casacuberta and Vidal, 2006], and other data driven techniques. These have been favoured by new efficient training and generation algorithms [Och and Ney, 2003][Koehn et al, 2003], automatic error measures [Papineni et al, 2002], higher computational power and bigger parallel corpora [Koehn, 2005].

In relation with speech-to-sign language translation, it is based on the same technology as the speech-to-speech translation with the difference that the output is provided by an avatar. In the last years, this kind of systems has grown quickly since this technology is especially useful to help deaf people to communicate with non-deaf people, and vice versa, as human interpreters are expensive and are not always available. In addition, many deaf people have problems when reading lips, and even written texts, as they are used to the sign language grammar [Zhao et al, 2000]. Unfortunately, sign language presents a great variability depending on the country, even between different regions or populations across a country, which make difficult the research in this field. However, several studies have appeared in order to establish some sort of standardization and common background. For instance, in USA we can mention [Stokoe, 1960][Christopoulos and Bonvillian, 1985][Pyers, 2006], in Europe [Engberg-Pedersen, 2003][Atherton, 1999] and [Meurant, 2004], Africa [Nyst, 2004] and Asia [Abdel-Fattah, 2005][Masataka, et al, 2006]. In Spain, there have been several proposals for normalizing the Spanish Sign Language (LSE: Lengua de Signos Española), but none of them has been accepted by the Deaf community. From their point of view, these proposals tend to constrain the sign language, limiting its flexibility. The most significant studies have been [Rodríguez, 1991][Montserrat and Gallardo, 2004][Herrero-Blanco and Salazar-García, 2005], [Reyes, 2005], and [Parkhurst and Parkhurst, 2007]. [Rodríguez, 1991] carried out a detailed analysis of LSE showing its main characteristics and the differences between the sign language used by Deaf people and the standard proposals.

It is well known that when developing systems for translating speech transcriptions into the sign language, it is necessary to have a big parallel corpus that guarantees efficient training of the parameters for the language and translation models. Unfortunately, most of the currently available Sign Language (SL) corpora are too small or too general for training purposes. In the literature, we can find references to the following corpus and research on machine translation experiments.

The European Cultural Heritage Online organization³⁴ (ECHO) presents a multilingual corpus in Swedish, British, and The Netherlands sign languages (ECHO corpus). It consists of five children fables and several poems, a small lexicon and interviews with the sign language performers. The corpus consists of different video material including one medium

³⁴ <http://www.let.kun.nl/sign-lang/echo/>

shot of the body and one for the face, as well as two to four signers, male and female, per language. It also includes transcriptions with general, manual, and non-manual information. In addition, an annotation tool called ELAN³⁵ is also freely available in order to create, edit, view, and search annotations of the video and audio data included in the corpora.

Another interesting corpus (ASL corpus) is made up of a set of videos, partly available online, in American Sign Language created by The American Sign Language Linguistic Research group at Boston University³⁶. The database consists of short stories consisting of around 200 elicited sentences to illustrate different grammar structures, with fixed vocabulary, where the signs occur in many different context and word orders. Besides, it contains 20-25 minute dialogues between two native signers, and contains annotated data and multiple synchronized video information providing different views of handshapes used in ASL.

In [Bungeroth et al, 2006] and [Stein et al, 2006] a corpus, of around 2500 sentences, called Phoenix for German and German Sign Language (DGS) in a restricted domain related to weather reports was presented. It comes with a rich annotation of video data, a bilingual text-based sentence corpus, and a monolingual German corpus. These works also describe a statistical machine translation system that includes additional pre- and post-processing steps, using grammar parsers, to improve the translation.

[Morrissey and Way, 2005] present an example-based Sign Language translation system from English to the Sign Language of the Netherlands. The corpus for this work consists of 561 sentences with an average sentence length of eight words. An advantage of this corpus is that the annotation of the videos is time-aligned allowing the automatic extraction of different linguistic information for improving the translation.

[Stein et al, 2007] describe an innovative sign to English translation system, using image recognition processing and statistical machine translation, with promising WER results (around 20%). However, the corpus used in this work consisted of only 680 sentences; therefore, it is difficult to obtain better results with such a small training database.

[Chiu et al, 2007] describe a corpus of about 2000 sentences for the language pair Chinese and Taiwanese sign language, which is used to perform experiments on machine translation. They show that their optimization method surpasses IBM model 2.

In relation with research and available corpus in Spanish, the most important corpus is provided by the Biblioteca Virtual Miguel de Cervantes that is available at their website³⁷; it consists of several videos with poetry, literature for kids, and small pieces of classical Spanish books. Unfortunately, this corpus does not provide any transcriptions, just video content (that is common in most SL corpora), and it is very different from our current task domain. Moreover, there is not a standard representation, or grammar, for the Lengua de Signos Español (LSE), which makes the problem of data scarcity even worse. In [San-Segundo et al, 2006] and [San-Segundo et al, 2008], up to the best of our knowledge, we describe the first automatic translation system for Spanish speech to gesture in LSE. Besides, a corpus of about 500 sentences is also described. This corpus and the speech to gesture architecture are the reference used in this thesis for the experiments described in section 6.2 (page 170). In [San-Segundo et al, 2007] three different MT approaches are compared: rule-

³⁵ <http://www.lat-mpi.eu/tools/elan/>

³⁶ <http://www.bu.edu/asllrp/>

³⁷ <http://www.cervantesvirtual.com/seccion/signos/>

based, statistical phrase-based and stochastic finite state transducers. In this thesis, we have followed the statistical phrase-based approach, as we will show in section 6.2 (page 170).

Finally, in order to allow the translation from text or speech input into the sign language it is necessary to use some kind of avatar or animated agent that provides the visual representation required by deaf users. Currently, there has been an increasing interest on developing and evaluating this kind of virtual agents in spoken dialogue systems for a great variety of services and domains. For instance, [Cassell et al, 2002] describe an animated agent for an information kiosk, and [Wahlster (Ed.), 2006] describe Smartakus, an artificial life-like character with lip synchronization included in the SmartKom project to provide visual information to the user. [Cole et al, 2003] and [Ma et al, 2002] describe several tools and animated agents included in the CSLU Toolkit, which allows the creation of multimodal dialogue services for a wide range of applications and domains. The toolkit includes the CU Animate toolkit that allows the creation of arbitrary animated sequences and the synchronization of the lips of the agent with speech sounds. [Balci et al, 2007] present an open source platform, called XFace³⁸, which allows the creation of custom 3D talking heads using the MPEG-4 Face Animation standard. In addition, [Granström et al, 2002] from the KTH group describe in detail several multimodal dialogue systems where the animated agents help to increase the capabilities of the service and allow the different final users to have access to the service.

Unfortunately, most of the previous avatars are not useful for playing signs for deaf people since they do not fulfil all the requirements imposed for this task. For instance, it is important to have a fine motor control over all the body, not just the face or head, including also arms and fingers, as well as the surrounding area since it avoids occlusions when using only a frontal view and because many signs require the use of spatial information. Moreover, facial expressions are also important since they transmit emotion, disambiguate between different words, and provide new signs. Currently, there are several available commercial and academic avatars that can be used for creating digital content, reading eBooks, or providing information to deaf people. Below, a brief list of the most important ones is presented.

SigningAvatar³⁹: Created and commercialized by VCom3D, the designer has the possibility of using several different avatars that can be used to create custom digital content. The platform includes several tools for exporting a signed sentence into a video sequence in different standard formats, for creating and editing new signs, and for the automatic synchronization of lips and speech sounds. Besides, the toolkit includes a built-in library of American Sign Language that can be used for speeding up the development process.

VSigns⁴⁰: It is a 3D human-like avatar that generates VRML (Virtual Reality Modeling Language) sequences using MPEG-4 Body Animation specification. The process of signing involves the conversion of the text input into SignWriting notation (see section 2.4.4.1), which is converted into an XML representation using the SWML⁴¹ (SignWriting Markup Language) format. Then, the signs are converted into a sequence of Body Animation Parameters following the MPEG-4 specification. Finally, these parameters are used to animate the VRML-based avatar.

³⁸ <http://xface.itc.it/>

³⁹ <http://www.vcom3d.com/>

⁴⁰ <http://vsigns.iti.gr:8080/VSigns>

⁴¹ <http://swml.ucpel.tche.br/>

WebSign: Reported by [Jemni and Elghoul, 2007], it is a Web-based tool that features a 3D avatar that is used to translate written text in Web pages into a sign language animation. The representation is created using a dictionary of words and signs. The dictionary is created incrementally thanks to the collaboration of users of the site. Since signs are locally dependent, the site allows the creation of “community” dependent vocabularies or global vocabularies. In addition, they report the creation of a new descriptive specification language for the signs called SML that provides a XML-based description of the movements for the avatar, regardless if the avatar is used for signing or for other purposes.

Finally, we have to mention the European project eSIGN⁴² (Essential Sign Language Information on Government Networks) that was one of the most important research efforts in the development of tools for the automatic generation of sign language contents. In this project, the main result was a 3D avatar, called VGuido, with enough flexibility to represent signs from the sign language, and a visual environment for creating sign animations in a rapid and easy way. The tools developed in this project were mainly oriented to translating Web content into the sign language. The avatar is currently being used in local government websites in Germany, the Netherlands, and the United Kingdom, and included in a recent automatic translation system sponsored by IBM called SiSi⁴³ (Say it Sign it), which is intended to be used in different areas such as education and entertainment. In our case, we decided to use this avatar since it is highly flexible, it can be used as an ActiveX plug-in in order to be easily integrated in the runtime platform, and because it uses the HamNoSys notation (see next section) that presents advantages over other notations

2.4.4.1 Sign language transcription formats

As stated above, nowadays there are very few sign language corpora, existing only a few public corpora available that contain little or no annotation at all. To make things worse, there is not a standardised written form for the sign language. However, this annotation is very important since it is required to train the translation system and to create the dictionary used by the real-time system to translate the sequence of glosses into an enriched language that the avatar can interpret and convert into an animated sequence of movements to be played. Currently, in the literature we can find several proposals of languages such as SignWriting, HamNoSys [Prillwitz et al, 1989], SML [Jemni and Elghoul, 2007], and VHML (Virtual Human Markup Language)⁴⁴. Among them, the most important ones are SignWriting and HamNoSys.

SignWriting⁴⁵ is a bi-dimensional representation of graphical symbols that represent hand, handshapes, facial expressions, body movements, location, and contact. The notation, as illustrated in Figure 2.8, is easy to read and write, and it can be linearly encoded in computers assigning numeric codes to each symbol. However, the SignWriting specification does not contain all the linguistic details required by an avatar in order to generate properly the signs. For that reason, we did not consider it to be used in our system.

⁴² <http://www.sign-lang.uni-hamburg.de/eSIGN/>

⁴³ <http://www-03.ibm.com/press/us/en/pressrelease/22316.wss>

⁴⁴ <http://www.vhml.org/>

⁴⁵ <http://www.signwriting.org>

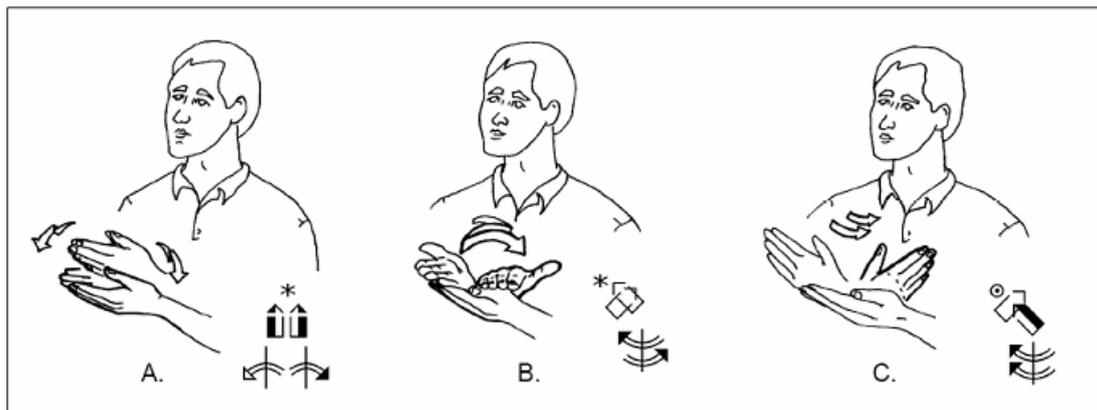


Figure 2.8. Example of SignWriting notation for the Spanish sign: Book (Source: [Parkhurst and Parkhurst, 2007])

HamNoSys, Hamburg Sign Language Notation System, developed as a research tool and a phonetic transcription system, was made publicly available in 1989. It consists of about 200 symbols covering the parameters of handshape, hand configuration, location, and movement. Although the symbols are easy to recognize, they have to be precise. Therefore they can be very long, difficult to decipher, and not easily usable for users to read or take notes. An advantage of this notation is that it is applicable to any sign language. For this reason, it has been used in several research institutions around the world and in our system. Besides, the specification is currently on development; and the new proposal is expected to include new symbols for information such as mouth movements and other facial expressions that will be used to generate new signs and to provide a higher degree of emotion to the signs. In any case, most avatars, including the one used in this thesis, that support this notation use some sort of non-standard symbols to codify this information. For all these reasons, we decided to use this notation. Figure 2.9 shows an example of one of the tools, the eSign editor included in the toolkit that allows the creation and execution of new signs and glosses using an embedded version of the avatar.

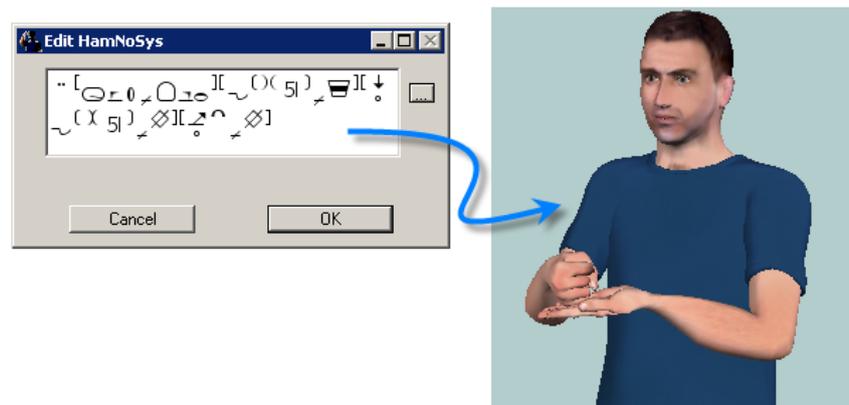


Figure 2.9. Example of HamNoSys notation and its representation using the avatar.

3 PLATFORM ARCHITECTURE

An important contribution of the GEMINI project was the design of an innovative platform that allows the specification of multimodal and multilingual services in an integrated environment. The architecture, called also Application Generation Platform (AGP), the modules that conform it and the information flow between them is shown in Figure 3.1. All the modules are independent of each other; nevertheless, they were integrated into a common graphical interface (GUI) to guide the designer in the design step by step and, at the same time, let him go back and forth. In the figure, the different colours describe the degree of implication of the author of the thesis in the creation or modification of each assistant in the platform. This way, the yellow boxes correspond to the assistants developed during the GEMINI project by other partners but that were modified afterwards by the author of the thesis with the accelerations presented in this thesis. The pink boxes correspond to the assistants that were completely designed and developed by the author of the thesis. The blue boxes correspond to assistants created by other partners of the project but that include minimum improvements and corrections introduced by the author of the thesis for compatibility with the work done, mainly in the runtime system or other assistants. Finally, white boxes correspond to assistants that were not modified at all.

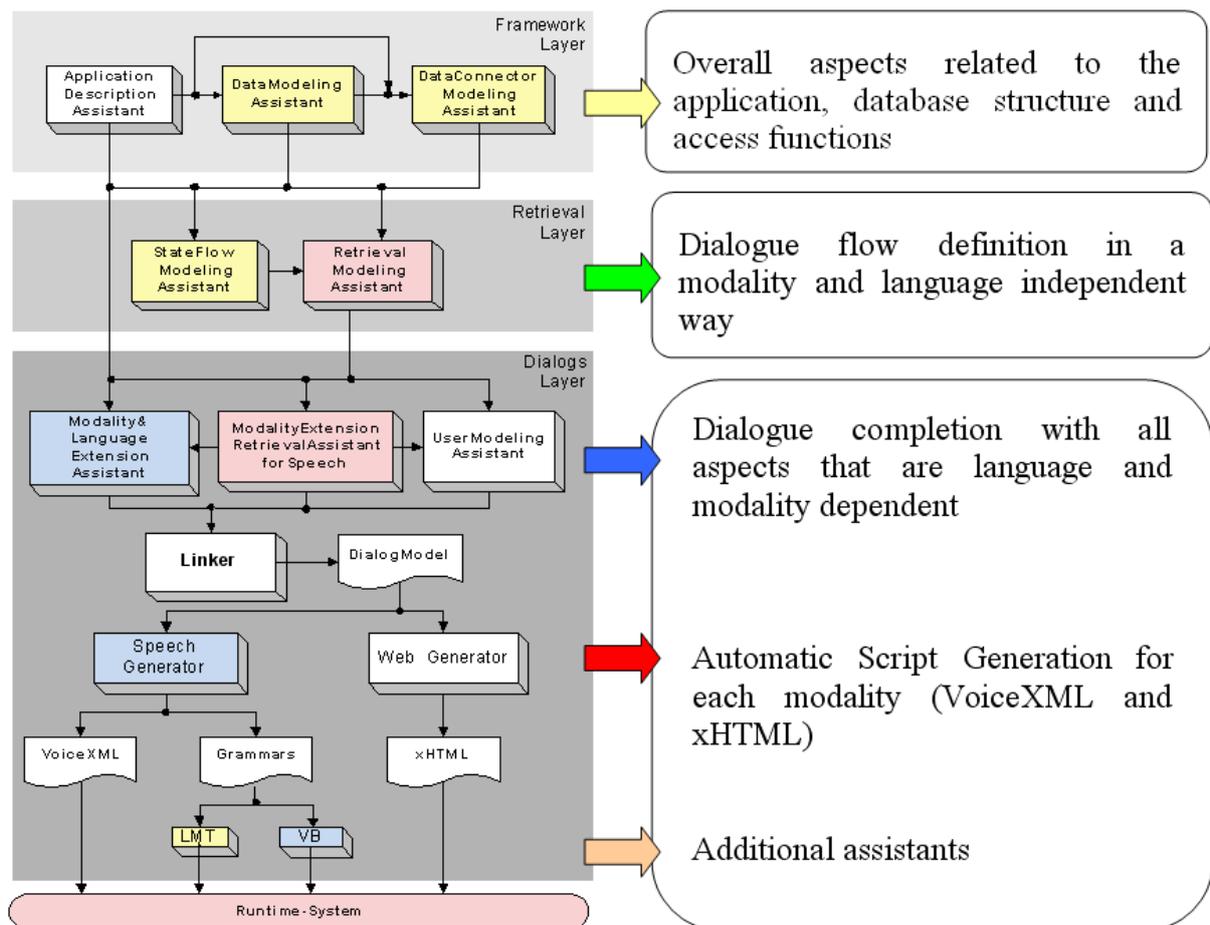


Figure 3.1. Platform architecture

The platform is divided into three main layers. The reason for this division is to separate clearly the aspects that are service specific (general characteristics of the application, database structure, database access), those corresponding to the high-level dialogue flow of the application (modality and language independent), and the specific details imposed by each modality and language. In this way, the designer is able to create several versions of the same service (for different modalities and languages) in a single step at the intermediate level.

In more detail, the assistants of the first layer are used to specify the overall aspects of the service (e.g., modalities and languages to be implemented, general default values for each modality, libraries, etc.); then, the database structure, not its contents, is described (classes, attributes, relations, etc.); and finally, the database access functions, needed for the real-time system, are defined (not their implementation).

In the second layer, the general flow of the application is modelled, including all the actions that form it (transitions and calls between dialogues, input/output information, calls to subdialogues, procedures, etc.). It is important to mention that in this layer no modality/language specific details are defined, such as prompts/grammars, recognition errors, design of the Web page, etc., as all these will be defined in the next layer. To be able to be modality and language independent, in this layer all the input/output data provided by/to the user are managed as language independent concepts.

Finally, the third layer contains the assistants that complete the general flow specifying for each dialogue the details that are modality and language dependent. Here, the prompts and grammars for each language, the appearance and contents of the Web pages, the error treatment for speech recognition mistakes or Internet access, the presentation of information on screen or using speech, etc., is defined. Furthermore, in this layer the final scripts of the service are generated, unifying all the information from the previous assistants.

As we describe in the next section, all the assistants communicate between themselves using a common XML syntax called GDialogXML. More details of the architecture can be found in [D'Haro et al, 2006][Hamerich et al, 2004a][Hamerich et al, 2004b]. From sections 3.2 to 3.5, each layer and assistant of the platform are described. To clarify the design process and the interaction with the assistants, we will show some steps of an example dialogue where a bank transfer between accounts is carried out, asking the user for the number of the source account, the destination account, and the amount of money to be transferred.

3.1 GDialogXML: Internal Descriptive Language for the Generated Models

In order to ease communication inside the platform, during the Gemini Project we participated in the development of a new object oriented abstract language based on XML tags named Gemini Dialogue XML or GDialogXML. The main feature of this specification is its flexibility, allowing the modelling of all application data, database access functions, definition of all variables and actions needed in each dialogue state, system prompts, grammars, user models, Web graphical interface, etc. Then, this information is used to carry out the conversion to the languages used for the final presentation of the service according to the modality (VoiceXML and/or XHTML). Besides, the syntax allows the addition of new modalities, and the update to new versions of the script languages generated by the platform with little effort.

As [Schubert et al, 2005][Hamerich et al, 2003][Wang et al, 2003] describe in more detail, the GDialogXML syntax provides the means needed to model the following aspects:

general concepts, data modelling, and dialogue modelling in a way both dependent and independent of modality and language. As general concepts, we can mention variable and constant definition, variable assignments, file paths, arithmetic, Boolean or string operations, control structures for loops and jumps, variable types (lists, objects, references to objects, atomic data), etc. For data modelling, we can specify the classes with attributes, which can have simple data types such as string, integer, Boolean or complex types as embedded or referenced objects or lists, supporting inheritance from base classes, etc. Regarding dialogue modelling, all dialogue models consist of dialogue modules that call each other.

During all the development of the GEMINI project, the author of this thesis constantly contributed with different proposals and minor changes to the GDialogXML syntax. To summarize, the main contributions were 1.) The definition of the template for creating mixed-initiative and over-answering dialogues (see section 4.5.4, page 106). 2.) The specification of a procedure to unset a dialogue variable and repeating the same dialogue or call to the database (see section 4.6.1, page 110), which was finally implemented through the creation of the DoFilling tag. 3.) Finally, in the definition of an internal procedure for allowing calls to non-returning dialogues from inside returning dialogues. For this case, our solution was to use a hidden dialogue transparent to the designer, which jumps to the dialogue specified by a global string variable that is previously set in the returning dialogue.

Throughout this thesis, and in order to clarify the input/output representation used in the assistants, we will include some fragments of the generated code in some assistants, giving suitable explanations of them.

3.2 Framework Layer

This layer has three assistants that allow the overall specification of the service, the description of the database structure and the database access functions.

3.2.1 Application Description Assistant (ADA)

Developed by other partners of the GEMINI project, in this assistant several overall aspects of the application, such as the number of modalities and languages, the location of some services such as the database access, database connection settings (total number of connection errors, timeouts), database path, etc., are specified; for the speech modality, the timeout values for some events such as no input, default confidence levels for speech recognition, maximum number of repetitions/errors before transferring to the operator, etc.; for the Web modality, possible errors (e.g., page not found, non-authorized, timeouts, etc.). More information regarding the error handling capabilities of the AGP can be found in [Wang et al, 2003]. Besides, the default overall strategy for dialogues is defined: system-driven or mixed-initiative.

Finally, the designer specifies the libraries, which will be used to speed up the design process. Several types of libraries can be selected containing the definition of: data models, database access functions, list of prompts and grammars for each language, and dialogues from the general model of the application (see Section 4.2, page 88). The platform provides some generic libraries, such as prompts and grammars for confirmations, generic data models, etc., but its main potential is the possibility of saving most of the work done in the platform as libraries, including complete dialogues, so that after the creation of a few applications, the designer will have a complete set of libraries that can be reused in future

applications. The platform allows the loading of libraries and provides the functionality to edit their code to adapt them to a new application.

3.2.2 Data Model Assistant (DMA)

This assistant, developed by the partners of the GEMINI project, defines the data structure (or data model) of the service specifying the classes, including inheritance, attributes and types that make up the database. It uses as input the location of libraries and files specified in the ADA. It is possible to define a class with attributes inherited from other classes. The attributes can be of several types: (a) atomic (e.g., strings, Boolean, float, integer, date, time, etc.), (b) full embedded objects or pointers to existing classes, or (c) lists of atomic attributes or complex objects.

A graphical view of a class and its attributes can be seen in Figure 3.2 where, for the bank transfer example, the Transaction class has been defined, which is made up of two object type attributes from the class Account: the first one, *DebitAccount*, to specify the source account and the second one, *CreditAccount*, to specify the destination account. On the other hand, the class Account has several atomic type attributes (balance and account number in the example) and other complex ones (account holder and last transactions list). We can also see the code generated for the Transaction class, together with a reference to its base class (Transaction inherits everything from the base class), called *TransactionDescription* (number 1) and the attributes that will be inherited (number 2). In number 3, the *DebitAccount* attribute is an object reference (ObjRefr) to the class Account, and the same applies to the *CreditAccount* attribute.

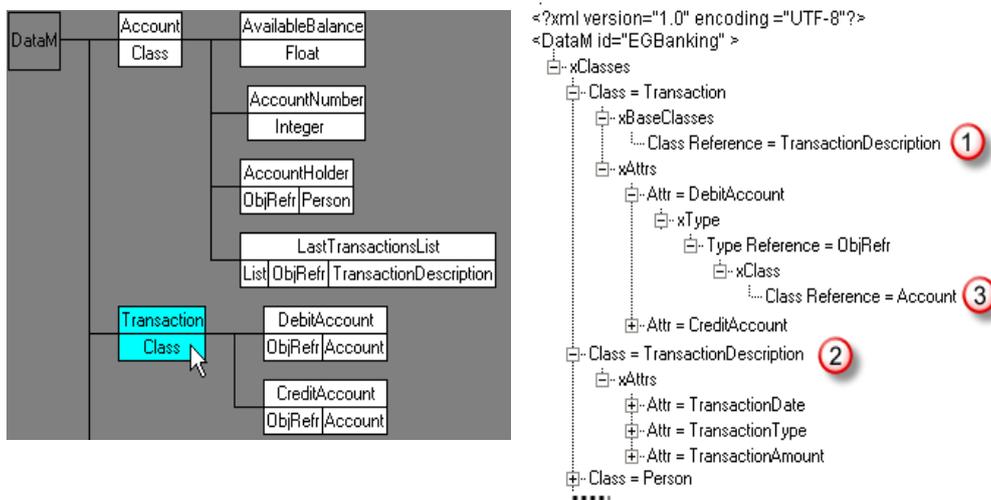


Figure 3.2. Graphical details of a class and its attributes, and code fragment generated for the Transaction class.

Finally, one advantage of defining the data model this way is that the dialogue designer does not need to have the information contained in the database. This could be important if the real database cannot be accessed for security reasons (e.g. a bank database with confidential information about the clients). Besides, the designer can reuse the dialogue model between similar services and modify the contents of the database with no effect in the dialogue structure. However, regarding the acceleration strategies applied to this assistant (see section 4.2, page 88) both cases were considered: with or without access to the database content and depending on the information defined in the ADA the system changes its behaviour accordingly.

3.2.3 Data Connector Modelling Assistant (DCMA)

This assistant allows the definition of the structure of the database access functions that are called from the runtime system. These functions are specified as interface definitions including their input and output parameters (see Figure 3.3). This allows the use of database functions by dialogue designers, without needing to know much about database programming at all. It uses the libraries specified in the ADA and the data model defined in the DMA. This assistant was created by the partners of the project GEMINI but improved in this thesis.

As the runtime platform itself must be independent from backend systems and databases used in an application scenario, the specific implementation (in any suitable programming language: SQL, ORACLE, Informix, etc.) of the access functions was left to database or backend experts, meaning that they will provide the functionality for the database functions, which have been defined by the dialogue experts. As the resulting model is independent from any implementation detail, it is not affected by changes in the system backend as long as the interface remains stable. However, in order to start with new acceleration strategies for this assistant, it currently supports the specification of Microsoft Access databases, accessing to the contents of such databases and proposing automatic SQL queries for the specified functions (see section 4.3, page 91).

The screenshot displays the Data Connector Modelling Assistant (DCMA) interface. The main window is titled "Data Connector Modelling Assistant" and contains a "Data Connector Model" section with "New", "Load", and "Save" buttons. Below this are tabs for "Description", "Procedure List", "Procedure Details", and "Library Procedures". The "Description" tab is active, showing the procedure name "PerformTransactionFromDebitAccountToCreditAccount" and its description "Transfer the specified amount from the debit to the credit account".

The "Argument Vars" section contains a table with the following data:

	Name	Clc	Type	Reference (DataM)	Reference (DB)
1	DebitAccountNumber	Atomic	Integer	Account.AccountNumber	accounts.account_number
2	CreditAccountNumber	Atomic	Integer	Account.Account	
3	TransactionAmount	Atomic	Float	Transaction.Tr	

Below the table are "Add", "Edit", and "Remove" buttons. A secondary dialog box titled "DCMA -- Edit Argument Variable" is open, showing details for the "DebitAccountNumber" variable. It includes fields for "Name", "Reference in DataM", "Reference in DB", "Collection Type", "Type", "Class", and "Description". The "References DataM" checkbox is checked. The "OK" and "Cancel" buttons are at the bottom right of the dialog.

The "Return Value Vars" section contains a table with the following data:

	Name	Clc	Type	C
1	TransactionPerforme	Atomic	Boolean	
2				

Below this table are "Add", "Edit", and "Remove" buttons. The "Implementation" section at the bottom has a "Source File" field, an "Edit" button, and a "Create SQL" button. A "Close" button is at the bottom right of the main window.

Figure 3.3. Form used to define the prototype of a database access function

3.3 Retrieval Layer

In this layer, the service flow is defined at a high level, i.e., in a language and modality independent way, so all it is done using concepts. The first assistant allows the definition of the states and transitions of the dialogue flow, together with the slots to be requested to the user at each state. The next assistant allows the specification of all the actions to be done at each state (e.g. actions to retrieve/show information to the user, to query the DB), as well as actions that control the transitions between states, conditional actions or looping procedures, etc.

3.3.1 State Flow Modelling Assistant (SFMA)

This assistant is very important because it drastically accelerates the design process, especially in the next assistant. As input, it uses the general strategy for the service, the data model, and the database access functions.

In this assistant, the designer has to specify the states that make up the dialogue flow (i.e. only the flow structure is defined, not the conditions that determine the transitions between states, internal actions, nor other more detailed aspects because these are defined in the next assistant in a rule-based manner). In addition, the designer specifies the data (slots) that have to be filled by the user in each state and the transitions between the current state and the following one(s). Besides, it is possible to specify which slots are optional (for over-answering) and which ones can be asked for by using mixed-initiative.

```

<?xml version="1.0" encoding="UTF-8"?>
<SFM id="EGBanking" >
  ...xGlobalDescription = This is the state flow model for the EGBanking application.
  ...All dialog slots are defined at this level, as well as the main dialog flow.
  xStartDialog
    ...Dialog Reference = WelcomeDialog
  xDialogs
    Dialog Name = TransactionDialog Returning = false
    ...xDescription = In this state the debit and credit account identifiers are
    ...required from the user, using a mixed initiative dialog.
    xInputFieldVars
      Variable Name = DebitAccountIdentifier
      ...xDescription = This slot is used to store user input with respect to
      ...the debit account, when an amount of money is transferred.
      xType
        ...Type Reference = String
      xIsOptional
        ...BoolC = false
      xDataMAttr
        ...Attr = Account.AccountIdentifier
      xIsMixedInitiative
        ...BoolC = true
      Variable Name = CreditAccountIdentifier
    xCalls
      Inline Call
        xProcedure
          ...Dialog Reference = GetTransactionAmount
      Inline Call
        xProcedure
          ...Dialog Reference = ClientAuthenticationDialog
      Inline Call
      Inline Call
    Dialog Name = ClientAuthenticationDialog Returning = true
  .....
```

Figure 3.4. GDialogXML code generated by the SFMA.

Figure 3.4 shows the code generated by the SFMA for the example dialogue; it includes information regarding the slots (field *xInputFieldVars*), dialogue transitions (field *xCalls*), and generic information of the application, such as the name of the initial dialogue (*WelcomeDialog*). The figure also shows the definition of the state where the bank transfer data are collected (*TransactionDialog*). In this example, only the account names (*DebitAccountIdentifier* and *CreditAccountIdentifier*) in that state have been selected, and the collection of the amount to be transferred has been left for the next state, called *GetTransactionAmount*. Besides, both slots are collected using mixed-initiative (the tag “*xIsMixedInitiative*” is set to true).

3.3.2 Retrieval Modelling Assistant (RMA)

Designed and developed completely by the author of the thesis, this assistant is the most complex and versatile module in the entire platform, and the one with the highest number of accelerations. Given the large amount of actions that can be carried out in each state, a large programming effort was necessary here, looking for its automation and flexibility. Starting with the main window, it allows several editing and visualization capabilities such as a tree-structured flow diagram where each leaf and branch represents the states and transitions defined in the previous assistant. A colour-coding convention shows whether a dialogue has been edited or not, the dialogue type, etc. In addition, it is possible to access information regarding actions and variables already defined for each leaf (dialogue) in the flow. All the automatically generated dialogues, libraries, and database access functions already defined can also be used and edited. Other available capabilities are the creation/deletion of dialogues, variables and constants, and the visualization of information from previous assistants, the creation of if-then-else structures, selection structures (switch-case), loops inside the dialogue (for, while), assignments between simple and complex (objects) variables, and an assistant for mathematical operations and another one for strings.

The platform provides four basic dialogues types that cover the usual possibilities in programming: based on a loop, based on a sequence of actions (or sub-dialogues), a switch construct based on information input by the user (i.e. menu-based dialogue), or a switch construct based on the value of a variable. Besides, empty dialogues, with no action inside, can be created (used to specify the call to a dialogue that will be defined completely afterwards) so that a top-down design of dialogues can be made; in this case, the dialogue type is selected whenever the designer tries to edit the empty dialogue. Another possibility is dialogue cloning, useful when the dialogue to be defined is very similar to an existing one.

The tool also provides the possibility of manually creating dialogues to obtain information from the user (called DGet), and dialogues to provide information to the user (called DSay).

Figure 3.5 shows the output generated by this assistant. The first section shows the global variables of the application, which store the slots defined for the application that may need to be accessed in all dialogues (for over-answering, as we will see in Section 4.5.4). The actions executed in the two dialogues that form the bank transfer (included in the *xReaction* tag) are also shown: *SFM_TransactionDialog* (number 1) and *SFM_GetTransactionAmount* (number 4). In the first one, there is a call to a sub-dialogue (marked as number 2) that fills in the source and destination accounts data using mixed-initiative; then, there is a call (marked as number 3) to the second dialogue. In the second dialogue there is a call to a sub-dialogue that collects the amount to be transferred (number 5), a call to the database access function

(number 6), using as input parameters the three items already collected, which returns a Boolean variable called *TransactionPerformed*, and a call to the next two dialogues specified in the SFMA (numbers 7 and 8).

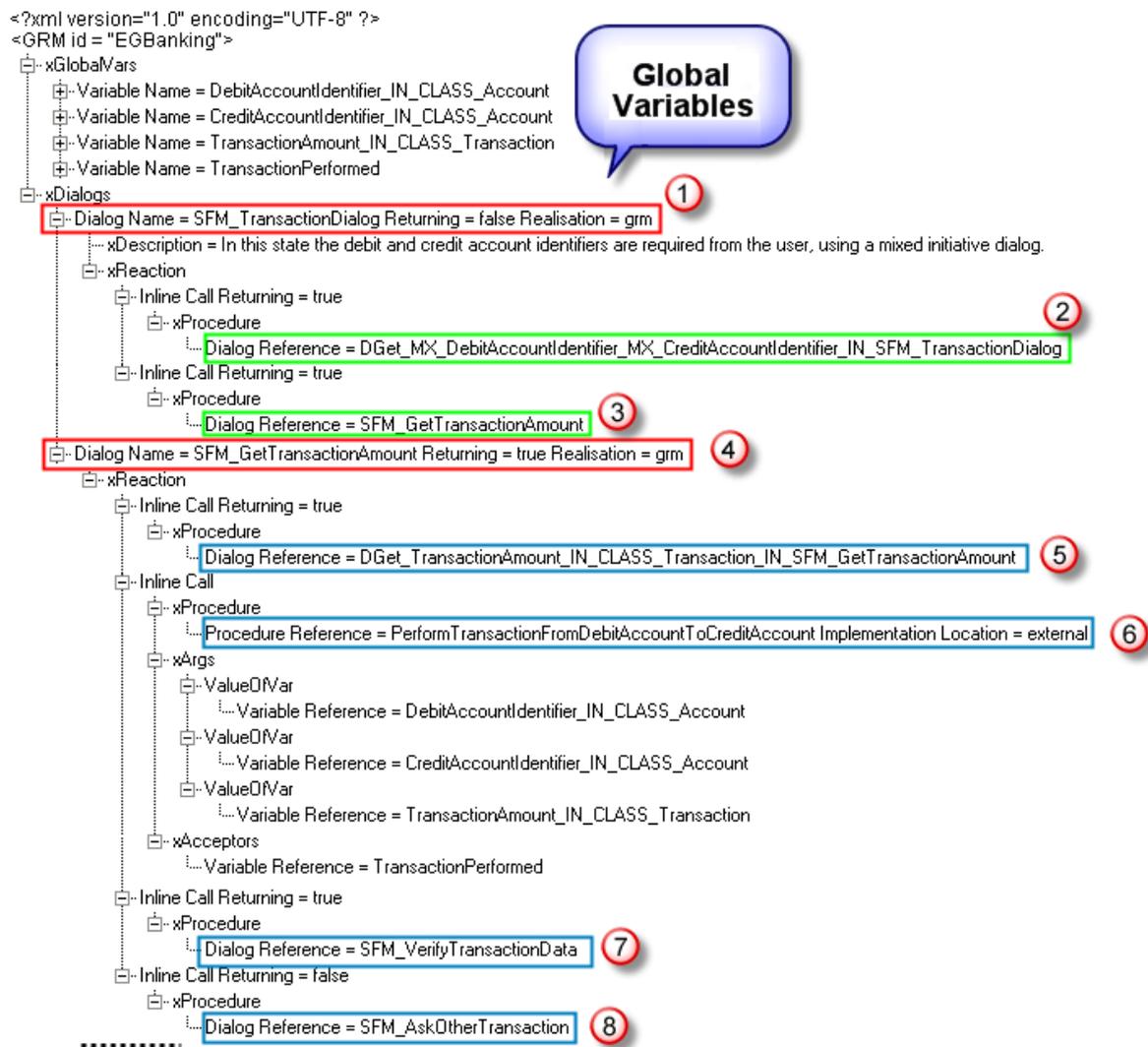


Figure 3.5. Code generated by the RMA for the bank transfer example.

3.4 Dialogues Layer

In this layer, the dialogue is completed with all modality and language dependent aspects. It has four main assistants that are dedicated to the following tasks:

- To define the user levels and their settings (UMA, see Section 3.4.1),
- To complete the modality dependent aspects of dialogue design (MERA-Speech, see Section 3.4.2),
- To complete the language dependent aspects and the input and output concepts for each modality (MEA, see Section 3.4.3) and finally,

- To unify all the information and generate the execution scripts according to the modality (see Section 3.4.5).

Finally, section 3.4.6 briefly outlines some additional assistants incorporated to the platform for specific tasks related, mainly, with the speech modality.

3.4.1 User Modelling Assistant (UMA)

This assistant, created by the partners of the GEMINI project, allows the specification of different user levels and settings for each dialogue in the application in order to provide a more personalized attention to the final user. It uses as input the default confidence and error values defined in the ADA, and all DGet dialogues defined in the RMA.

To start with, all the values are specified first for the defined user levels, but later they can be customized for each specific dialogue state, so that all settings can be user-level dependent and dialogue state dependent. This way, the designer may impose, for example, a stricter confirmation for some critical data such as the amount in a banking transaction.

The designer can specify different settings as the possibility of barge-in for a particular user level, the maximum number of retries if there is an error (considering several error types), the maximum timeouts for several events, etc. Besides, the confidence levels that should be used in recognition for each user level are specified as they determine the confirmation type that should be used (see Section 4.6.2, page 113): no confirmation (confidence between the specified value and 1.0), implicit (confidence between the specified value and the value for ‘no confirmation’), explicit (between the specified value and the value for ‘implicit’) and repeat (between 0 and the value for ‘explicit’).

The decision as to the current user level is made by a runtime component that is called after each interaction in the script generated by the platform and that sets the common internal variable that is used in the final script. This way, the platform is independent of the user modelling technology that is used.

3.4.2 Modality Extension Retrieval Assistant for Speech (MERA-Speech)

This assistant adds special subdialogues that complement the dialogues already defined for the application in the RMA. This way, the designer can include complex dialogues to deal with modality specific problems. This assistant was completely created and developed by the author of the thesis during the GEMINI project, and it is the second assistant with the higher number of accelerations in the platform.

In this thesis, the research has been focused on providing semiautomatic solutions for two basic problems that are specific to the speech modality: the presentation of object lists in several steps (applied to DSay dialogues concerning a list) and confirmation handling, i.e. how to handle recognition errors in dialogues that obtain information from the user (applied to DGet dialogues). The input is the database model specified in the DMA and the dialogues defined in the RMA (especially those marked as DGet and DSay for lists). The details concerning the accelerations for this assistant are outlined in section 4.6.

3.4.3 Modality and Language Extension Assistant (MEA)

Developed by the partners of the GEMINI project, in this assistant the language dependent aspects of an application (input and output prompts/concepts) are specified for each modality. For the speech modality, the extensions consist of links to grammar and prompt concepts, while for the Web modality the extensions consist of links to input and output concepts. In both cases, the extensions are language independent. In addition, language dependent information, specifically wording for both speech prompts and Web output concepts, is also set here. All this information is saved in different files for each language and modality, whose content and organization is explained at the end of this section. As input, it uses all dialogues defined in the RMA and MERA-Speech, together with the specification of the user levels from the UMA. The assistant detects the input/output dialogues (DGets and DSays) defined in previous assistants and asks the designer to define prompts and recognition grammars for them. In addition, the assistant lets the designer define the help prompts for high-level dialogues that are not classified as input/output.

For the speech modality, several prompts for each input dialogue have to be defined: the default one, for the different user levels and for all possible recognition errors. In all dialogues, the input/output parameters and the global variables can be used as part of the prompt. To speed up the process of typing all these prompts, the assistant offers two possibilities: reuse prompts already available for the current application or reuse prompts generated in previous applications and saved as libraries. Prompts are set using three alternatives: text-to-speech (TTS) prompts, pre-recorded audio files, or generated by a Natural Language Generation (NLG) module in the runtime system.

In case of TTS prompts, the SSML⁴⁶ markup language can be optionally used. The tags considered for the runtime system were as follows:

1. “emphasis” (to emphasize specific fragments), with the following values for the “level” attribute: “strong”, “moderate”, “none”, and “reduced”,
2. “break” (a break of a specific duration in ms), with the “time” attribute,
3. “prosody”, with “pitch”, “rate” and “volume” attributes. To specify them, in the platform we have used a relative value as a positive or negative percentage, e.g., “+10%”.

Once the prompts for the main language have been specified, the designer has to specify them for the additional languages. This process is accelerated by using the main language prompt as a template to edit the string parts of a prompt (see section 4.7.1.1, page 116). These prompts can be specified either at once for one language and for all dialogues, or for each dialogue for all additional languages. Figure 3.6 shows an example of the creation of a TTS prompt for the dialogue *DGet_ConfirmTransaction* and for the default user (number 1). The assistant allows the designer to select the arguments of the dialogue to be inserted into the prompt (number 2), as well as to use SSML tags (number 3). The final prompt is displayed to the designer in 4. It is also possible to include a link to an audio file in order to allow hybrid prompts (number 5). Besides, the assistant allows the designer to use audio prompts instead TTS (number 6) or to use a natural language generator module called during the real time system (number 7).

⁴⁶ <http://www.w3.org/TR/speech-synthesis/>

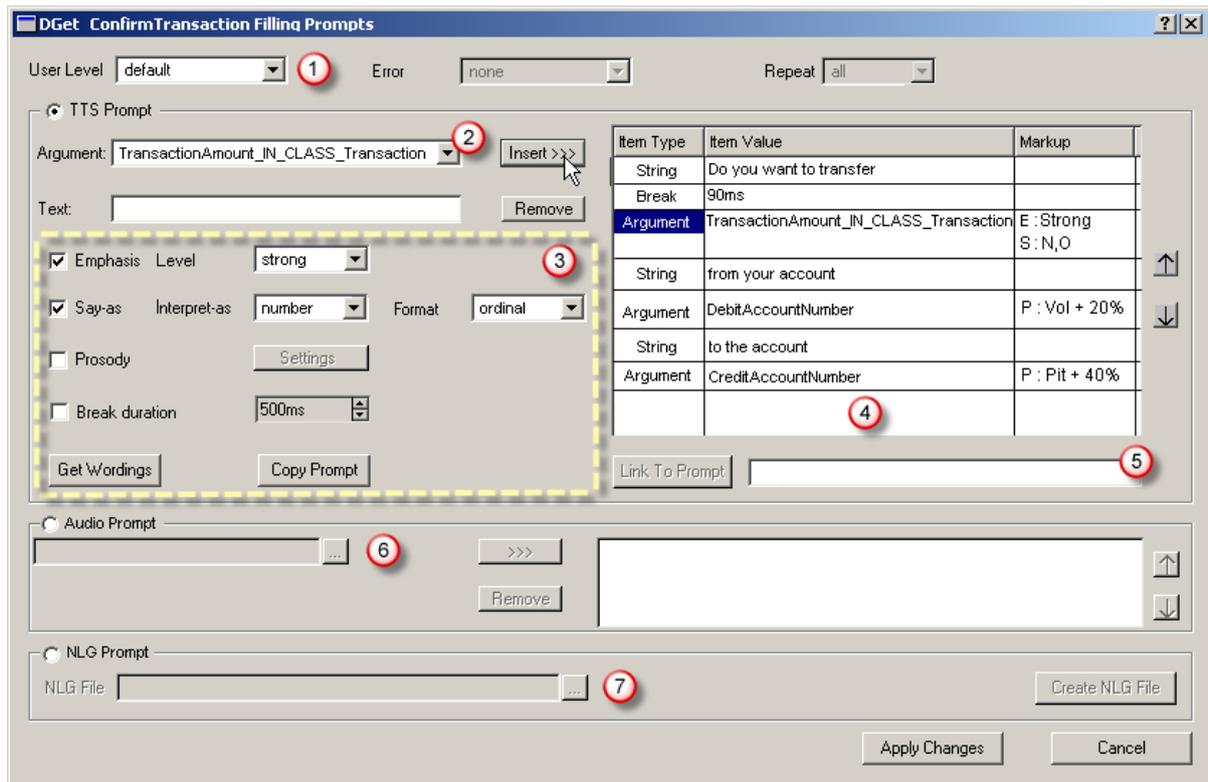


Figure 3.6. Example of the definition of a TTS prompt using SSML tags.

For the Web modality, the procedure is somewhat different because of different concepts for user interaction. On the one hand, each output concept corresponds to some XHTML markup code, optionally parameterized. On the other hand, each input concept corresponds to a Web form control like a text input field, a text area, a select or a choice box, etc. In addition, a set of attributes can be defined for each component: text elements for a label, a hint, an alert, or an error message can be set and the rendering behaviour of the control can also be defined.

As output, the assistant generates four different files for each modality. The first file contains information regarding every dialogue in the application and references to the input/output concepts used in each one. Figure 3.7 shows the code for this file for the speech modality and for the dialogue that collects the amount to be transferred in the bank example. The figure highlights the use of the tag `Realisation` because it tells the linker that this code is an extension of a dialogue already defined in the RMA. Besides, the tag `xPresentation` holds the information related to the system prompt concepts (marked as number 1); the tag `xFilling` gives the information related to the behaviour of the recognizer, i.e., the prompts used to inform the user of an unrecognized utterance (number 3) and no input detected (number 4), together with the grammar to be used in the recognition (number 2).

As mentioned previously, multilinguality is achieved using concepts, so all definitions here for prompts and grammars refer to concepts (PC suffix for prompts and GC for grammars). These references are solved in auxiliary files that are described below. Finally, we should mention that for the Web modality the same tags are used (*xFilling* and *xPresentation*) but, instead of prompts and grammars, input (using the tag *InputControlCall*) and output (tag *OutputControlCall*) concept references are used.

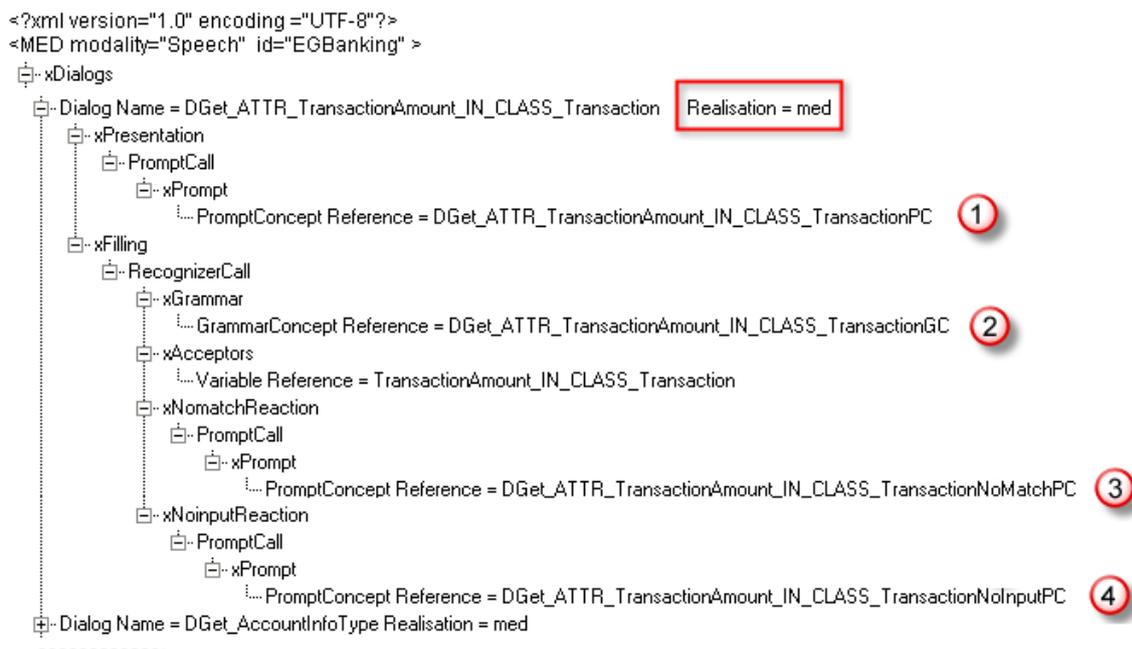


Figure 3.7. GDialogXML code generated by the MEA for the speech modality.

The second file, for the speech modality case, is called ‘grammar concept file’, and it contains the association between ‘grammar concept’ (GC) and the filename of the grammar(s) that will be used in the real-time system, so it is language independent. As we have different grammars for each language, to achieve multilinguality the grammars in all languages have the same name but are in separate directories; the directory name is the language code, so the real-time application just concatenates the language code with the filename to retrieve the correct grammar. The assistant allows the specification of different grammar formats such as JSFG, SRGS and grXML. Besides, during this step, a different grammar for the speech recognizer can be also specified. In this way, the assistant allows the specification of stochastic grammars (i.e. n-gram based language models) that can be used by the ASR, and the specification of rule-based grammars for the Natural Language Understanding module (NLU). The main motivation behind this behaviour was that at present stochastic grammars are not fully supported in JSGF or SRGS grammars. For that reason, our NLU and ASR runtime modules were properly configured to accept both source of information.

The third file is the ‘prompt concept file’ and it contains, for each input/output dialogue, the association between ‘prompt concept’ (PC) and the name of the text concept or the audio file that has to be used for it (not the prompts for each language). The fourth file, called ‘text concept file’, holds the actual prompts (the real texts in SSML format as we mentioned above) that correspond to the text concepts defined in the ‘prompt concept file’. Therefore, this file is language dependent and it is repeated for every language and, again, it is saved with the same name in separate directories. This separation might seem complicated, but it is the only way to ensure multilinguality and the flexibility to handle audio files, prompts, etc., in the same application.

For the Web modality, similar files are generated, but now, instead of the ‘grammar concept file’ and ‘prompt concept file’, the files are called ‘input concept file’ and ‘output concept file’, again language independent, which describe the appearance of each input or output item (radio button, submit button, secret text, labels, combo boxes, lists, etc.). They

also include a reference to the text concepts that they use, which are also specified in a ‘text concept file’ just like the speech modality.

3.4.4 Dialogue Model Linker (DML)

This module was developed by the partners of the GEMINI project and it is the responsible of generating one file for each selected modality where all the information from previous assistants is automatically linked together: dialogues, actions, input/output concepts, prompts and grammars, etc.

The final dialogue model is a combination of the files produced by the RMA, the MERA-Speech and the MEA. All these models are linked together by filling different sections of GDialogXML dialogue units; see [Hamerich et al, 2003] for further information.

3.4.5 Script Generators

In this section, the modules that convert the dialogues coded in GDialogXML syntax into the execution scripts needed for each modality (VoiceXML and xHTML) are described. These modules were developed by the partners of the GEMINI project, but some modifications were introduced by the author of the thesis in the VoiceXML generator in order to adapt it to the runtime system described in section 3.5.3. To carry out the process, they solve the problems and limitations of each standard and manage those issues regarding the handling of multilinguality, database access, the preparation of prompts or Web text, and the handling of concepts in the language-independent specification of the dialogue.

One important issue is how the system handles in real-time a prompt that includes information returned by a database query. To solve this, the script generators include one global variable that codes the language of the service with an identifier in ISO639-1 format (language code) followed by an identifier ISO3166 (country code). This variable is set by the language identification module at the beginning of the session and it is always passed as an argument to all database access functions specified in the DCMA, where it is concatenated with the field name that is going to be retrieved. Obviously, the database (there is a single database for all languages) should contain the same information for each language used in the service using fields with the same base name but different codes as suffix, e.g., `info_text_en_UK` for the field with the information in English, `info_text_es_ES` with the information in Spanish, etc. Once the query is made, the right variables are filled in and the information is provided to the user in the correct language.

3.4.5.1 VoiceXML generator and connection with the runtime platform

Using the file created by the linker (DML) in the previous step for the speech modality, this module generates a file in VoiceXML format for each language used in the service.

The script generator for VoiceXML has to overcome the limitations imposed by this language in its version 2.0. The main limitation is probably that VoiceXML does not allow returning calls (subroutines), which are needed to solve the problem of presenting lists of objects (see Section 4.6.1, page 110), as ordinary statements (returning calls are only allowed at certain positions). Therefore, all complex statements and value expressions have to be “flattened” into simpler operations and into calls to intermediate dialogues that allow jumps to other states in the dialogue flow.

Besides, VoiceXML does not allow input fields to be global variables; however, we used global variables for over-answering so that they can be filled in previous states and do not lose their contents when jumping to other states. Therefore, a synchronizing strategy had to be implemented to map global variables to local input fields and vice versa. To handle over-answering it is also necessary that, if a slot is optional or it is already filled, the recognition process may be omitted. In VoiceXML all the slots in a form must be filled, so we introduced additional intermediate variables and conditional blocks to find out whether the slots associated to over-answering are filled or not. Another issue is how to clear the contents of a slot in a form and jump back to previous dialogues or states, which is another behaviour needed for list handling, as the VoiceXML manager would automatically repeat the filling process for that slot and that is not the desired behaviour. To solve this, the VoiceXML generator creates intermediate variables so that, when the slot has been cleared intentionally, the filling process is not repeated.

When compared to other programming languages, VoiceXML lacks common constructs for program logic, like loops (e.g. while and for). In general, the proposed solution is to use an ECMA script, but this is useless when operations across fields or dialogues are needed. In our case, we implemented a complex structure of blocks and calls that provides these functionalities although it makes difficult to debug the final script.

GDialogXML supports the idea of connecting services during runtime, e.g., services providing access to databases, services generating prompts on the fly, and so on. The VoiceXML generator implements these calls as HTTP requests via a CGI script. This CGI script works as a data bridge and contacts the actual services. It is needed because it has to produce VoiceXML code, since this is the only way to integrate dynamic data (result values) into the dialogue flow. By using the bridge, the services are freed from the burden of producing VoiceXML themselves. Although this solution works fine when transmitting data to a database (i.e. updates procedures), several problems arise when the results from the database have to be returned into the dialogue flow; besides, the script could be insufficient in some applications, delaying and making difficult the design. In our case, we solved this problem providing a general-purpose built-in script that dynamically generates the VoiceXML code by using assignments of string constants to variables.

In order to generate prompts on the fly, the platform uses language-dependent JSGF grammars, in which the correspondence between prompt and concept is specified, and the recognizer would return in real-time the concept specified in the grammar instead of the prompt. To increase performance, the VoiceXML generator uses a reference resolution strategy for result values of the runtime services. This means that if the result of a service request is a reference to a complex data object (e.g., a person), only the reference (consisting of the identifying attributes) is transmitted. At the time when details of the object are needed, the complete data structure of attribute values is transmitted. This is particularly important, when the result of a service request is a large list of references to complex data objects, which happens a lot when navigating through large information databases.

Another issue we found was that in order to simplify the reuse of the same VoiceXML script for different languages (localisation), it would be useful to establish the external representation of prompts by introducing prompt concepts that, in the real time system, can be translated according to a global variable and looking up into the language dependent prompt files. In the current platform, we create a different file for each language. On the other hand, in VoiceXML there is no specification regarding how to identify the different active recognizers. This could be interesting to accelerate the service, since several different

recognizers can operate in parallel (i.e. one for continuous speech, other for isolated speech, etc.). In the current platform, we solved this problem using the <property> tag.

Finally, the VoiceXML generator automatically creates global variables in the final script, where the dynamic runtime values returned by the corresponding modules, are kept to handle several aspects of the runtime system: the user level, the speaker identifier, the confidence value from the last recognition, the current language, etc. The user level variable, for example, is needed for switching prompts depending on the user level, which is set by calling the User-Level-Detector runtime service.

In [Hamerich et al, 2003] and [Cordoba et al, 2004b] other limitations of VoiceXML are described, together with some recommendations to improve the standard. It is important to mention that they were submitted to the W3C for their consideration in future releases of VoiceXML standard.

3.4.5.2 Web script generator

Using the file created by the linker (DML) in the previous step for the Web modality, this module generates a file in xHTML format for each language used in the service.

Unlike the voice modality, for Web the distinction between the flow control, the data and the presentation (e.g., buttons, images, frames) is not too clear. Nowadays, there are many integrated development environments for the presentation part (Web editors), whereas for the control and data access they have to be specified using script languages (e.g., perl, php, python) or usual programming languages (e.g., Java, .Net). In other cases, the overall flow control is supported by frameworks (Jakarta Struts...), but in general there is no widespread language, so the task of integrating all of them is difficult. Therefore, the objective of this assistant is not to compete with widespread Web editors but to provide a complementary support to try to facilitate the separation between the modelling of the flow control, the data and the presentation. To this end, the assistant automatically transforms the GDialogXML models related to input and output concepts into xHTML files with embedded xForms elements. This way, the generated files can be used as templates for Web designers who can add additional design elements (xHTML tags, images, styles, etc.), while the dialogue flow is preserved separately. A runtime interpreter for GDialogXML may execute the dialogue model “as is” in the Web server environment to control the dialogue flow and take care of the database transactions.

Thanks to this separation, the final script is platform independent and easily adaptable to multiple display devices (browsers, PDA, public terminals, etc.). Although there are some limitations with xForms as not all browsers support them, the use of plug-ins or rendering programs in the server provides that support. Besides, the use of xHTML tags could favour the integration of the two modalities, in a future development, so that they can work at the same time using the standard X + V.

Although this module is an important part of the design platform, because it provides one of the modality capabilities, in this thesis it will not be described in detail since the author did not contribute to this module.

3.4.6 Auxiliary Assistants

In addition to the assistants described above, there are some other assistants that complement the platform and for the voice modality. All these assistants were developed by

the partners of the GEMINI project but some accelerations and minor modifications were introduced by the author of the thesis.

3.4.6.1 Vocabulary builder

The first one is the vocabulary builder (VB) which prepares the vocabularies that will be used by the recognizer. Thus, this component gets input from the language model resources and produces the lexicon. The lexicon contains the phonetic transcription of each word and in most cases the phonetic alternatives. There are equivalent dictionaries for each of the different languages allowed in the platform.

3.4.6.2 Language modelling toolkit

A second assistant is the Language Modelling Toolkit (LMT) that allows the designer to specify the grammar files that will be used in the runtime system to assign a meaning representation from the different user answers to the system questions. The assistant allows the creation and edition of grammars in JSGF format both for recognition and for prompt generation using the Natural Language Generation (NLG) module. More details can be found in section 4.7.1.2 and in [Georgila et al, 2004].

3.4.6.3 Diagen

The third assistant is called Diagen. The assistant allows the manual creation and fine tuning edition of the different GDialogXML models and libraries generated by the assistants of the AGP. Originally created during the development of the GEMINI project, it was later extended and improved by [Hamerich, 2008] to allow the creation of new speech dialogues, and to adapt it to new user environments and updates of the GDialogXML specification. In addition, the author of this thesis also contributed to the development and extension of this assistant in order to allow the creation and edition of some of the models generated by the platform. In detail, the assistant was extended thanks to the possibility of creating models for the DMA, DCMA, SFMA, and MERA-Speech assistants, as well as some minor changes to the process of generating models for the RMA assistant.

Several accelerations were included in this assistant in order to simplify model edition or to allow the creation of models from scratch. The main acceleration is the possibility of creating any section of the GDialogXML specification with minimum effort. Instead of typing all the tags nodes and children, the assistant uses a set of pop-up windows that are sequentially displayed according to the information that the designer needs to specify.

Figure 3.8 shows an example of the process for creating, from scratch, a dialogue for the retrieval model assistant. According to this figure, in (number 1) the designer specifies that the new dialogue will require the definition of at least one global variable and dialogue. After accepting, a new pop-up window is displayed (number 2) allowing the designer to specify the information for the global variable to add. In order to add new variables, the designer only needs to check the corresponding checkbox (i.e. Enter another variable) and a similar pop-up window will be displayed after accepting the current one. The next step (number 3) is the definition of the dialogue state. In this case, the designer first defines some attributes of the new dialogue such as name, returning flag (i.e. a Boolean property that indicate if after a transition to another dialogue state the system will require to unconditionally return to the calling dialogue), etc. Besides, the new pop-up window allows the specification of different properties for a dialogue such as variables, calls, help messages, reactions, etc. In this case, the designer checks two elements to be defined.

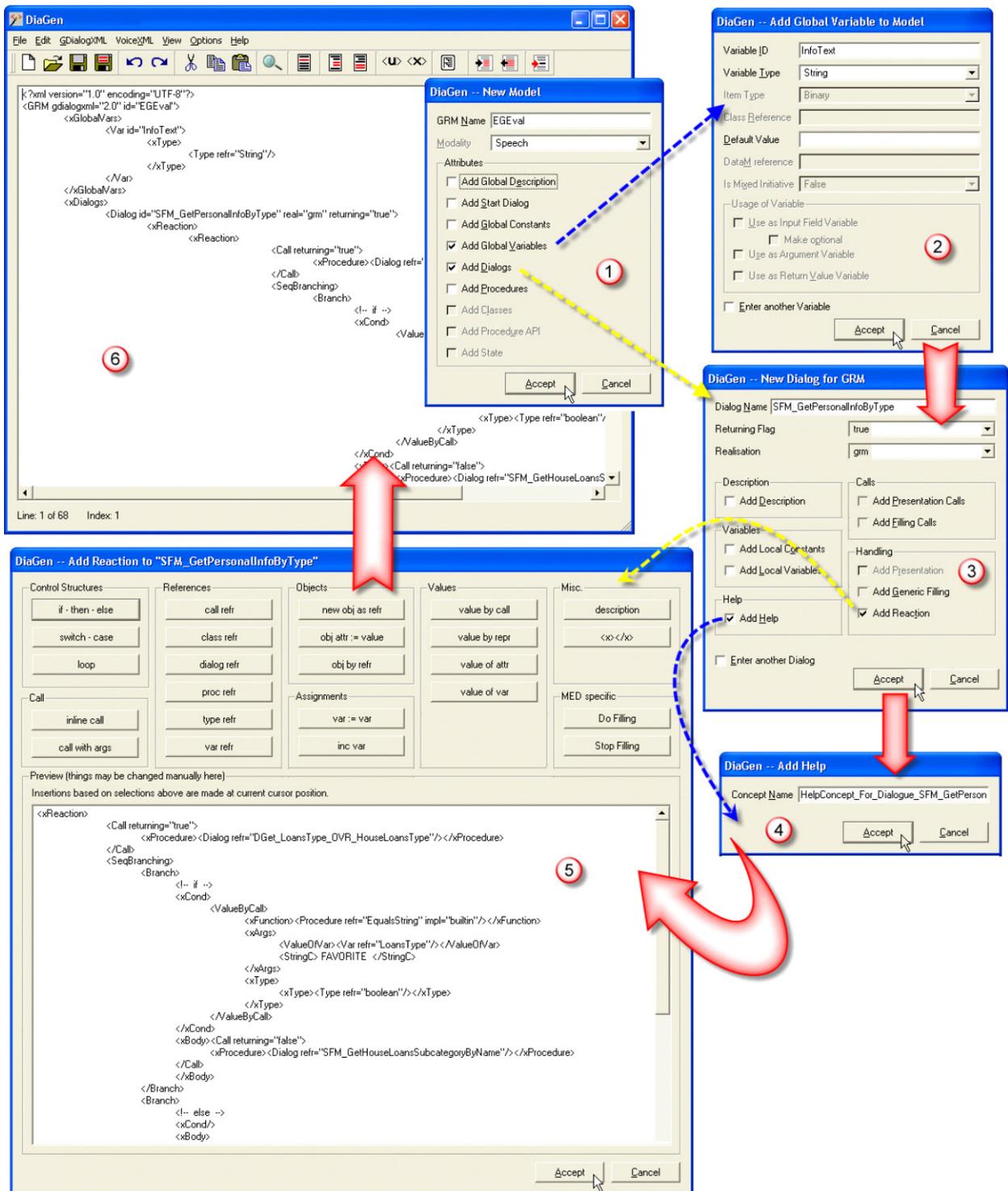


Figure 3.8. Process for creating a dialogue in GDialogXML using the Diagen assistant

In the example, the designer selects the specification of the help concept, defined using the pop-up window marked as number 4, and the reaction procedure, marked with number 5. A brief glance of number 5 gives a glimpse of the complexity of the actions that it is possible to define using this pop-up window. For instance, it is possible to define conditional actions, loops, calls to other dialogues, variable assignments by reference or value, repetitive processes, etc. After clicking on any of the buttons of this pop-up window, the system

automatically pastes the text of a default template that the designer carefully needs to complete.

Considering that this is the only acceleration provided by the platform at this very critical point of the design, most negative commentaries during the objective evaluation were focused on this step, together with the total time required to complete the dialogue. Finally, after accepting, the system returns to the main window of the assistant (number 6). In this case, the GDialogXML code generated from all the information collected and defined using the previous windows is pasted into the workspace.

Although the main purpose of this assistant is to allow the fine-tuning of the models generated with the AGP and its assistants, it was used during the objective evaluation of the platform. The main reason was that this assistant includes some auto-complete templates and advanced editing functionalities that were adequate to provide comparison results between using this assistant or the accelerated assistants included in the platform when creating the dialogue service. For more details regarding the evaluation, please refer to section 5.2.

3.5 Runtime System

Although all the voice scripts generated by the AGP can be executed and tested using any VoiceXML interpreter that includes a basic speech recognizer or synthesizer, in our case, with the objective of having more control over the final system, we decided to implement the runtime system using proprietary modules and open source code. In this section, we will describe in detail the main components of the runtime platform.

3.5.1 Speech Recognizer and Synthesizer

Regarding the speech recognition system, our group has developed a recognizer based on continuous HMM with multiple Gaussians per state [Cordoba et al, 2001] trained using the SpeechDat database in Spanish with more than 4000 speakers. Moreover, it is possible to adapt the acoustic models using MLLR (Maximum Likelihood Linear Regression) or MAP (Maximum A- Posteriori) techniques through the functionalities provided by Hidden Markov Model Toolkit (HTK)⁴⁷. In this way, it is possible to use new records to improve the models or to adjust them to a particular speaker or the acoustic environment of the final service [Cordoba et al, 2006a].

Boris is the Text-To-Speech synthesizer developed in our group [Pardo et al, 1995]. It is a concatenative diphone synthesizer where the fundamental frequency and duration of the units are calculated automatically using several features from the input text and an Artificial Neural Network [Cordoba et al, 2002][Montero et al, 2003]. In addition, the synthesizer is able to process a subset [Cordoba et al, 2004b] of the SSML standard that allows the designer to modify some characteristics of the voice.

⁴⁷ <http://htk.eng.cam.ac.uk/>

3.5.2 Animated Agent Used by the Sign Language Translation System

As it was mentioned in chapter 1, one of the main goals of the platform presented in this thesis is to provide the final service in multiple modalities and for different types of final users with a minimum effort for the designer. For that reason, we worked in the creation of an automatic procedure for helping designers to quickly translate the prompts of the spoken dialogue system into a representation in the sign language that it is played afterwards by a 3D avatar in order to let deaf people use the service.

The process of converting the speech or textual representation into the final visual representation can be divided into two parts as presented in Figure 3.9. The first part is the automatic translation of the prompts into the corresponding visual messages in the sign language. The second part is the process of creating and storing the signs using the appropriate format required by the avatar that plays the signs. In this section, we will focus on the second part, leaving the first part, i.e. the full description of the machine translation system and the work done to improve the translations, for sections 2.4 and 6.2.

According to [Timmermans, 2005], nowadays there are an estimated of one million deaf people just in the 26 European states. For many of them, sign languages are their first language for communication. In spite of what many people thinks, currently there is no a single or universal sign language. All of them differ from each other in a similar way as spoken languages, having its own grammar, syntax, lexicon, rules, etc.

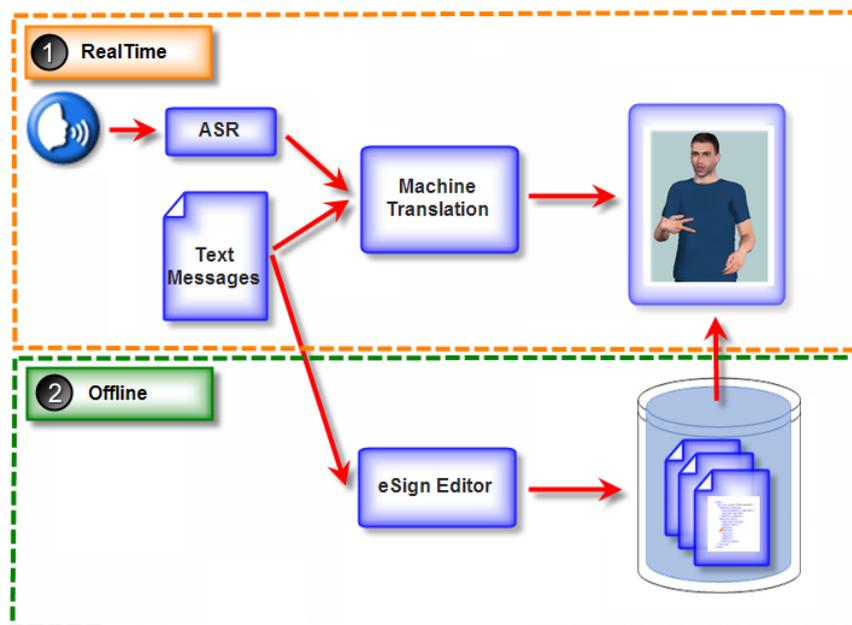


Figure 3.9. Offline and Online process for creating and using sign language prompts

At the very beginning of this project, we thought that a simple interface displaying the text of the prompt or the recognized sentence by the ASR was enough to allow deaf people to use the service. However, as we were going further we realized that most deaf people have a low ability to understand written text because they are used to communicate using the sign language grammar that, as stated above, is different from the grammar used by hearing people, or because many of them did not have access to academic studies. For these reasons, it is very important for them to have access to information in their mother language. In order

to provide this kind of visual information, we integrated, in the runtime system, a 3D avatar called VGuido⁴⁸. This avatar was the result of the European project eSIGN (Essential Sign Language Information on Government Networks). It constitutes one of the most important efforts in developing tools for the automatic generation of Sign Language contents. Although the tools developed in eSIGN were oriented to translate Web content into Sign Language, and as a result it is being used right now on local Government websites in Germany, the Netherlands and United Kingdom, the 3D avatar allows a quick generation and playing of any content in the corresponding Sign Language.

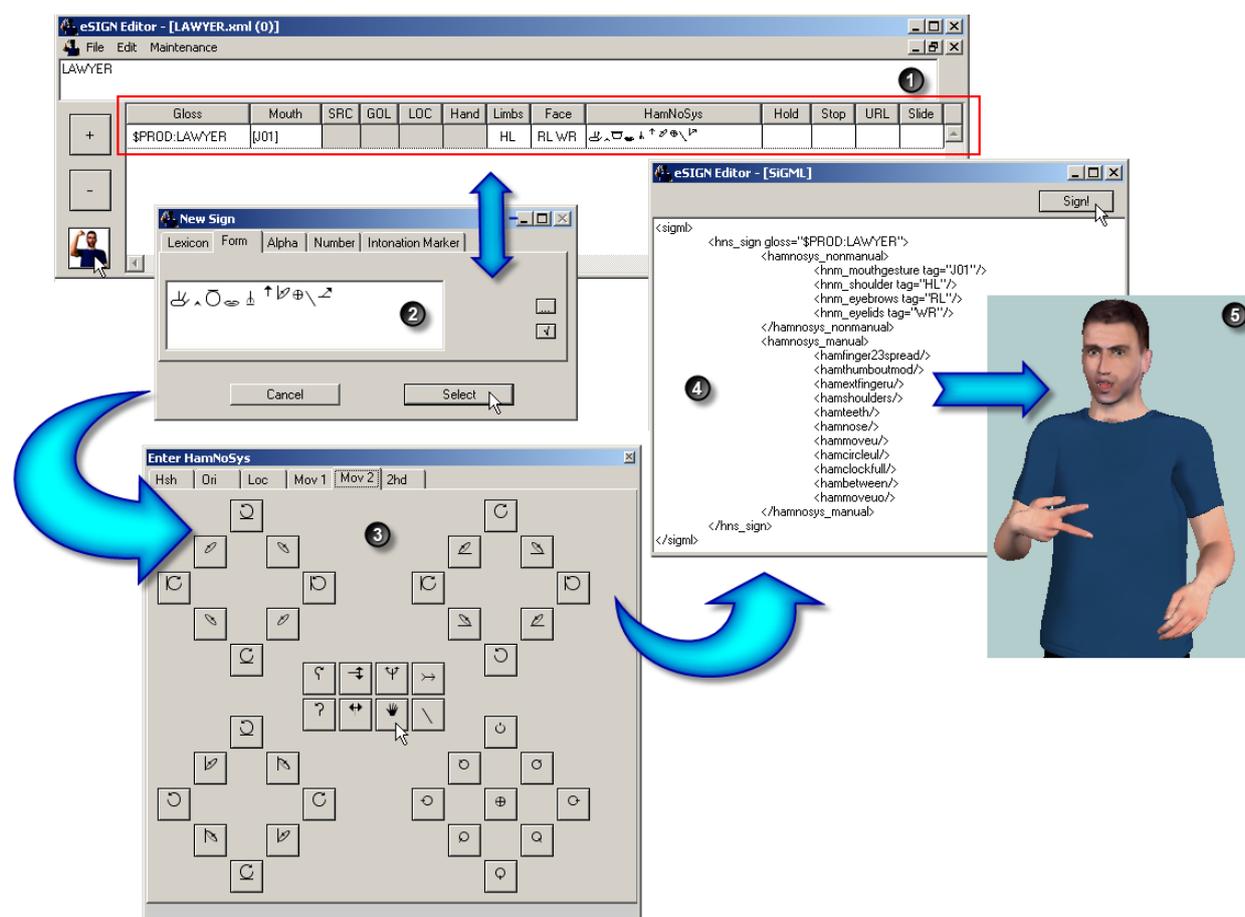


Figure 3.10. Example of process to design and play a sign with VGuido.

For the design stage, the toolkit has a graphical editor that allows the creation of any kind of sign through dynamic configurations of different elements of the avatar body (i.e. head, left and right hands, eyes, trunk, etc.). The process is depicted in Figure 3.10. The first step is to define the gloss (i.e. the written word representing semantic information of the sign), in the example it is the capital word: LAWYER. The second step is to define the gloss using the HamNoSys glyphs representation. HamNoSys glyphs describe the hand-shape, hand configuration, location, and movement that the avatar has to play. The editor allows the creation of all the possible symbols covered by the HamNoSys standard using the pop-up

⁴⁸ http://www.sign-lang.uni-hamburg.de/eSIGN/AnnualReport2003/VGuido_Internet.html

window in step 3. Although not covered by HamNoSys notation, other information such as mouth, eyes, shoulders, speed, size of the movement, etc. can also be included to define the gloss through other pop-up windows. In the figure, we have included the J01, HL, RL and WR symbols (step 1) to represent a specific movement of lips, face and head for instance. The next step, number 4, is to create a XML representation of the avatar movements for the corresponding gloss. The XML is written using a proprietary language called SiGML (Signing Gesture Markup Language) that are used later by the runtime system to concatenate the sentence to play to the user (i.e. the signing system constructs human-like motion from these scripted descriptions of signing motions). These signing motions belong to “Gestural-SiGML”, a subset of the full SiGML notation, which is based on the HamNoSys notation for Sign Language transcription [Prillwitz et al, 1989]. Finally, in step 5, the designer can play the sign with the avatar to verify the motion of the sign.

At runtime, the system uses the translated sentence, consisting of a sequence of glosses, and picks up the corresponding predefined file for each gloss containing the sequence (script) of the animation coded written in SiGML. Then, using the script, the avatar builds and plays a human-like movement that represents the sign. The animation consists of a sequence of temporal frames that define a static position for the avatar at each moment.

An important advantage of this agent is that it is possible to store these scripts files (one for every gloss in the vocabulary) in order to sequentially concatenate them to form a sentence. In this way, in order to play a Sign Language sequence from a defined prompt in the system two conditions have to be fulfilled: a) the script for the sign must exist; b) the prompt has to be translated from its written representation into a sequence of signs following the grammar structure of the Sign Language.

The first condition cannot be easily made automatic since signs change from country to country (even from city to city in the same country) and because the creation of every sign is a time consuming task. However, after some time, it is possible to have a big number of signs stored that simplify future developments.

Regarding the second condition, since, in general, dialogue designers do not necessarily know Sign Language, it is hard for them to translate a defined prompt into a sign sequence, or it could be too expensive to hire an expert to do this work. Then, an automatic solution can be proposed. In this thesis, we propose an automatic solution based on machine translation techniques (see section 2.4, page 41). In our platform, we have used free available software for training the translation models and for translating sentences between both languages. In our case, we used Giza++ [Och and Ney, 2003] and Pharaoh [Koehn, 2004] toolkits. These open source programs provide all the required tools to train and run a phrase-based translation system [Koehn et al, 2003]. A full description of the developed system in our group can be found in [San-Segundo et al, 2006] and [San-Segundo et al, 2008]. In section 6.2, we will show how the quality of the translations can be improved using an adapted language model with online counts.

3.5.3 Distributed Platform and VoiceXML Interpreter (OpenVXI)

Finally, another important component in order to run the VoiceXML script generated by the AGP is the interpreter or browser that executes the script and performs the connections with the other modules (recognizer, synthesizer, database access, telephonic interface, etc.).

Currently, several commercial and free applications can be downloaded from internet and used as VoiceXML browser. For instance, we can mention VXI VoiceXML browser⁴⁹ for Asterisk PBX, VoiceXML Gateway Software⁵⁰, Hewlett-Packard OpenCall Media platform⁵¹, JVoiceXML⁵², PublicVoiceXML⁵³, among others. The selected interpreter for this thesis was the open source library OpenVXI⁵⁴ [Eberman et al, 2002] supported by Vocalocity Inc. It consists of a collection of configurable components that the developers can use, modify, or completely substitute with their own code where appropriate, supporting also proprietary grammar formats, URI types, and VoiceXML objects. The platform includes basic telephony functionalities, an XML parser to process VoiceXML and JavaScript files, processing of user input, a complete implementation of the Form Interpretation Algorithm (FIA) following the specification 2.0, basic debugging functionalities, simulated speech recognition, and generation of prompts and text-to-speech. Since the source files are available, there were not restrictions to adapt, mainly, the TTS and ASR interfaces to our proprietary modules and platform. In [Cordoba et al, 2004a], [Cordoba et al, 2004b] and [Hamerich et al, 2003], the process to adapt the browser to the characteristics of our runtime platform are described in detail.

Another task that was undertaken during the realisation of this thesis was the integration of the runtime system into the distributed platform presented in Figure 3.11. This platform was the result of the DIHANA project⁵⁵ [Hurtado et al, 2005]. The platform is made up of seven modules that execute the different processes in a dialogue system. The architecture also defines the different messages the modules can use to share information among them.

Each module, when initialized, opens a specific port that allows the communication of that module with the Hub and, in through it, with the other modules. The architecture allows the communication between each module with all the others using sockets through the Hub (represented by solid arrows in Figure 3.11). However, the common communication flow is represented by dashed arrows.

The first module in the platform is the Dialogue Manager (DM). It is the most important component in the dialogue system. It controls the dialogue flow according to the answers and messages to be retrieved/presented from/to the final user, and with the interaction with external knowledge sources (i.e. database, language identification module, user models, etc.). Another important task performed by the DM is to provide solutions for miscommunications during the dialogue, such as inaccurate meaning representation of the user's input and the output of the language-understanding component, or discrepancies between the information the final user wants and the available in the database. In this case, the DM has to manage different clarification and verification strategies according to the problem. In the current runtime system, the DM corresponds to the OpenVXI interpreter. Its function is to load the dialogue flow as a scripted sequence of states, with its transitions and procedures, coded in VoiceXML.

⁴⁹ <http://www.i6net.com/products/vxi/>

⁵⁰ <http://www.visibridge.com/>

⁵¹ <http://www.hp.com/>

⁵² <http://jvoicexml.sourceforge.net/>

⁵³ <http://publicvoicexml.sourceforge.net/>

⁵⁴ <http://sourceforge.net/projects/openvxi/>

⁵⁵ <http://www.dihana.upv.es>

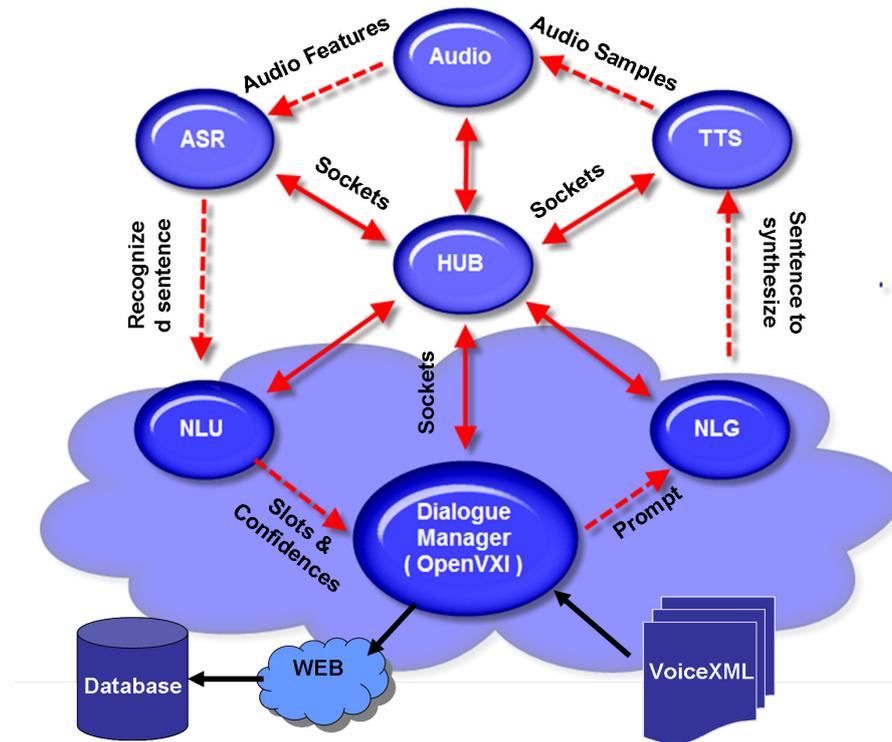


Figure 3.11. Distributed architecture for the runtime system

The interpreter also provides methods for remote access to the service database using a HTTP request to a Web and servlet servers (Apache and Tomcat server respectively in the current implementation). The procedure is exemplified in Figure 3.12. When the DM finds a *subdialog* tag with a *url* address in the VoiceXML file (number 1), it calls the Apache server converting the parameters in the *namelist* attribute into a http request (number 2) to the CGI script file, i.e. *dc_script.cgi* in the example, which in turn prepares the call (number 3) to a Java servlet executed by the Tomcat server. The servlet performs the database access using the corresponding SQL instruction (number 4) and retrieves the information back to the Apache server (number 5). Finally, the cgi file in the Apache server transforms the information (number 6) into a dynamic VoiceXML file (number 7) that the DM uses to get the DB results and to present them to the final user.

The next component in the distributed architecture is the Natural Language Generator (NLG). This module is responsible of generating a natural language message that conveys the information retrieved from the database or the different messages when the system needs to provide a confirmation or clarification message to the user. [McTear, 2004] describes three main approaches for language generation: canned text, template filling and planning. VoiceXML supports the first two. In the current platform, the canned text can be supported when a database field contains the textual description of the information to be provided to the user. For the second approach, the VoiceXML file has to specify different prompts, with slightly variations, according to different situations, e.g. when the system cannot recognize a spoken utterance the first, second or third time, or it can specify different message according to the user level or number of retrieved items (see section 3.4.2 and 3.4.3, page 65), etc. Since the language generation is 'embedded' into the VoiceXML file, this module does not need to be present in the real-time system.

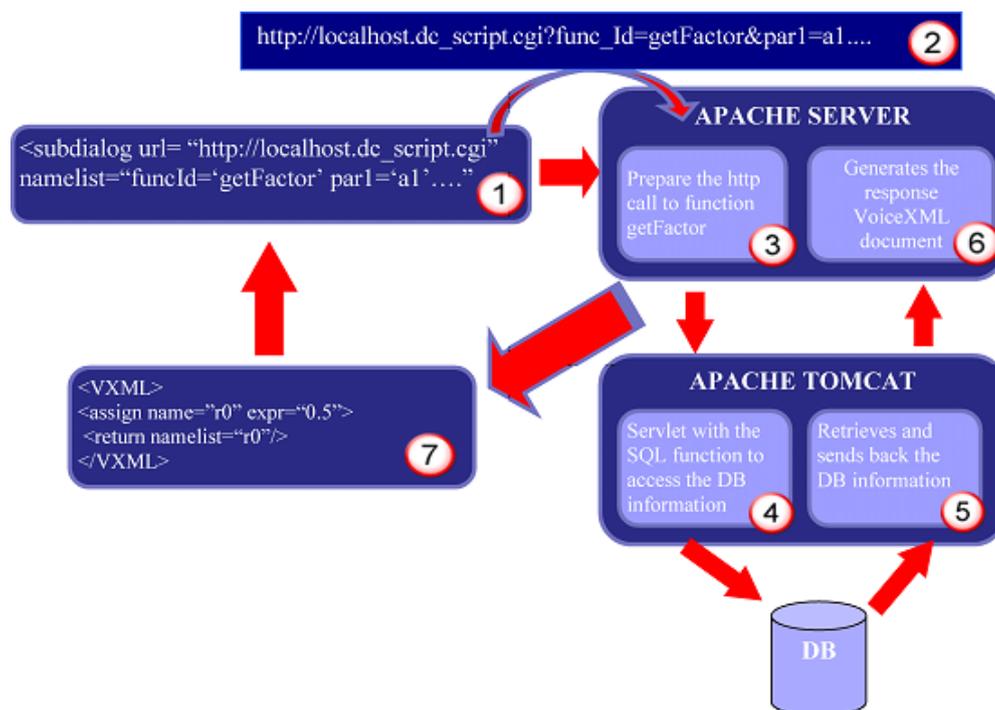


Figure 3.12. Procedure to retrieve information from the database using the runtime system

After the natural language generator module constructs the message to be presented to the user, the TTS module converts it into a spoken form. The simplest case is to play a previously recorded message, with the possibility of filling slots with the information retrieved from the database. However, in most cases the TTS plays the message provided by the language generator. The process involves analyzing the text message to convert it into a linguistic representation (i.e. phonemes) that can be used by the speech generator to produce the synthetic speech.

The next module is the audio server; it has two main functions: a) to play to the user the synthetic speech (i.e. audio samples) received from the TTS, b) to collect the audio input from the microphone/telephone that the final user uses to communicate with the system. The module can send directly the audio samples to the speech recognizer module or it can first process the samples in order to send audio features (i.e. MFCC, delta and delta-delta features) instead.

The next module in the platform is the Automatic Speech Recognizer (ASR). After processing the audio features, it produces an N-best list with the most probable sentences for the given utterance. During this process, the recognizer loads the speech grammar that restricts the sentences that the system can recognize. The language models, supported by our platform, are stochastic n-gram models. Finally, the system also assigns a confidence score for each word in the recognized sentence.

Then, the Natural Language Understanding module (NLU) loads the corresponding semantic interpretation grammars⁵⁶ in order to analyse the recognized sentence and to assign its meaning, i.e. interpretation, which is used later by the dialogue manager. In general terms, the interpretation can be considered as a simple assignment between relevant sections of the uttered sentence and the slots that the application defines. For instance, a sentence like: "I

⁵⁶ <http://www.w3.org/TR/semantic-interpretation/>

want to flight from New York to London” can be “interpreted” as `departure_city:NY` and `arrival_city:London`. Following the standard VoiceXML specification, our real time platform allows two kinds of speech grammars: SRGS⁵⁷ (Speech Recognition Grammar Specification) and JSGF⁵⁸ (Java Speech Grammar Format) files. The main advantages of using these finite state grammars were two. First, since these grammars specify, through a finite number of rules, the whole set of grammatically correct sentences that the final user can speak to the system, it was easy to use them to filter the n-best list of sentences recognized by the ASR using statistical grammars that do not necessarily provide grammatical sentences but provide robustness to the system. Second, the specification of both kinds of grammars defines special tags that allow the semantic interpretation of the sentence at the same time. This way, it was not necessary to include new parsers or modify the VoiceXML browser in order to use a new grammar specification. The runtime platform allows both kinds of grammars to be referenced in the VoiceXML file using inline or external links, and described in two formats, as Augmented Backus-Naur Form⁵⁹ or in XML Form. In our runtime platform, the first format was used for specifying the JSGF files, and the second one for SRGS files.

```
<?xml version="1.0"?>
<grammar root="Identification" version="1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="MoneyQuantity" scope="public">
    I want to convert <ruleref uri="#Number"/> <ruleref uri="#Currency"/>
    into <ruleref uri="#Currency"/>
  </rule>
  <rule id="Number" scope="private">
  <one-of>
    <item>one hundred<tag><![CDATA[number = 100]]></tag></item>
    <item>two hundred<tag><![CDATA[number = 200]]></tag></item>
    ...
  </one-of>
  </rule>
  <rule id="Currency" scope="private">
  <one-of>
    <item>euros<tag><![CDATA[currency = euros]]></tag></item>
    <item>pounds<tag><![CDATA[currency = UK_Pounds]]></tag></item>
    ...
  </one-of>
  </rule>
</grammar>
```

Figure 3.13. Example of a SRGS grammar file used by the NLU module in the run-time system

Figure 3.13 shows an example of the SRGS file used in our banking application. In this example, there are three rules, one public (*MoneyQuantity*) and two privates (*Number* and *Currency*). The public rule contains the sentence the user can speak, including references to the private rules for completing the sentence. The one-of element identifies the set of alternative elements for the private rules. In this case, the recognized sentence must contain at least one of these items. Finally, the item element includes a special label tag, [CDATA] that

⁵⁷ <http://www.w3.org/TR/speech-grammar/>

⁵⁸ <http://www.w3.org/TR/jsgf/>

⁵⁹ <http://rfc.net/rfc2234.txt>

defines the semantic interpretation for each item. In the example, the semantic interpretation for the Number rule allows the conversion of the string one hundred into its numeric representation (100). This number is then assigned to the slot number as semantic interpretation.

Another important issue performed in this module is related to the confidence given to the recognized sentence and to the confidence assigned to each word in the sentence (this confidence can be different to the one provided by the speech recognizer). Several methodologies have been proposed for assigning these confidences [Jiang, 2005][Ferreiros et al, 2005][San-Segundo et al, 2001b]. In our case, the module does not modify the confidence at word level given by the speech recognizer, but it assigns as sentence confidence the mean of the confidences values assigned to each slot, not words, in the recognized sentence (see Eq. 3.1).

$$Sentence_Conf = \frac{1}{N} \sum_i^N slot_conf_i$$

Eq. 3.1

The idea of not including all the words in the equation is to avoid contributions from spurious or non-relevant words. Finally, this confidence is assigned, at run-time, to a global variable called fConfidence that is used in the flow defined in the Modality Extension Retrieval Assistant for Speech (MERA-Speech, see sections 3.4.2 and 4.6.2, pages 65 and 113) for the confirmation handling (i.e. for handling nomatch, explicit, implicit and none confirmation).

3.5.4 Portability and Use of Standards

Amongst the main objectives of our platform are portability, meaning independence of the operating system and the runtime platform, scalability, widespread use of standards and feasibility to use existing or new technologies. This section describes our main efforts to successfully fulfil these objectives.

In first place, all the graphic components of the platform have been programmed and generated using Qt⁶⁰, which is a multi-platform (Linux, Windows, Mac, X11) integrated development environment, compatible with C++, with which the designer can write code that can be executed in different operating systems and development environments, e.g. for Visual Studio, Visual .Net, Borland, just by recompiling. Thanks to Qt, it is possible to have a platform version available for Windows and another one for Linux. Moreover, Qt provides several methods and tools to quickly translate all texts in the graphical interface to adapt them to another language.

We have also used the UTF-8 format (8-bit Unicode Transformation Format), which is a variable-length character coding for the Unicode standard in multiple languages, to help us in our target of multilinguality. Besides, it is the default coding in XML and it is used by all internet protocols. This format is especially crucial in the definition of prompts and grammars for all languages in the service.

⁶⁰ <http://www.qtsoftware.com/>

3.6 Scope and Limitations

As we mentioned in the introduction (Chapter 1), the main objective of the platform is to allow the construction of dialogue applications for multiple modalities and languages at the same time. The generated applications can be used to access services based on database queries/modification (e.g., banking, train reservations, share prices in real time, etc.) through a telephone or a Web browser. Considering the limitations imposed by the standards used in the scripts generated by the platform, it is limited in the current version to the execution of each modality on its own.

In any case, we consider that the platform is well prepared for true multimodality. The only missing things right now are new code elements for synchronization in our XML syntax and a new code generator (e.g., for X + V). For the speech modality, the platform generates a script using the VoiceXML 2.0 standard that allows a certain degree of mixed-initiative dialogues. Regarding the Web modality, the platform generates pages made up with Web forms (including radio buttons, textboxes, combo boxes, etc.), and coded using the xHTML language, so they are accessible from a conventional Web browser.

Besides, the platform allows the coding of multimedia contents (e.g., videos, recordings, images, etc.) as part of user output. Finally, because the output is coded in xHTML, an expert designer might use it as a base to add other more complex audiovisual resources, such as animations, interactive maps, etc., using specialized Web design tools.

In [Allen et al, 1999] four levels of mixing initiative are identified: unsolicited reporting, sub-dialogue initiation, fixed subtask initiative, and negotiated mixed-initiative. Unsolicited reporting allows an agent to inform others about critical information needed out of turn. Sub-dialogue initiation allows the system to initiate a sub-dialogue in certain situations, e.g., to ask for a clarification. In a fixed subtask initiative, the system keeps the initiative for a task, and it executes the task interacting with the user when necessary. In the negotiated mixed-initiative level, there is no fixed assignment of responsibilities or initiative, so agents can negotiate who takes the initiative and proceeds with the interaction based on it.

On the other hand, [McTear, 2002] states that finite-state models are always fixed system-initiative, while frame-based systems may permit some degree of mixed-initiative, but that they may also be fixed user-initiative. Finally, [Allen et al, 2001] survey five levels of systems in increasing complexity of software architecture: finite-state, frame-based, sets of contexts, plan based, and agent based models. Considering this perspective, our platform covers the first two levels of task complexity in Allen's classification, and supports, as a frame-based system, the lower levels of mixed-initiative interaction: unsolicited reporting and a few cases of sub-dialogue initiation for the management of lists of objects (see Section 4.6.1, page 110).

4 SPEED UP STRATEGIES APPLIED IN THE DIALOGUE DESIGN

In this section, all the strategies to accelerate the dialogue design are explained in detail. The goal of all these accelerations is to reduce the design time by simplifying the definition of the different dialogues, actions, and elements required to design and run the service. Moreover, the proposed accelerations guarantee that the generated models are well formed and optimized, as well as contribute to minimize the possibility of the designer making mistakes in the design.

The important contribution of the thesis is that most of the accelerations presented in this section are innovative and non-existing, up to the best of our knowledge, in any commercial and research platform. In case that a similar acceleration is available in any of the current platforms, we have tried to go one-step further by incorporating new automation mechanisms.

In summary, the accelerations can be classified into three classes: Heuristic based, Rule and Context based, and Wizards for simplifying the design process. The first one corresponds to accelerations that use the database contents and the data model structure. The second one corresponds to the application of configurable domain knowledge rules that we have incorporated into the assistants taking into account our experience in designing dialogue systems. On the other hand, context based accelerations correspond to strategies that use the available information saved in previous assistants to generate different kind of proposals that take into account the goal of each assistant and its level of access to the information. Finally, the third one corresponds to other accelerations mainly based on the incorporation of different wizard windows that help designers to automate/eliminate repetitive or common procedures when designing dialogue applications. In relation with the accelerations based on using the data model and database contents, it is important to mention that they can be classified as content-independent and content dependent. In the former case, the system has no access to the contents of the database; this situation is common in situations where the developers have restrictions to access the database due to security reasons since the database could contain confidential information regarding the clients (e.g. pin codes, credit card numbers, etc.). In this case, the platform adapts the functionality of the assistants in order to only exploit the data model structure, at the expense of reducing the number of accelerations available to the designer. In the latter case, the platform has full access to the database contents (i.e. tables and fields) allowing the system to propose new accelerations such as full custom classes and attributes or new dialogue proposals, in addition to the already available accelerations provided by using the data model structure.

Defining a ranking of importance of the proposed accelerations is not an easy task as each one contributes to the definition of the service. However, considering the level of innovation in comparison to other platforms and the effort we did for allowing them in the platform, we can be sure that the accelerations included in the RMA (section 4.5) are the most important ones since this is the assistant where the higher number of information has to be defined in the entire platform. This way the possibility of automatically proposing the actions for each assistant and the possibility of creating dialogues combining mixed-initiative and over-answering capabilities are really important. On the other hand, all the accelerations

included in the MERA-Speech assistant are the second most important accelerations for two reasons. First, because we could not find any similar kind of assistants in other platforms for solving the problem of presenting lists of results and confirmation handling, leaving the solution to the designer without offering any kind of predefined proposal or leaving the solution to the ASR engine used in the real-time system. In second place, because the complexity of defining all the flow for the presentation of results and confirmations considering different conditions (e.g., number of items retrieved, levels of confidence and number and type of the slots to be confirmed) required the creation of innovative templates described in detail in Appendix C.

The chapter is organized as follows: section 4.1 describes the heuristic information extracted from the database contents in order to accelerate the design in different assistants of the platform. Section 4.2 describes the accelerations to create the object-oriented representation of the data model structure used by the platform. In section 4.3, the accelerations for creating the prototypes of the functions used by the runtime system to access the backend database are described. Sections 4.4 and 4.5 describe the accelerations implemented in the assistants for creating and complementing the state flow model of the dialogue service. Section 4.6 explains the accelerations implemented in the assistant that defines the dialogue flow for the presentation of lists of objects retrieved by a database access and the confirmation handling for the speech modality. In section 4.7, the accelerations for creating speech prompts and grammars are explained. Finally, section 4.8 outlines the conclusions of this chapter.

4.1 Heuristics

In order to accelerate the design of the service in different assistants, we have first implemented a new module that automatically extracts heuristic information from the database contents when it is available. These heuristics are obtained using an open SQL query that retrieves all the information from every table in the database. The system automatically collects information regarding the name and the number of the different tables and fields, and the number of records for every table. In addition, for each field the following features are also collected:

- a) The average length in characters
- b) The average number of words
- c) The vocabulary size (number of words that are different)
- d) The proportion of values that are different
- e) The field type
- f) The number of empty values

These features, grouped or individually, are mainly used to accelerate the design or to improve the presentation of information in many assistants of the platform, as we will show in the following sections. For instance: (a), (b), (c) and (d) have been used to detect candidate slots to be requested using mixed-initiative dialogues (see section 4.4.3, page 99), (e) accelerates the creation of the data model structure (section 4.2.1, page 89) and to create and debug SQL statements (section 4.3.2, page 93), (f) is used to sort by relevance the attributes displayed by the wizard when creating the database structure (section 4.2.1, page 89) and when proposing dialogues to retrieve information from the user in the RMA (section 4.5, page 101).

An important issue we observed when retrieving the field type was that sometimes the metadata information provided by the SQL function was incorrect due to: a) the driver for accessing the database was only able to return a limited number of field types, hence some types like Boolean or dates were mapped as integer or string types respectively, b) the designer of the database defined a field using a generic type such as string or float when they actually corresponded, for instance, to dates or integers, and c) we found problems for mapping special types such as hyperlinks, or currencies, etc. into the types supported by the platform.

In order to correctly identify the field type, which results in a considerable reduction of the number of times that the designer will need to change the proposed type for a given attribute when creating the classes (see Figure 4.1), we implemented a post-processing step to confirm or reassign the types returned by the metadata information from the database using a special SQL query. The post-processing is made using regular expressions (RE) to detect the following types: integer, float, date, string, Boolean, empty fields, or mixed (e.g., URLs, emails, binary info, etc.). During this step, the system analyzes all the non-empty values for a given field and selects as field type the one that appears more than 90% of the times. The exceptions to this criterion are: a) a numeric field is considered integer if all its records are classified as such; if not, it is classified as a float, b) the empty type is assigned to fields containing more than 95% of the time empty values.

With the purpose of analyzing the performance of the regular expressions, an objective evaluation was carried out. In this evaluation, twenty-one databases, most of them available online, were retrieved and visually inspected field by field. In total, there were 109 tables (an average of 5 tables per database), 767 fields and 610,506 records.

		Real Type							Total
		Integer	Float	Date	String	Blank	Mixed	Bool	
RE Type	Integer	205 94.0%	2 0.9%	0	7 3.3%	1 0.5%	0	0	215
	Float	0	60 96.8%	0	1 1.6%	1 1.6%	0	0	62
	Date	0	0	43 100%	0	0	0	0	43
	String	0	0	0	336 99.1%	2 0.6%	1 0.3%	0	339
	Blank	1 2.8%	0	0	1 2.8%	34 94.4%	0	0	36
	Mixed	2 6.7%	4 13.3%	0	8 26.7%	0	16 53.3%	0	30
	Bool	5 11.9%	0	0	0	0	0	37 88.1%	42
	Total	213	66	43	353	38	17	37	89.6%

Table 4.1. Confusion matrix for automatic field types detection comparing the human classification (real type) and the proposed type by the system (RE type)

Table 4.1 shows the performance of the different regular expressions (RE) when compared with the classification made by a human evaluator. According to this table, the average recognition is 89.6%, obtaining the best rates for dates, strings, and numeric quantities, which are the most common types in most databases. Analyzing in detail the misrecognitions, 0.9% of floats were incorrectly detected as integers due to values such as 2.0, 30.0, etc. which were automatically returned by the database driver without the decimal part. Another source of errors was detecting some numeric quantities due to special symbols such as dashes, percentages, or the euro symbol, which were incorrectly interpreted as a string type (3.3% and 1.6%). The major problems occurred for the Mixed type. Here, the system was not able to distinguish between this type and the String type since they are, in practice, the same. However, we wanted to separate them classifying as Mixed things as: URLs, emails, long strings, etc., since for a speech recognizer they may be handled using different strategies (e.g. spelling, general grammars, etc.).

On the other hand, Table 4.2 shows the confusion matrix for comparing the type retrieved by the driver and the real classification made by a human evaluator. Observe that for some columns (e.g. blank, mixed and Boolean) the system cannot distinguish from its actual type since the driver does not return them correctly (for instance, Booleans are returned as integers, -1, 0, or 1) or because they do not exist as such but were defined for this thesis only (blank and mixed). However, checking again Table 4.1 we can see that the regular expressions were robust enough to provide a good performance for these special types.

		Real Type							Total
		Integer	Float	Date	String	Blank	Mixed	Boolean	
Driver Type	Integer		0	0	7	0	0	30	162
	Float	37		0	0	0	0	0	77
	Date	0	0		0	0	0	0	43
	String	51	26	0		38	11	7	478
	Blank	0	0	0	0		0	0	0
	Mixed	0	0	0	1	0		0	7
	Boolean	0	0	0	0	0	0		0
	Total	213	66	43	353	38	17	37	767

Table 4.2. Confusion matrix comparing the human classification (real type) and the driver classification (driver type)

4.2 Strategies Applied to the Data Model Assistant (DMA)

As described in section 3.2.2 (page 60), in this assistant the data model structure of the service is created by the definition of object oriented classes. The objective of these classes is to provide information about which fields in the database are relevant for the service and how these fields can be grouped together. Therefore, we can think that the attributes in a class correspond to the possible fields to be requested or presented to the user in one or more dialogue states. Each class can be characterized by a list of attributes and optionally a list of base classes (inheriting their attributes). The attributes may be: a) of atomic types (e.g., string, Boolean, float, date, etc.), b) complex objects, they refer to an existing class (e.g. ObjRefr or ObjEmbed), or c) lists of either atomic type items or complex objects. Since this

is one of the first assistants in the platform, a significant effort was done in order to accelerate the creation of the database structure and to include relevant information that can be used for other assistants in the platform.

The main acceleration included in this assistant is the incorporation of a new wizard window that uses the heuristic information described in the previous section to propose full custom classes and attributes that the designer can use when creating the structure. In addition, if the system has not access to the database, the assistant also provides the following accelerations: a) re-utilization of libraries with models previously created, which can be copied totally or partially, b) automatic creation of a class when it is referenced as an attribute inside another one, and c) definition of classes inheriting the attributes of a base class. These accelerations were incorporated during the GEMINI project by other partners of the project.

4.2.1 Semi-automatic Classes Proposals

In order to allow the designer to create custom classes selecting the tables and fields from the database or from already existing classes in the model, we have included a new wizard (see Figure 4.1) that using the heuristic (e), the field type (section 4.1, page 86), automatically sets the field types in the wizard. For example, in Figure 4.1, the field type for “minimum debit” in the database is string, but the wizard changes it to integer because all its values are actually this type. In any case, this can be modified by the designer. Besides, in order to first show the most important or relevant fields for each table in the database we have included a simple mechanism to sort the fields in the assistant by relevance using the heuristic (f), the number of empty values in a given field. This way, if the number is high, then the system considers that it is unlikely that this field will be used to request information to the user, then the attribute is placed at the bottom of the list and displays a warning message when the attribute is selected.

Moreover, the assistant accelerates the design proposing automatic names when a new class is being created; in this case, the proposed name is a combination of the selected attributes. Besides, the system also proposes an automatic name when it detects that the class or any attribute has the same name as a previously defined one; in this case, the proposed name is the original name of the class/attribute plus a sequential number. Then, when the designer finishes the creation of the class, the assistant automatically saves all the information defined in the wizard window in the final GDialogXML file for this assistant. When the designer selects an attribute from a field in the database, this information is also saved in the output xml file using the GDialogXML tag xDataMAttr. In this way, subsequent assistants will use this information to implement better acceleration strategies.

Finally, in order to reduce the information displayed and to make the wizard more intuitive, the first time the designer uses it, if the number of tables in the database is too high it is possible to select those that will be actually needed during the design. In addition, it is also possible to customize the name of the tables in the database. This feature could be especially relevant if the database designer is different from the dialogue designer or if the names of the tables and fields are not very intuitive. . In case the designer uses a custom name, the assistant will present the information of the table/field using the custom name, although internally and when the final XML file is saved, the real name is used.

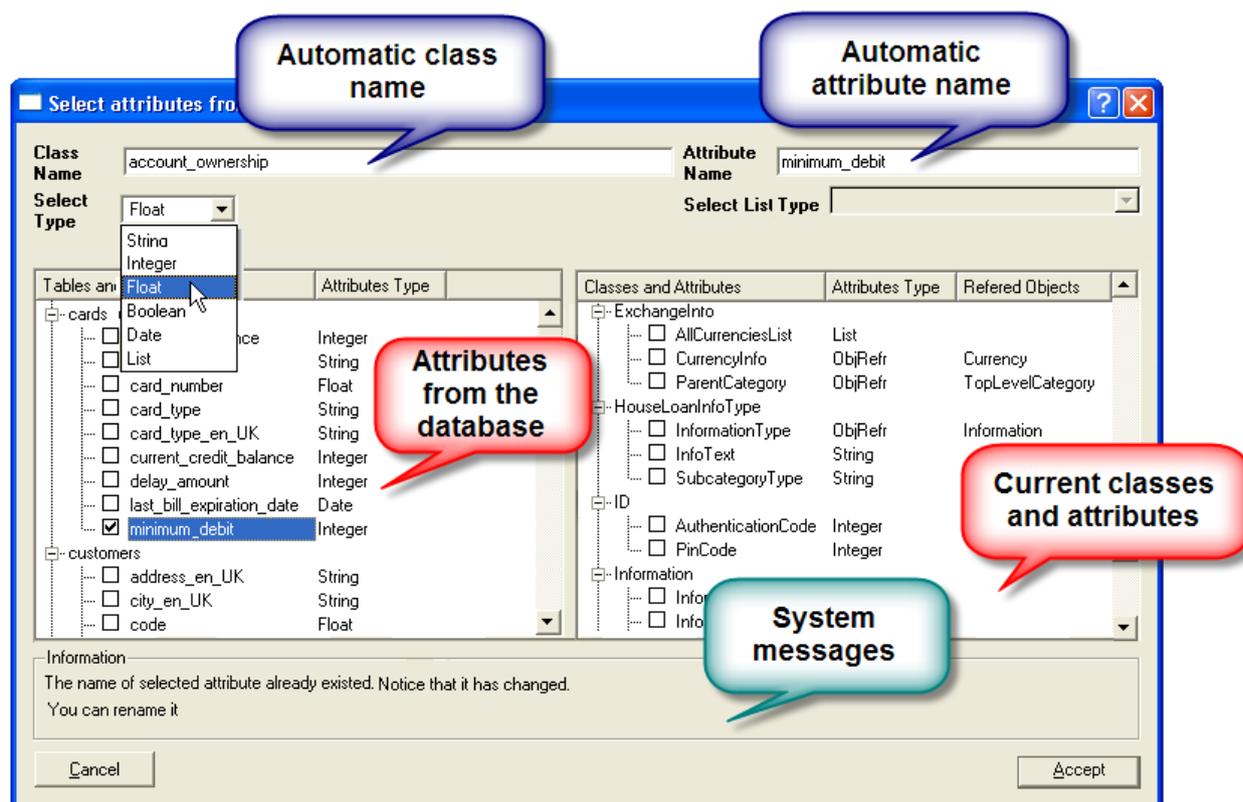


Figure 4.1. Form fill-in window that allows the creation of custom classes (from the database and classes from the current model) in the DMA.

4.2.2 Common Accelerations

Besides, for both approaches (content dependent and content independent), the design of the data model is accelerated in the assistant by the following features:

1. Re-utilization of libraries with models previously created, which can be copied totally or partially, or a new class can be created by mixing several original classes. The assistant allows the creation and loading of libraries. In this way, it is possible to take advantage of previous knowledge or from previous prototypes of the service in order to improve it.

2. Automatic creation of a non-existing class when it is referenced as an attribute inside another one. For instance, consider the example shown in Figure 4.2. In this case, suppose that the designer is defining the attributes for class *Account*. When the complex attribute *AccountHolder* is included into the class, the assistant automatically searches the referenced object, i.e. the class *Person*, in the internal list of already defined classes. Since this class has not been defined previously, the assistant automatically creates it as an empty class. Afterwards, the designer can edit the new class including the attributes that belong to it. This way the assistant allows a top-down design. In the example, the same process is done for the reference class *TransactionDescription*.

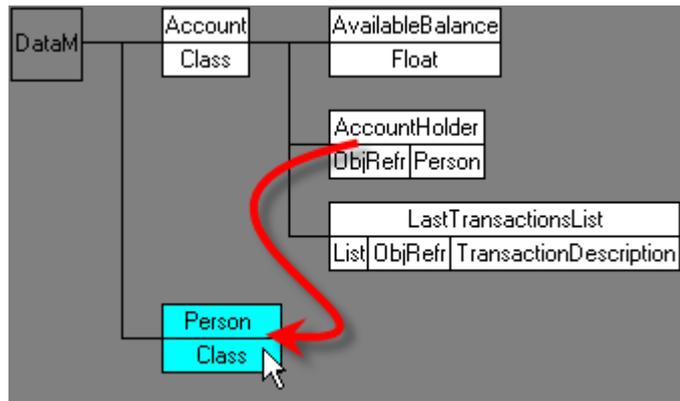


Figure 4.2. Example of the automatic creation of a referenced class

3. Definition of classes inheriting the attributes of a base class (i.e. parent classes). In this case, when defining the new class, the designer only needs to specify all the classes to be used as base classes. Then, the assistant automatically inherits all the attributes defined in the selected base classes into the new class. This way, the platform uses concepts inherited from object-oriented programming.

4.3 Strategies Applied to the Data Connector Model Assistant (DCMA)

According to section 3.2.3 (page 61), this assistant allows the definition of the prototypes (i.e. the input and output parameters) of the database access functions that are called from the runtime system. The platform only requires the prototypes because they provide enough information for the following assistants of the platform and their actual implementation is not needed when designing the dialogue flow. However, it is also possible to take advantage of this assistant in order to create the actual implementation of such functions and to include meta-information to accelerate the dialogue design in subsequent assistants.

The main acceleration strategy, designed by the partners of the GEMINI project, is the association of the input/output parameters to attributes and classes from the data model structure. This information is especially useful for the Retrieval Model Assistant (RMA) and the State Flow Model Assistant (SFMA) in order to create dialogue state proposals and to propose database access functions for a given state in the design. In addition, the author of this thesis has contributed with a new wizard window that allows the automatic generation and debugging of the SQL queries needed to perform the functions in the real-time system. This wizard is useful for designers with little knowledge on query languages and can be used to check if the prototypes have the correct number of input/output parameters. This acceleration is also interesting since most of the current development platforms do not include such kind of accelerations, and in those where we found a similar assistant it did not automatically propose the SQL query but only allowed to specify and debug it.

4.3.1 Definition of Relations between the Function Arguments and the Data Model

As mentioned above, the first, and main, acceleration strategy included in this assistant is the possibility of defining the relation between the input/output arguments of the database

access functions and the attributes and classes from the data model, which were defined in the previous assistant. All this information is kept in the output model, which is going to be used automatically in future stages of the design process (see section 4.4.2.2 and 4.5.2, pages 98 and 103).

In Figure 4.3, the code generated by the assistant for the banking example is shown. In this case, we show the GDialogXML code generated by the function *PerformTransactionFromDebitAccountToCreditAccount* using the same process shown in Figure 3.3. This function has three argument variables to collect the information regarding the accounts and the quantity to transfer, and one returning variable defined as Boolean. In the code, the tag `xArgumentVars` (number 1) contains the information regarding the input parameters: the debit account number (*DebitAccountNumber*), the destination account (*CreditAccountNumber*) and the amount to be transferred (*TransactionAmount*) and the tag `xReturnValueVars` (number 2) contains the return argument *TransactionPerformed* (in this case, a Boolean variable that indicates if the transfer is successful or not). In the figure, the variable *TransactionAmount* has a dependency, specified through the `xDataMAttr` tag, with the data model attribute *Transaction.TransactionAmount*. This dependency will be used in the posterior assistants (i.e., SFMA and RMA) to create dialogue proposals and the automatic proposal of a database access function for a given state in the design. An important acceleration included during this process is that the assistant automatically proposes the class and attribute which is more likely to be related to the argument, as well as the database table and field. The mechanism is to use the name of the argument being edited to search for similar classes or attributes in the data model structure. The table and field of the database is extracted from the data model since this information has been already defined in the previous assistant.

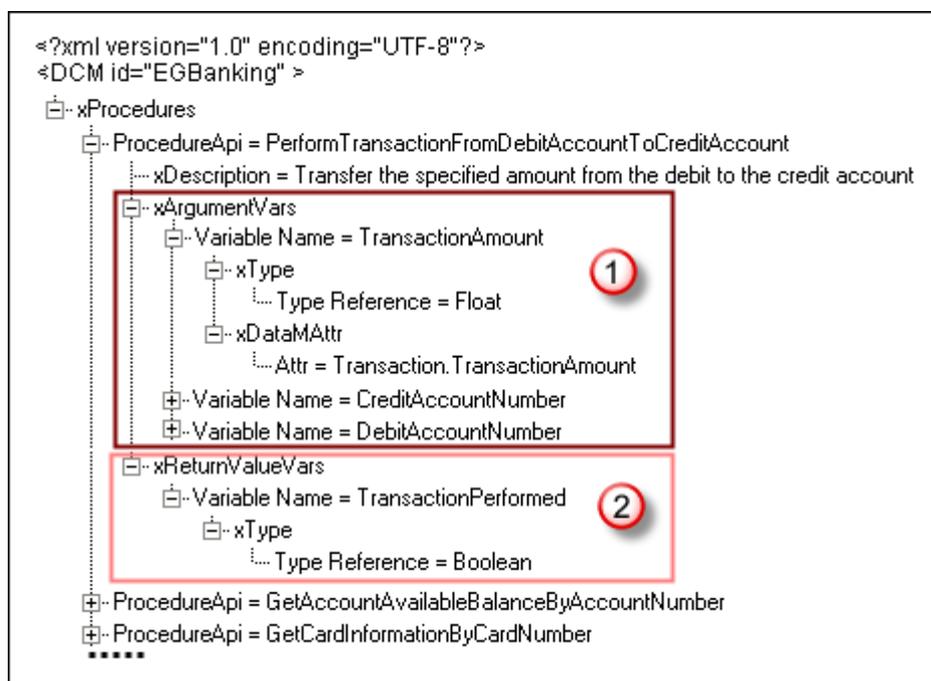


Figure 4.3. GDialogXML code generated by the DCMA for the bank transfer.

4.3.2 Automatic Generation of SQL Queries

The second strategy, proposed and implemented by the author of the thesis, was the inclusion of an assistant that generates automatically the SQL query for a given function. The main motivation behind this assistant was to reduce the necessity for the designer of knowing a new programming language and, at the same time, to simplify the inclusion of the generated query in the Java servlet created for accessing the database at runtime.

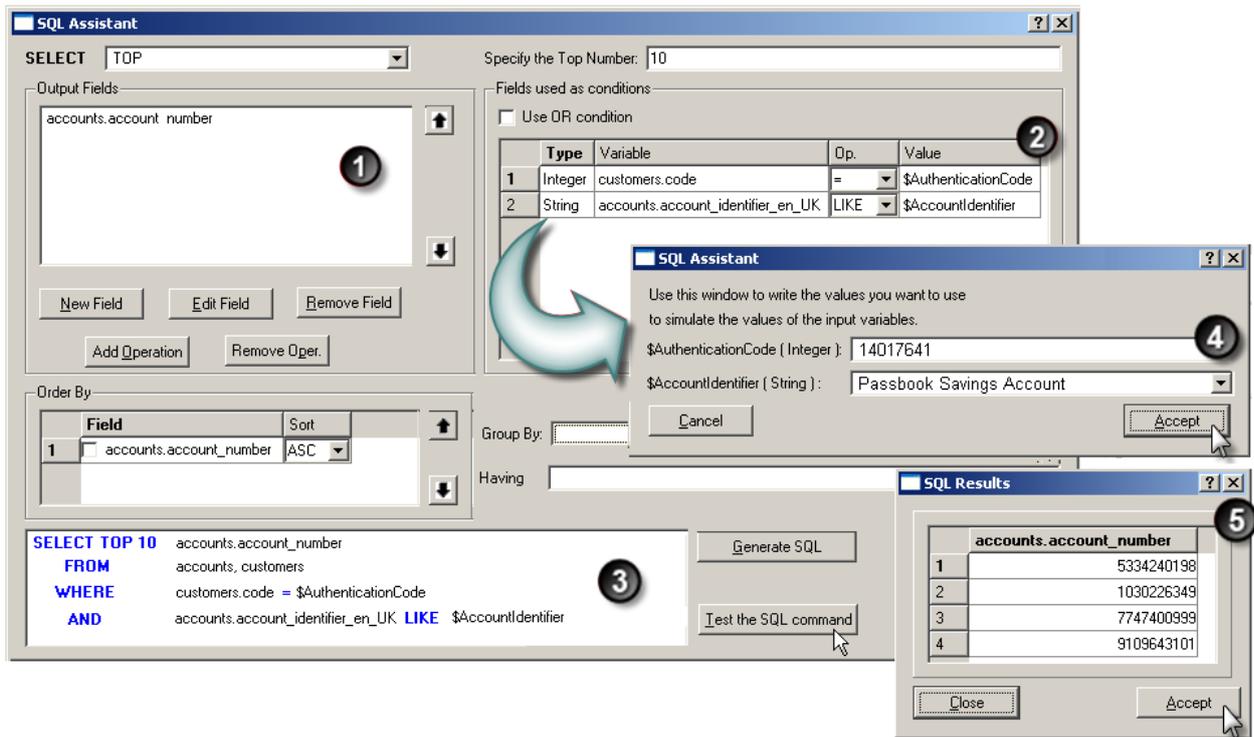


Figure 4.4. Form fill-in window for the automatic creation and testing of SQL queries for database access functions.

Figure 4.4 shows the main window of this assistant. The assistant allows the inclusion of several constraints supported by the SQL language such as maths functions (average, max, min, ln, exp, etc.), sorting, selection (Top or Distinct), clustering (Group By), Boolean operators (And, Or) for combining the query restrictions, among others.

In order to automatically create the query, the assistant uses the input arguments (defined in the function prototype, see number 2 in the figure) as constraints for the WHERE clause, and the information of the output arguments as returned fields for the SELECT clause (number 1). During this process, the wizard also uses the heuristic (e), the field type, in order to correctly create and debug the SQL statement. The assistant allows the inclusion of new input or output arguments if the function prototype is not complete or if the designer wants to test new argument combinations. The next step is to generate automatically the SQL sentence. It is presented in a textbox (number 3) that the designer can use to edit the proposed query.

Since the input/output arguments could be defined using different types (i.e. atomic: string, integer, float, etc, or object oriented such as list, embed, and reference) several strategies were applied in order to create SQL queries that can use such kind of parameters. In general, if the argument is atomic then the query uses the argument directly. However, a

special case appears when the returning argument has been defined as a Boolean type since there are two possibilities. The first one is that the associated database field is Boolean too; in this case, there is no problem since the regular expressions do the association directly. The second possibility is that the returning parameter does not have an associated database field; in this case, we need that the query returns this type. In order to do it, the assistant builds a query that finishes in a Boolean comparison between the number of retrieved records and zero records (i.e. $\text{count}(\text{records}) \leq 0$). The most complex case corresponds to object-oriented arguments. In this case, if the returning argument is an object the system analyzes the object class and provides a list with the atomic elements of that class in order to allow the designer to select the corresponding arguments to be used in the SQL query from that class. Then, the query is created returning all the selected arguments. Finally, it is the servlet the responsible of taking all the retrieved fields and returning the object, or a pointer to the object, used by the final script. A similar case appears if the argument is a list. In this case, the system looks for the type of the elements in the list and allows the designer to select the corresponding table and field if that information was not defined when creating the function prototype in the previous step (see section 4.3.1, page 91).

In addition, the assistant has a debug window (element number 5 in the figure) that allows the designer to view the retrieved records when using the proposed query. In order to debug the query, the assistant first asks for specific values for the input arguments of the function (using a pop-up window, see number 4). The assistant detects automatically the type of each argument and pre-process them in order to avoid problems when performing the query (i.e. escaping especial characters, confirming that the introduced values correspond to the type of the fields, etc.). Finally, the system shows the retrieved results that allow the designer to know if the query is correct or not.

4.4 Strategies Applied to the State Flow Model Assistant (SFMA)

As mentioned in section 3.3.1 (page 62), in this assistant, the designer defines the state transition network that represents the dialogue flow at an abstract level, i.e. specifying only the high-level states of the dialogue, the slots to be asked to the user, and the transitions between states, but not the specific details of each state. The specific details will be defined in the following assistant, the RMA, which is drastically accelerated thanks to these high-level states of the dialogue specified here.

Considering the different versions of this assistant released throughout the GEMINI project, the author of the thesis has contributed with several improvements and accelerations that are described next. In summary, the new accelerations are the automatic generation of state proposals, the possibility of specifying the slots through attributes offered automatically from the data model, and the unification of the slots to be requested. In addition, the new GUI allows the definition of new states using wizard driven steps and a drag-and-drop interface.

4.4.1 Functionalities Included in the Graphical User Interface

One of the first conditions imposed to this assistant was that the graphical user interface would allow several editing and visualization capabilities such as the possibility of creating the flow diagram using a tree-structured description. In this kind of representation, each leaf and branch represents a state and a corresponding transition. This kind of visual representation is common in most of the commercial and research platforms [McTear, 1998]

because it simplifies the visualization of the flow through its different states and transitions, although it is limited by the complexity of the task, because as the number of states grows the visualization degrades. Several strategies have been proposed to solve this problem (see sections 2.1.1 and 2.1.2, pages 8 and 16). For instance, it is possible to reduce the displayed information dividing the design into different layers (e.g., dialogue flow and error handling), providing more or less information, or encapsulating common actions or a big number of actions into a single object. In our platform, depending on the assistant, the GUI allows the designer to show detailed or minimum information about the states, as well as some degree of encapsulation using libraries. In relation to this assistant, we implemented an automatic algorithm that helps the designer to place the objects in the canvas and reduces the visualization problems produced when all the transitions between states are displayed.

In detail, the algorithm is applied each time the designer connects two or more states among them. By default, transitions are displayed in the canvas using a solid line connecting the states. However, the algorithm is used to automatically evaluate if it is suitable to use a line or to use a connector symbol, like the ones used in flow charts, instead. The decision mainly relies on two factors: 1) the distance between the connected states in the GUI, and 2) the number and size of the objects that are along the path of the connection line. The distance is calculated as the hypotenuse between the x and y coordinates of the two states to be connected. If the distance is longer than one third of the size of the canvas (i.e. the workspace that the designer views without using the horizontal or vertical scrollbars) then a connector is used, if not the system evaluates the second factor. In the next case, the system evaluates the existence of other state boxes along the path of a straight solid line connecting the selected states. In case there are not collisions, the system uses the solid line. If not, the system evaluates the size of the intersected object; in case the size is small (when compared to the canvas size in a given ratio), the system uses the solid line, if not it uses the connector.

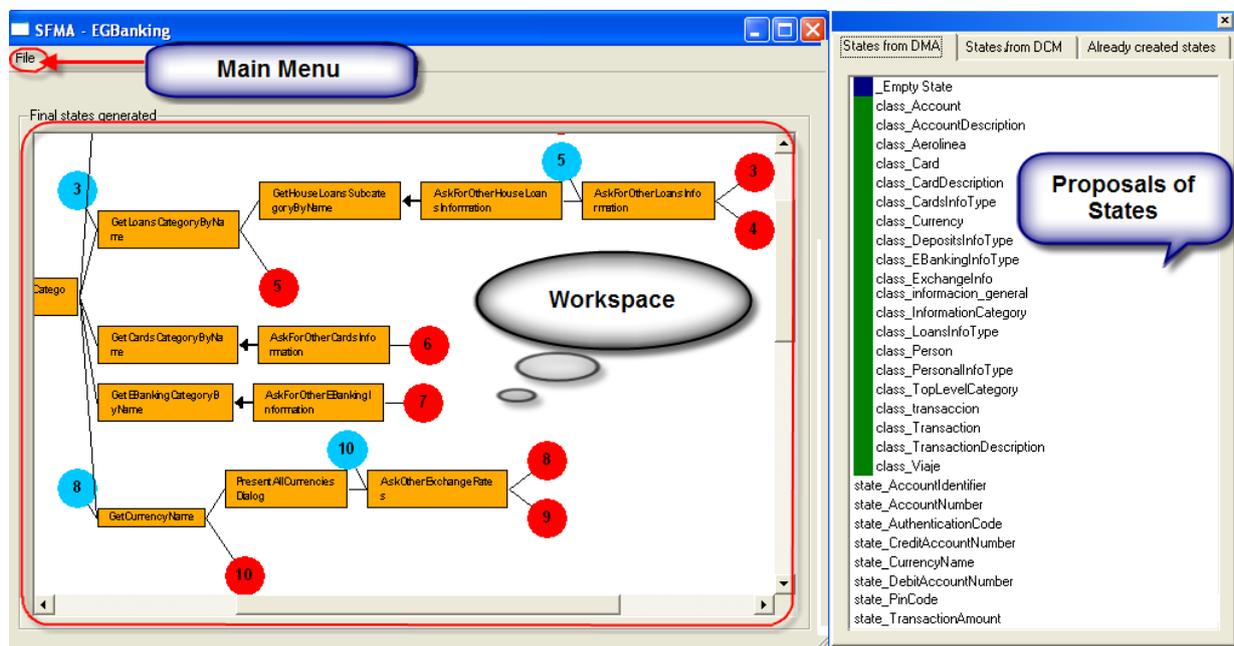


Figure 4.5. Appearance of the SFMA main window

As described above, the main objective of this algorithm is to avoid the creation of a confusing network of crossing lines or to force the designer to follow long lines beyond the

area of visualization of the canvas. After finishing the connection, in case the system had selected the solid line if the designer clicks over it the line is repainted using a thicker line in order to distinguish it from the other lines. In case the system had selected the connector symbol if the designer clicks on it, the system automatically moves the cursor to the location of the other state associated with the selected connector.

In addition, the assistant includes an automatic procedure to rearrange the objects in the canvas. In this case, the system takes advantage of the algorithm described above and a built-in function provided by the programming language and objects (i.e. Qt). In any case, the designer can modify the location of any object in the GUI dragging and dropping it to any place in the canvas.

Finally, the system uses an internal xml file where all the objects (i.e. input/output connectors, states, lines, etc.) and their attributes (i.e. name, colour, position, size, etc.) are saved in order to use this file the next time the designer loads the current design.

Figure 4.5 shows the main window of this assistant, including an example of the visualization of the canvas (workspace), the states, transitions, and connector symbols. For instance, observe that the state `GetCurrencyName` is connected to the state `AskOtherExchangeRates` by the connector number eight (8). Without the connector symbols it would require a confusing line connecting them from one extreme of the canvas to the other (a similar case applies for the connectors 3 and 10).

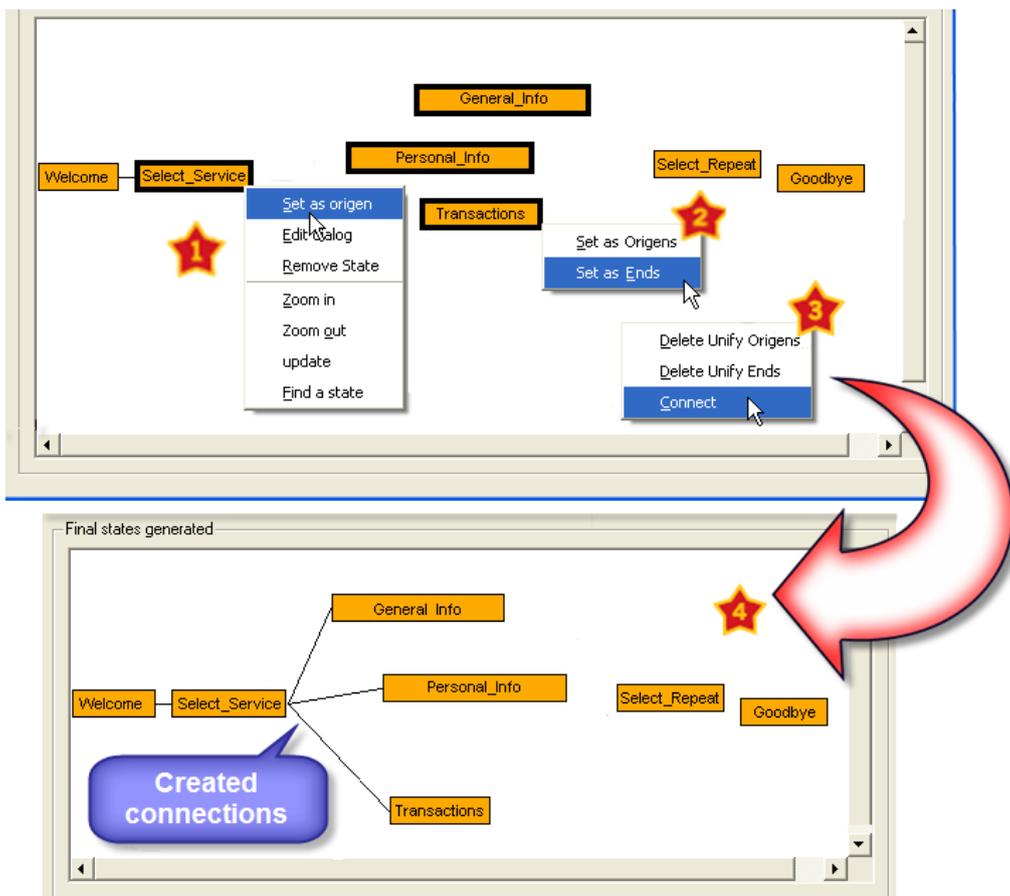


Figure 4.6. Process for the creation of a 1:N transitions in the SFMA

The main window allows designers to create new states just dragging and dropping them from the floating window with the proposal of states, or using contextual right click commands. Besides, the GUI allows the creation of different types of transitions between states such as N:1, 1:M or N:M. In all the cases, the procedure is to select first the set of initial states and then the target states. After that, the assistant automatically creates and shows the connections (see Figure 4.6). As usual in other GUI, the designer can select or unselect nodes using the Ctrl key and the mouse pointer. Finally, several other GUI actions such as find/create/delete/edit a state, or zoom in/out the workspace are also available.

4.4.2 Automatic State Proposals for Defining the Dialogue Flow

One of the most important accelerations proposed and implemented in this assistant by the author of the thesis was the automatic proposal of dialogue states that include the slots to be requested to the user. The advantage of these proposals is that they can be used directly by the designer with little or no modification. In order to create these proposals, the assistant uses the information from the database structure (from the DMA) and the prototypes of the access functions to the database (from the DCMA). The proposed states are available in a floating window through the GUI (see Figure 4.5). The next sub-sections provide a detailed explanation of the algorithm used to generate these state proposals.

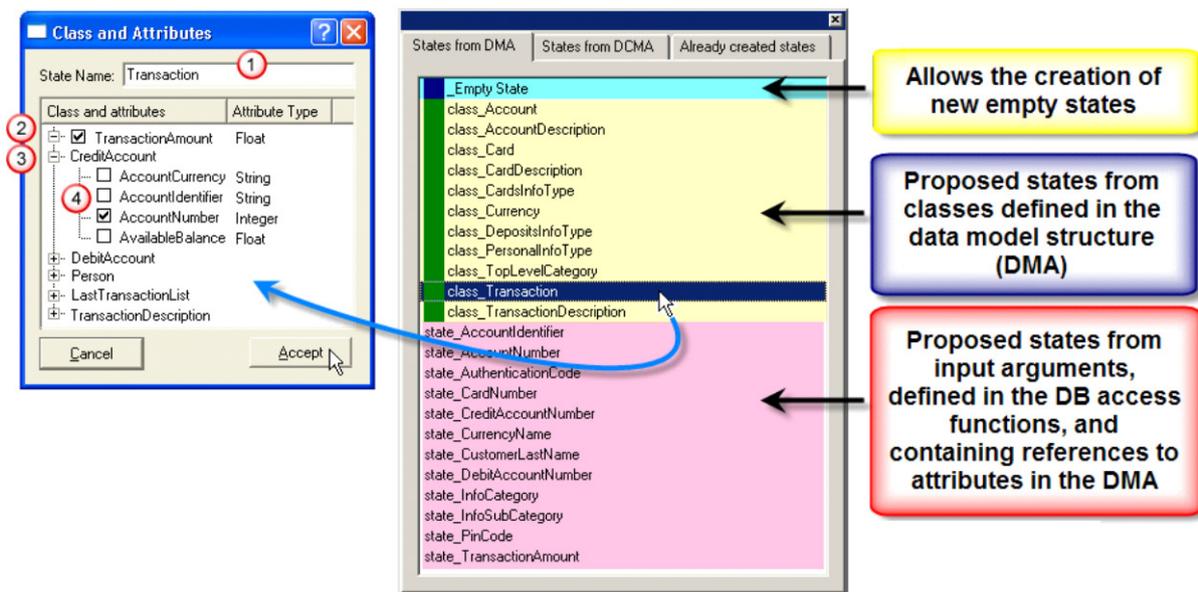


Figure 4.7. Pop-up window with states proposals from classes defined in the data model structure (DMA)

4.4.2.1 Class dependent states

“States from DMA” in Figure 4.7. For each class defined in the DMA, the assistant creates a class template, identifiable by the prefix “class”, which the designer can drag and drop into the workspace. A pop-up window allows the designer to select the attributes to use as slots in the new state. The assistant also allows the designer to select multiple templates in order to create the new state. In this case, the pop-up window shows all atomic attributes that belong

to the selected classes. The name for the new state is automatically generated from the selected classes, but the designer can change the name. Finally, the new state is inserted into the workspace allowing the designer to define the transitions (i.e. connections) to other states. Figure 4.7 shows an example of using the template class `Transaction`. In this case, the designer selects the attributes `TransactionAmount` (number 2) and `AccountNumber` (number 4) to be used as slots in the new state `Transaction` (number 1). Observe that the assistant expands complex attributes (with inheritance and objects) allowing only the selection of atomic attributes because only these attributes can be asked to the user in the real time system (number 3 and 4).

4.4.2.2 States from attributes with database dependency

This kind of states is created from any attribute defined in the database model (DMA) that refer to a database field and conditioned, at the same time, which the attribute has been used as an input argument in any database access function defined in the DCMA. The proposed states are also included in the “States from DMA” tab and include the prefix “state” (see Figure 4.7). The main motivation for proposing these states is that these attributes are likely to be asked to the user. The proposed states contain only one slot and its name corresponds to the name of the attribute in the data model. However, the designer can select several states before making the drag and drop allowing the creation of states with multiple slots. The proposed name, as in the previous case, is automatically generated from the selected classes but can be edited afterwards.

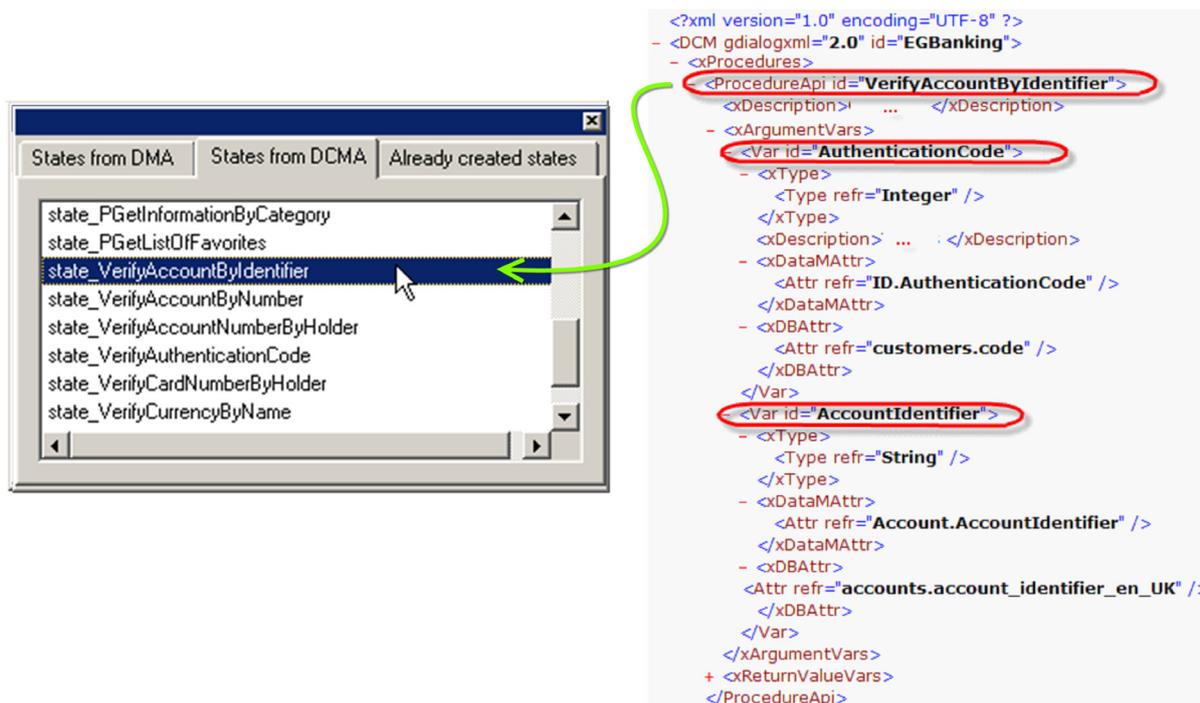


Figure 4.8. Example of a proposed state from a defined database access function. The GDialogXML code corresponds with the definition of the function in the DCMA.

4.4.2.3 From the database access functions

“States from DCMA” in Figure 4.8. In this case, the system analyzes all the prototypes of the database functions defined in the DCMA containing input arguments defined as atomic types. Then, the system uses the name of the function as proposal for the name of the state, and the input arguments as slots for that state. Again, the assistant allows the designer to select

several of these proposals when making the dragging and dropping, in order to create more complex states. Figure 4.8 shows examples of proposed states and the GDialogXML code generated by the DCMA when defining the database access function. In this case, there is a database access function called *VerifyAccountByIdentifier*, which receives two input arguments (i.e. the *AuthenticationCode* and *AccountIdentifier*), then the system automatically creates a new state proposal called *state_VerifyAccountByIdentifier* including two slots.

4.4.2.4 Empty state and already created states

The first one allows the creation of a new empty state, with no defined slots inside, that the designer can define completely afterwards. This way, we allow a top-down design. The second one allows the designer to re-use already defined states to create new states. In this case, the slots are copied but the name of the state should be different to avoid confusions. The assistant does not deny the possibility of using the same name but the optimal solution would be to make a new connection to the existing state using the GUI.

4.4.3 Automatic Unification of Slots for Mixed-Initiative Dialogues

This acceleration helps the designer to decide when two or more slots are good candidates to be requested at the same time (using mixed-initiative forms) or one by one (using directed forms) only when mixed-initiative is not advisable. This is a feature we offer and distinguish our platform from others, since in other platforms they leave the decision up to the designer. Since this functionality relies on using heuristic information it is only available when the system has access to the database contents and when the slots in a given state have been related to a table and field in the backend database.

In this case, the assistant uses the average length, the vocabulary size, the proportion of different values, and the field type as main heuristics obtained for the candidate fields (section 4.1, page 86) and applies a set of customizable rules to decide which slots can be unified and which ones cannot. The existing rules have been created taking into account that, at present, the only modality that actually needs the definition of mixed-initiative slots, in our platform, is the speech modality (i.e., in the Web modality the final user can fill all the slots using just one form).

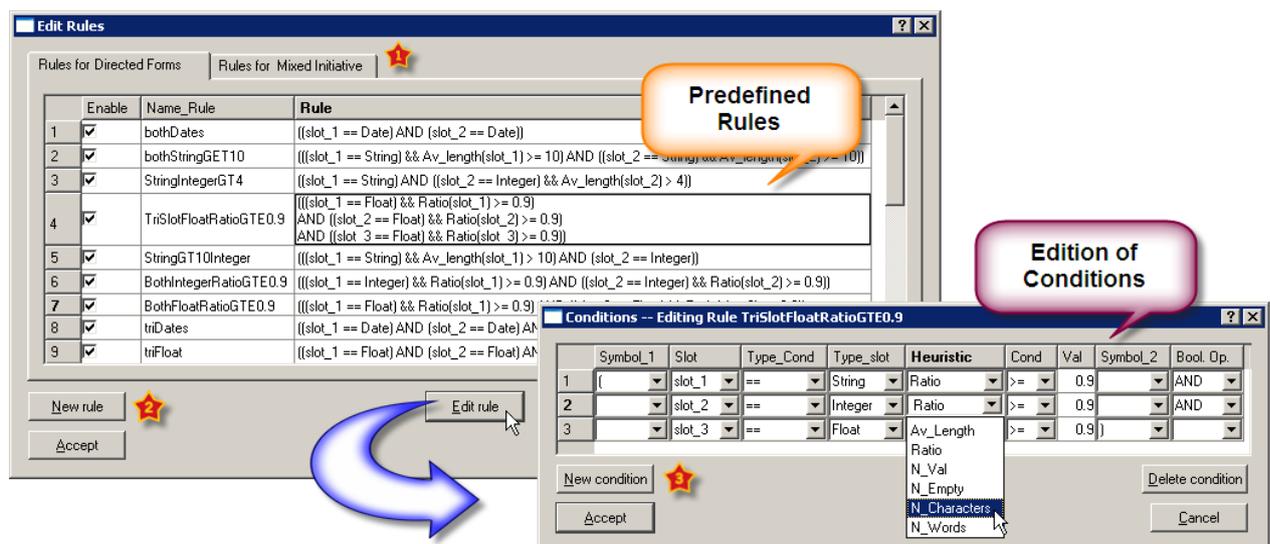


Figure 4.9 Configuration window for creating or editing rules for automatic detection of directed or mixed-initiative dialogues

According to the predefined rules included in the platform, although it is configurable by the designer (using the wizard shown in Figure 4.9), the system does not propose the unification using mixed-initiative dialogues when:

1. There are two slots defined as strings and the sum of the average length of both is longer than 30 characters. In this case, the system tries to avoid the recognition of very long sentences
2. One of the slots is defined as a string with an average length greater than 10 characters, and the other slot is an integer/float number greater than 4 digits. In this case, the rule tries to avoid the recognition of long strings, e.g. an address or name, plus long numeric quantities, e.g. phone or social security numbers, etc., in the same sentence, which again is very likely to fail.
3. There are two numeric slots with a proportion of different fields for a given attribute which is close to one, and the vocabulary size of both fields is high (configurable value). Again, there is a high probability of misrecognition.

Therefore, in all three cases, the system decides that it is better to ask one slot at a time using directed dialogues. The configuration window allows, in any case, the edition of the predefined rules and the creation of new rules (number 2 in Figure 4.9) and conditions (number 3). In addition, it is also possible to create rules for detecting directed dialogues using the wizard (number 1).

If there is a conflict between two or more rules (i.e. one rule proposing MI unification and another one proposing direct dialogues), the system will apply the directed strategy since this is the default choice in VoiceXML. In any case, in spite of the system proposal, after applying the previously mentioned rules, the designer can modify the decision allowing two or more slots to be unified, or not, as mixed-initiative.

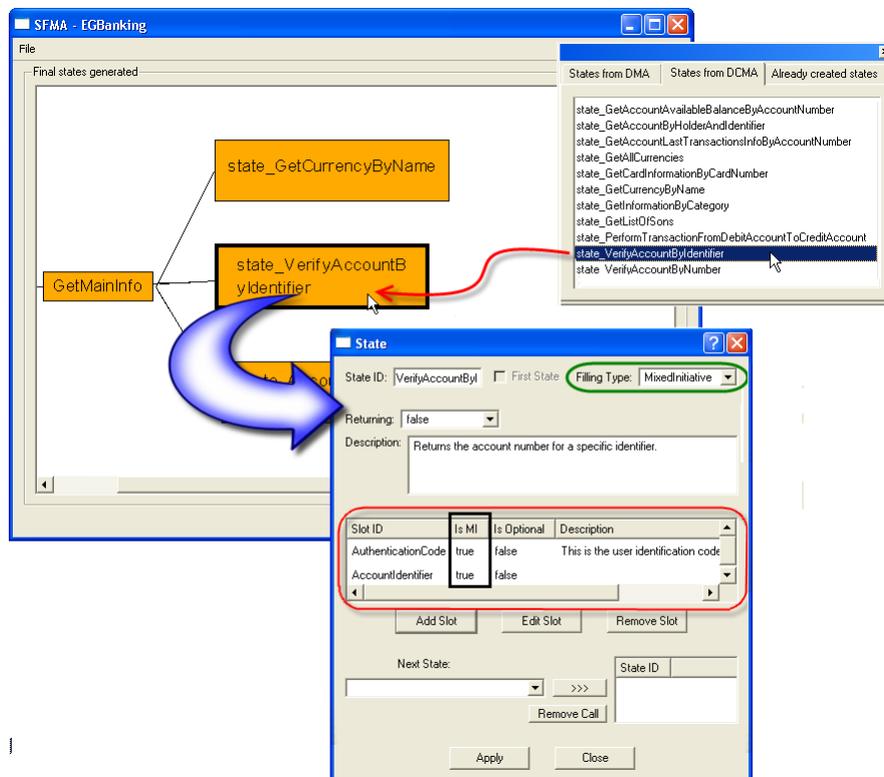


Figure 4.10. Example and GDialogXML code for two slots automatically unified for mixed-initiative

Figure 4.10 shows an example of automatic unification for a proposed DCMA state called *VerifyAccountByIdentifier*. Observe that the GDialogXML code for the corresponding function in the DCMA contains information about the tables and fields used to define the input arguments for that function. According to the heuristics information for those fields, the *AuthenticationCode* corresponds to an integer field with an average length of 4 numbers, and the *AccountIdentifier* field corresponds to a string field with an average length of 9 characters. In this case, the assistant proposes to unify both slots as mixed-initiative, setting the flag *Is MI* to true and the filling type to Mixed-Initiative (the other filling type value is “system initiative”).

4.5 Strategies Applied to the Retrieval Model Assistant (RMA)

As described in section 3.3.2 (page 63), this assistant is used to specify in detail all the information and actions (e.g., variables, loops, conditions, math or string operations, calls to subroutines and dialogues to provide/obtain information to/from the user, etc.) to be done in each state previously defined in the previous assistant, the SFMA, and optionally in new states. Therefore, this assistant provides the most complete functionality for dialogue design in the platform. For that reason, we made a strong effort on including several accelerations in this assistant.

In summary, the assistant allows the following accelerations: automatic generation of several dialogues that the designer can drag and drop in the different windows that make up the assistant, to obtain information from the user (dialogues with prefix DGet) and to provide information to the user (dialogues with prefix DSay); the automatic generation of relevant action proposals according to the dialogue being edited at each time; the automatic passing of arguments when connecting different actions and dialogues; and, finally, the possibility of creating complex dialogues using Mixed Initiative and Over-answering capabilities, among other accelerations that are described in detail below and published in [D’Haro et al, 2006], [D’Haro et al, 2004a], and [D’Haro et al, 2004b].

4.5.1 Automatically Proposed Dialogues

When the RMA is started, it analyses the information from the data model and the database access function looking for all attributes defined as atomic types to automatically generate dialogues to obtain information from the user (called DGet) and dialogues to provide information to the user (called DSay). These dialogues include a GDialogXML property that allows the Modality Extension Assistant (see Sections 3.4.2 and 3.4.3, page 65) to identify them from other dialogues, and to know when the designer has to specify for the DSay dialogues the prompt/output concepts to be presented to the user (for the speech/Web modality respectively), and for the DGet dialogues the grammar/input concepts used by the recognizer/Web generator and the confirmation strategies.

In general, the assistant creates different types of DSay/DGet dialogues depending on the type of the input parameters and the class or database function used to generate them. For instance, dialogues with the mask `DGet/DSay_ATTR_attribute-name_IN_CLASS_class-name` (see Figure 4.11) are created from the data model structure and require only a matching with one input atomic parameter (i.e., the calling dialogue has to pass as argument to the DGet/DSay dialogue one atomic variable). In this case, these dialogues may be useful to obtain/provide information to the user before/after calling a database function that receives/returns an atomic parameter. On the other hand, dialogues with the mask `DSay_ATTR_attribute-name_GIVEN_CLASS_class-name` require a matching with one input

complex parameter (i.e. a reference to an object or the object itself) in this case to a specific class in the data model. In this case, the database function returns a complex object. Observe that the platform does not offer DGet dialogues in this case, since it is expected that DGet dialogues only fulfil atomic parameters, not complex ones (i.e., DGet dialogues ask users for strings, integers, floats, etc. data, not complex objects). Finally, the mask DSay_return-variable_FROM_databaseFunction-name is used to identify DSay dialogues created from the atomic returning variables for all the database access functions defined in the DCMA. In this case, the designer can use these dialogues for providing atomic data after retrieving the information from the database.

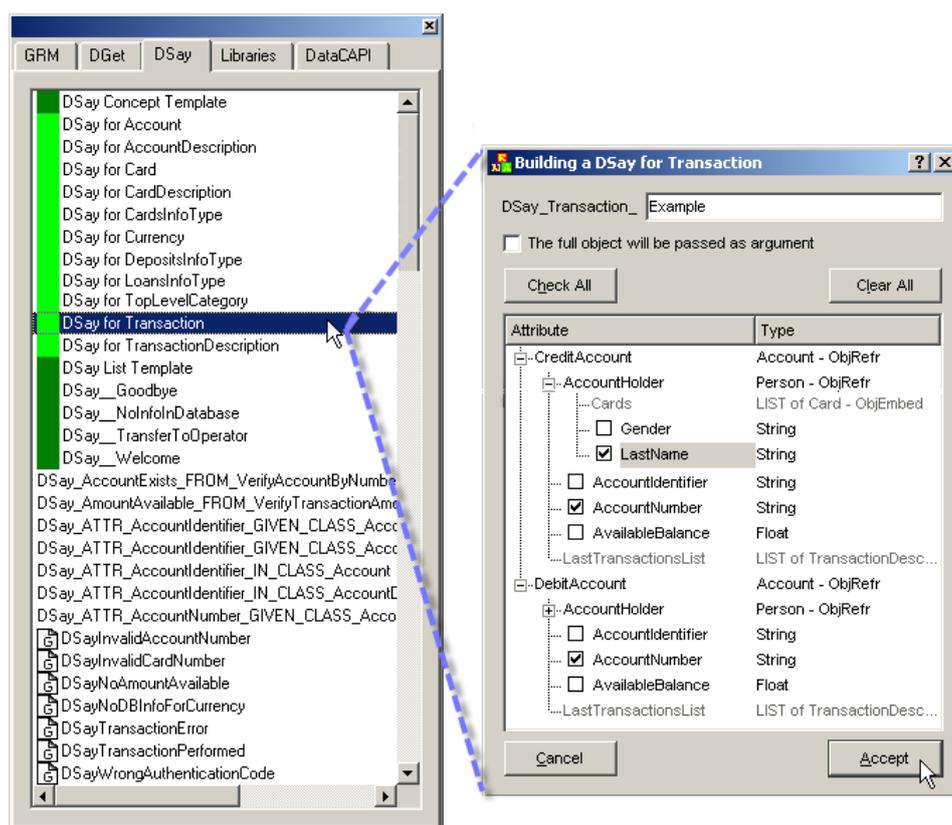


Figure 4.11. Auxiliary screen of the RMA and popup window for dialogue configuration.

Although all the previous automatic DGet/DSay dialogues are enough for most dialogue applications, the algorithm suffers of some limitations. For instance, if the attributes in the data model classes are complex or include object inheritance, the assistant is not able to generate automatic dialogues for them. Besides, these dialogues cannot be merged to create more complex ones. In order to solve these problems, the assistant provides configurable DSay dialogues using a template (dialogues with prefix “DSay for” in Figure 4.11,) that shows the class and its attributes, expanding the complex attributes (with inheritance and objects) and allowing the designer to select any of these attributes to be used when designing the prompt in the modality extension assistant (MEA).

Finally, other DSay dialogue templates are also available, for instance: a) A generic DSay template to provide concepts (DSay Concept Template), which are useful for providing generic information to the user without querying the database or asking any information to the final users. b) Configurable DSay to present variables from a dialogue, in case that the

variable has not any dependence with the data model or it has not been returned from a database access. c) A generic DSay template to present lists of objects, specifically designed for handling lists of results after querying the database (see section 4.6.1, page 110), and d) Predefined DSay such as: Welcome, Goodbye, Transfer to operator, etc. In addition, dialogues from loaded libraries and database access functions can also be used.

Figure 4.11 shows all the dialogues mentioned above, with an example of the configurable template for the Transaction class called ‘DSay for Transaction’ where several attributes have been selected and will be provided to the user in the real-time system. The flexibility of this template lets the designer select attributes from the different child classes of the Transaction class (e.g., *AccountNumber*), complex attributes coming from inherited classes and contained in another class (e.g., *LastName* from class *AccountHolder* included in class *CreditAccount*). When the dialogue definition is over, it is added to the list of dialogues tab, so that it can be used later on.

4.5.2 Automatic Generation of Action Proposals in Each State

This is one of the most important accelerations included in this assistant. The main motivation for this strategy was to include in one popup window, called “SFM proposals” all the actions that the designer could require, or at least with many chances to be used, to complete the definition of all the dialogues previously defined in the SFMA. In general, typical actions for a dialogue are, first, to request some information from the user (i.e., the slots for that state), then to access the database through a call to a database access function, then to provide the results of the database query, and finally to jump to the next dialogue. Considering that these actions are the most common ones, we decided to implement different mechanisms for including automatically proposals for each case, as explained below.

To decide which actions are relevant, all the information already defined in previous assistants, especially the SFMA, is analyzed using the following strategies for each of the four sections in the window from Figure 4.12:

- Slots asked in the current state, the transitions, and the corresponding slots in those destination states. In this case, the strategy is to use directly the information from the SFMA. In the figure, the system shows the current slot for the edited dialogue (i.e., *currencyName*) and for the following dialogues (e.g., *slotYesOrNo*). Besides, the system shows the calls to following dialogues with up to two levels in depth.
- State specific DGets: to select them, the system looks for the slots defined in the SFMA; if they are related to the Data Model, the system selects the corresponding dialogues automatically; if not, a more relaxed criterion is used, which is to look for a match in the name or attribute type.
- Database access functions: to filter the possible functions already defined in the DCMA, the system first considers functions with the same number and type of input parameters as the defined slots for the current dialogue. The next criterion is as follows: if the input parameter includes a reference to the data model there should be a match in class and attribute between slot and parameter; if not, they should match in type. If no function passes these filters, a more relaxed filter is applied (e.g., similarity between names). If even with the relaxed filter there is no function to be proposed in this window, it would probably mean that there is no database access function suitable for that state and it should have been defined

before. The assistant offers the possibility of creating those functions, and then reload the information in this window. Finally, the other possibility is that for the current dialogue it is not necessary to access the database since that will be done in another dialogue.

- State specific DSays: they are selected in a similar way to DGet dialogues, but we also include DSays specific to the values returned by the database functions selected in the previous step.

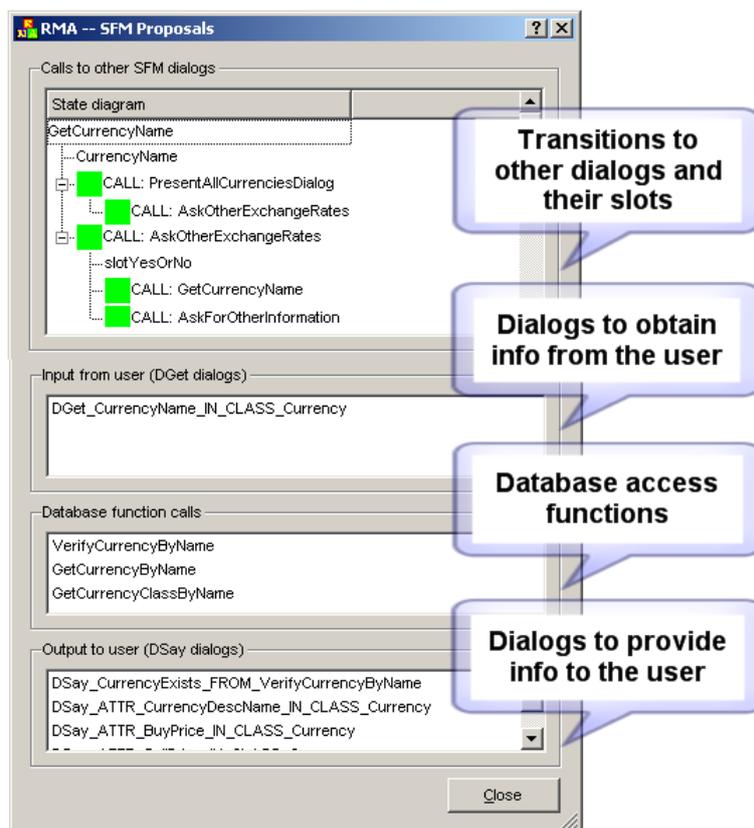


Figure 4.12. Example with automatic dialogues and database access function proposals

Figure 4.12 shows an example of the proposals for the banking application. In this example, the designer is editing a dialogue where given a currency name the system provides its specific information (buy and sell price, general information, etc.). Using the proposal window, all the designer would need to do is to select the corresponding DGet in the window (`DGet_ATTR_CurrencyName_IN_CLASS_Currency`), then the database access function `GetCurrencyByName`, and finally the DSays that provide the desired attributes from the currency. To finish, the designer would drop the call to the next state (e.g., `AskOtherExchangeRates`).

As we can see, this is one of the most useful accelerations, as most common actions that are needed in most of the dialogues can be accessed just dragging and dropping the ones proposed in this window.

4.5.3 Automated Passing of Arguments between Actions

This is a critical aspect of dialogue applications design. Several actions and states have to be ‘connected’ as they use the information from the preceding dialogues. In general, most current design platforms allow the same kind of functionality, offering the user a selectable list of all the available variables in the dialogue. In other cases, especially considering the connections with database access functions, some platforms only allow the designer to define the matching by modifying by hand the script code. In this acceleration, we have tried to go one-step beyond by automating the connection through automatic proposals. In this case, the assistant detects the input/output variables required in each action and, using a popup window, it offers the most suitable already defined variable of a compatible type; if there are more than one variable of a compatible type, the assistant sorts them according to the name similarity between variable and dialogue. If there is no a compatible variable already defined in the system or the name proposed by the assistant is not desired, a new local or global variable can be created in the same window. Moreover, if the designer makes a mistake or needs to edit the matching made in the previous steps, the assistant provides a window where all this matching can be edited.

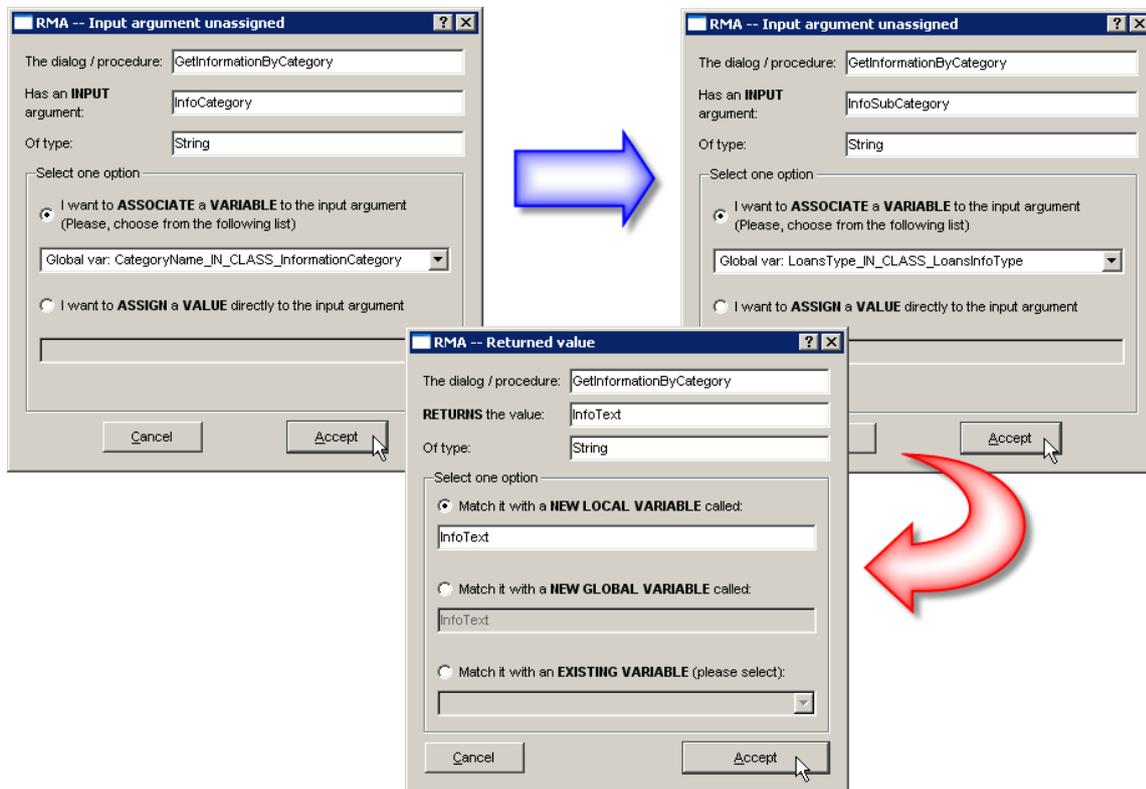


Figure 4.13. Form fill-in windows that automate the process of passing arguments between actions

Figure 4.13 shows an example of this acceleration for the *GetInformationByCategory* function defined in the data connector model. This function requires two input arguments: *InfoCategory* (e.g. loans, cards, deposits, etc.) and *InfoSubCategory* (e.g. car loans, house loans, visa, invest savings, etc.) and returns the information to be presented to the final user. The form fill-in windows allows the designer to associate the input arguments as local variables already defined and to assign the result to a new local string variable called *InfoText*. In all the cases, the system automatically proposes the values and options presented

in the forms. In this way, the designer only needs to click the accept button and continue with the design.

4.5.4 Mixed-Initiative and Over-Answering

This acceleration allows the creation of complex dialogues where the system can ask for several slots at the same time or the user can answer with optional information. Even though these two functionalities might be considered as speech modality dependent or unnecessary for the Web modality, therefore they should not be handled at this stage (i.e. the RMA is modality and language independent), we preferred to include them here for two main reasons: first, because some other modality that can be included in the future might benefit from this functionality too, and, second, we could make it possible that in a Web page the user does not have to fill in all required fields at the same time. In this case, the Web system would detect that a certain slot is missing and, instead of generating an error, it would ask for the missing data in a subsequent form, in a similar way as VoiceXML handles mixed-initiative with several slots.

As we have mentioned at the beginning of this chapter, this acceleration is one of the most important speed up strategies applied to the platform. The main motivation for this acceleration was to overcome some of the main limitations we found in current design platforms. In first place, this acceleration allows the quick creation of dialogues with mixed-initiative that are difficult to define in many platforms since they do not offer a similar procedure for creating them, leaving the designer, in most cases, only the possibility of creating directed dialogue forms (i.e., where only one slot can be asked at each time). Second, if we consider the platforms that allow the definition of mixed initiative dialogues, we found, as far as we know, that none of these platforms allows the creation of dialogues with over-answering capabilities or a combination of mixed-initiative with over-answering. This is mainly due to limitations of the VoiceXML language that we overcame during the GEMINI project. Finally, as an additional improvement, the use of this acceleration allows designers to create a better dialogue flow since the assistant automatically proposes the slots that can be asked using mixed initiative based on heuristic information (see section 4.4.3, page 99). This way, the system tries to avoid requesting complex or too confusable slots to the final users. In addition, since the internal code and flow is automatically generated it reduces designer mistakes in the design or in the codification of the information.

To provide this functionality, the system offers a Mixed-initiative Template that the designer can drag and drop over the dialogue that is being edited. The template shows available slots that can be selected (by default, the ones specified in the SFMA for the current dialogue). Moreover, the template gives the possibility of adding optional slots to be used for over-answering at the same time. With this information, the system generates the necessary programming code (including calls and automatic dialogues) in GDialogXML syntax that controls the mixed-initiative handling. It is important to mention that the author of the thesis directly contributed during the GEMINI project to the definition of the mixed-initiative and over-answering templates using as reference the VoiceXML specification and the possibilities of our own XML syntax.

Figure 4.14 shows an example of a mixed-initiative dialogue created using the template. In number 1, the slots to be asked are declared. Number 2 defines the procedure for asking several slots at the same time; numbers 3 and 4 handle the situation in which the user answers partially or only some of the slots are filled after the recognition, so the system has to ask again for unsolved slots.

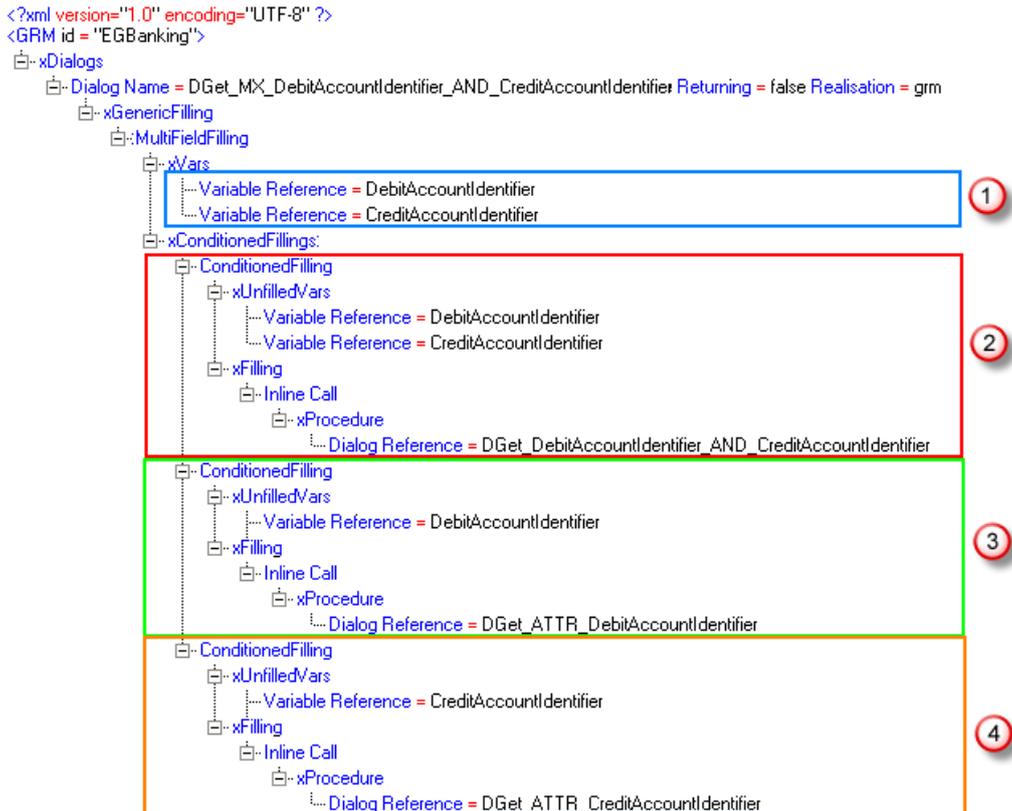


Figure 4.14. Example of the GDialogXML syntax for a mixed-initiative dialogue created in the RMA

Figure 4.15 shows, for the bank transfer example, the process followed to create a dialogue where two slots, *CreditAccountIdentifier* and *CreditDebitIdentifier*, are asked using mixed-initiative. The designer just needs to drag and drop the Mixed-initiative Template (the selected item marked as number 1) on the main window of the dialogue (identified with number two); then, a popup window appears (number three) where the designer selects the desired slots (by default, the ones selected in the SFMA, *CreditAccountIdentifier* and *DebitAccountIdentifier*) and presses Accept with the possibility to add optional slots (over-answering) or not.

To admit over-answering, the procedure is very similar: when the designer drops any DGet (action to obtain data from the user) the system automatically offers to select additional slots as over-answering from that specific state and from the following states in the flow (with a limit of two in the hierarchy). As default, the slots defined as optional in the SFMA (see section 3.3.1, page 62) are automatically converted into over-answering slots here. In the runtime system, the behaviour is that before any DGet the system checks whether the data to be asked has been already obtained in a previous state in the flow (as would be the case with over-answering). To help in this checking, in the final script all slots are declared as global variables, so they can be accessed from any state.

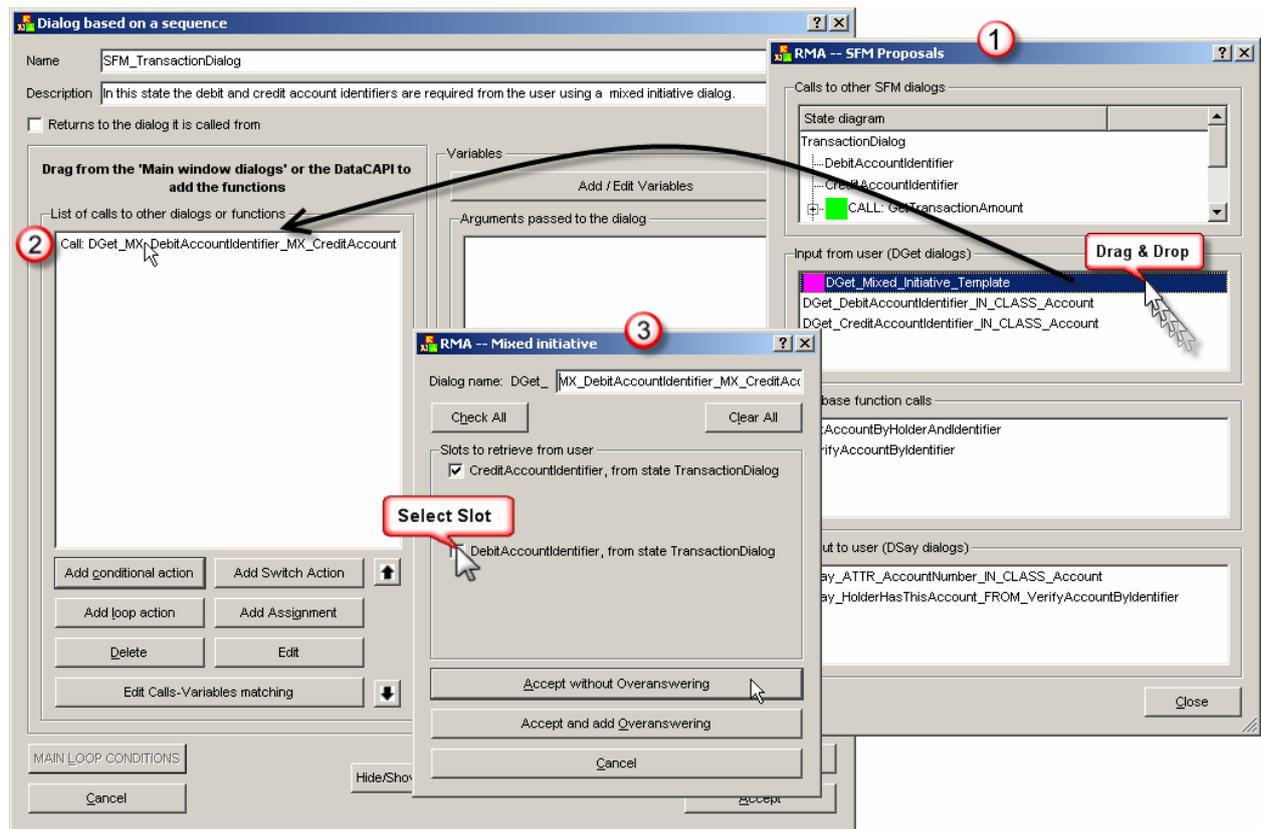


Figure 4.15. Example of the creation of a mixed-initiative dialogue

Similarly to the mixed-initiative case explained above, the system generates the necessary programming code (including calls and automatic dialogues) in GDialogXML syntax for handling the over-answering.

```

<?xml version="1.0" encoding="UTF-8" ?>
<GRM id = "EGBanking">
  <x:Dialogs
    <- Dialog Name = DGet_ATTR_InformationType_OVR_HouseLoansType Returning = true Realisation = grm
      <- x:InputFieldVars
        <- Variable Reference = LoansType
      <- x:InputFieldVarsOpt
        <- Variable Reference = HouseLoansType
      <- x:GenericFilling
        <- MultiFieldFilling
          <- x:Vars
            <- Variable Reference = LoansType
            <- Variable Reference = HouseLoansType
          <- x:VarsOpt
            <- Variable Reference = HouseLoansType
          <- x:ConditionedFillings:
            <- ConditionedFilling
              <- x:UnfilledVars
                <- Variable Reference = LoansType
              <- x:Filling
                <- Call Name = DGet_LoansType_AND_OVR_HouseLoansType Returning = true
                  <- x:Procedure
                    <- Dialog Reference = DGet_LoansType_AND_OVR_HouseLoansType
                <- x:Help
                  <- TextConcept Reference = HelpFor_Loans
  
```

Figure 4.16. GDialogXML code for a dialogue to ask for a single slot and define another one as optional using over-answering

Observe in Figure 4.16 that in this case the code includes the name of the compulsory slot (number 1 in the figure) and the optional one (number 2). In order to allow the user to answer both slots at the same time, the template defines both slots as target variables (in number 3) but remarking the optional variable in number 4. Then, in number 5, the template defines the condition for repeating the same query until the compulsory slot is filled by the user. Number 6 specifies the DGet dialogue to ask the compulsory and optional slots. Finally, number 7 specifies the concept (i.e., language independent) used for providing help to the user in case of problems or if the user requests it. Then, in the MEA assistant the prompt associated to this concept is defined, see section 3.4.3 (page 66).

4.5.5 Other Functionalities

Besides all the strategies mentioned herein, we have also included in the GUI some useful characteristics as hotkeys for accessing the most common functionalities of the assistant, different colours for distinguishing each kind of dialogue (i.e. already filled, empty, DSay or DGet dialogues, etc.). Besides, in order to reduce the number of dialogues shown in the canvas the designer can switch between a basic presentation of the dialogue or a more detailed visual/textual flow (i.e., including internal information about variables, dialogues that are called from or call to the current one, type, etc.)

There is also a method to display the contents of complex or nested actions contained in a dialogue using tooltips (see Figure 4.17), which help the designer in their interpretation avoiding the need to open or edit them.

Finally, several other GUI actions such as find/create/delete/edit a dialogue, or zoom in/out the workspace are also available, as well as other contextual right click commands.

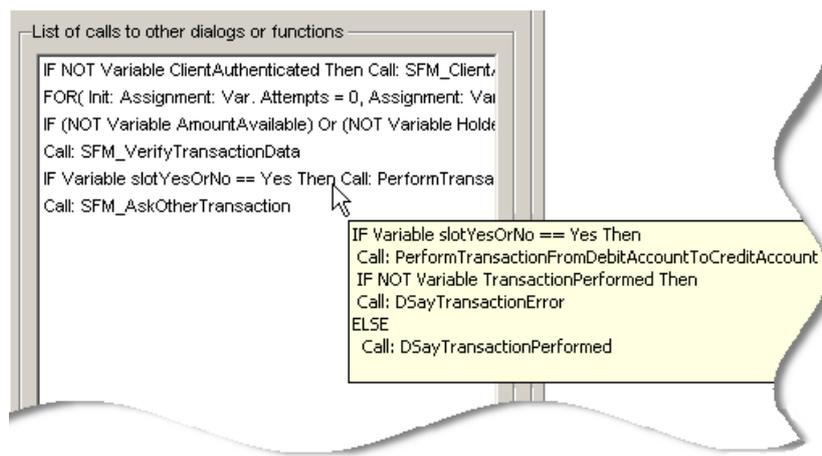


Figure 4.17. Tooltips functionality for a quick description of all internal actions

4.6 Strategies Applied to the Modality Extension Retrieval Assistant for Speech (MERA-Speech)

As we have mentioned in the introduction of this chapter, the accelerations introduced in this assistant can be considered as the second most important strategies applied to the platform. In this assistant, we considered solutions for two specific problems for the speech modality: the presentation of results to the user after accessing the database, and the confirmation of user answers. When we studied the mechanisms offered by current both

commercial and research design platforms to deal with these problems, we found that the typical solution was to leave the designer to specify the complete dialogue flow or to leave the problem to some predefined actions provided by the ASR engine. Obviously, these solutions are not satisfactory since they imply the codification by hand of too many situations and conditions. Besides, as we will see in section 4.6.2 (page 113), there are some restrictions for some confirmations that the designer could not take into account. On the other hand, the assistant also accelerates the design by providing automatic proposals for the different data that the designer has to provide, automatically generating all the dialogue flow according to the designer selections, and using predefined and reusable built-in dialogues. Finally, another important contribution of the thesis are the templates that we have defined and used to codify all the internal dialogue flow and actions required to solve each of the problems and situations considered (see Appendix C). These templates were created from our experience in designing dialogue applications, considering the common solutions to the problems we have dealt in this assistant, as well as taking into account the limitations that we found for the VoiceXML standard.

As explained in section 3.4.2 (page 65), this assistant allows the designer to adapt specific dialogues defined in the RMA to the speech modality. The dialogues considered by the assistant are those that show lists of information to the user (e.g., mainly the ones created using the DSay template for list of objects and marked with prefix DSay_From_List, see section 4.5.1, page 101) and those that obtain information from the user (e.g., the ones created from the automatic dialogues in the RMA and marked with prefix DGet, see section 4.5.1).

For the DSay dialogues, the assistant allows to specify the dialogue flow for providing the information contained in a list of retrieved results after making a database access. The flow depends on the size of the list. Four cases have been considered: when there is not any retrieved result, when the list has only one item, when the number of items lies on a defined range, or when there are too many items, so it is difficult to say all of them using speech.

On the other hand, for the DGet dialogues, the assistant automatically generates the flow for confirmation handling (i.e. what to do when the user does not provide an answer after a system query, to allow an implicit confirmation, or when the confidence level is in the range of the explicit confirmation, etc.).

Finally, since the algorithms used to automatically create the flow for the dialogues completed in this assistant are too complex to be described in detail in this section, we have decided to include an easy to read description in pseudo-code in Appendix C. The appendix describes the cases considered in this assistant: a) handling of lists of objects, b) simple confirmation and full confirmation for dialogues with one slot, c) confirmation of dialogues with mixed-initiative, d) confirmation handling for dialogues with one compulsory slot plus slots with over-answering, and e) the most complex case, confirmation handling for dialogues with mixed-initiative and over-answering slots.

4.6.1 Presentation of Object Lists

Object lists are the result of a database query, so there is usually a lot of information to be provided to the user. The assistant considers four different cases as a function of the number of items in the list. For each case, a simple form allows the designer to specify the actions that have to be carried out. After filling the four forms, the actions and new dialogues needed to provide/obtain information to/from the user are automatically generated.

In order to accelerate the process of filling in the forms, the assistant provides the most reasonable default values for all dialogue and slot names after the analysis of the input files of the assistant (in this case, the models generated by all the previous assistants in the platform: ADA, DMA, DCMA, SFMA, RMA, and UMA). The assistant also considered a simplified case: when the list only depends on one slot input by the user, e.g., when asking for a list of banking transactions. In this case, the assistant presents a simplified version of the following windows where the designer does not need to specify the slots to be cleared. The four different cases and their actions are as follows:

1. **The list is empty:** As the query has been too restrictive, some slots need to be unset and a new query has to be done with less restrictive values. The first action is to use a predefined DSay dialogue to tell the user that there is no available information and then to jump back to a previous state selected by the designer, where the user is asked again for the slots that the designer decides that have to be unset. The objective is that the user answers the next time with a more generic answer to the slot so that the query is less restrictive.
2. **The list has one item:** This is the simplest case, since there is only one item, the designer only needs to define a configurable DSay that can provide complete or partial info from the item found. In order to configure the DSay dialogue, a pop-up window shows a list with the available attributes for the class of the item.

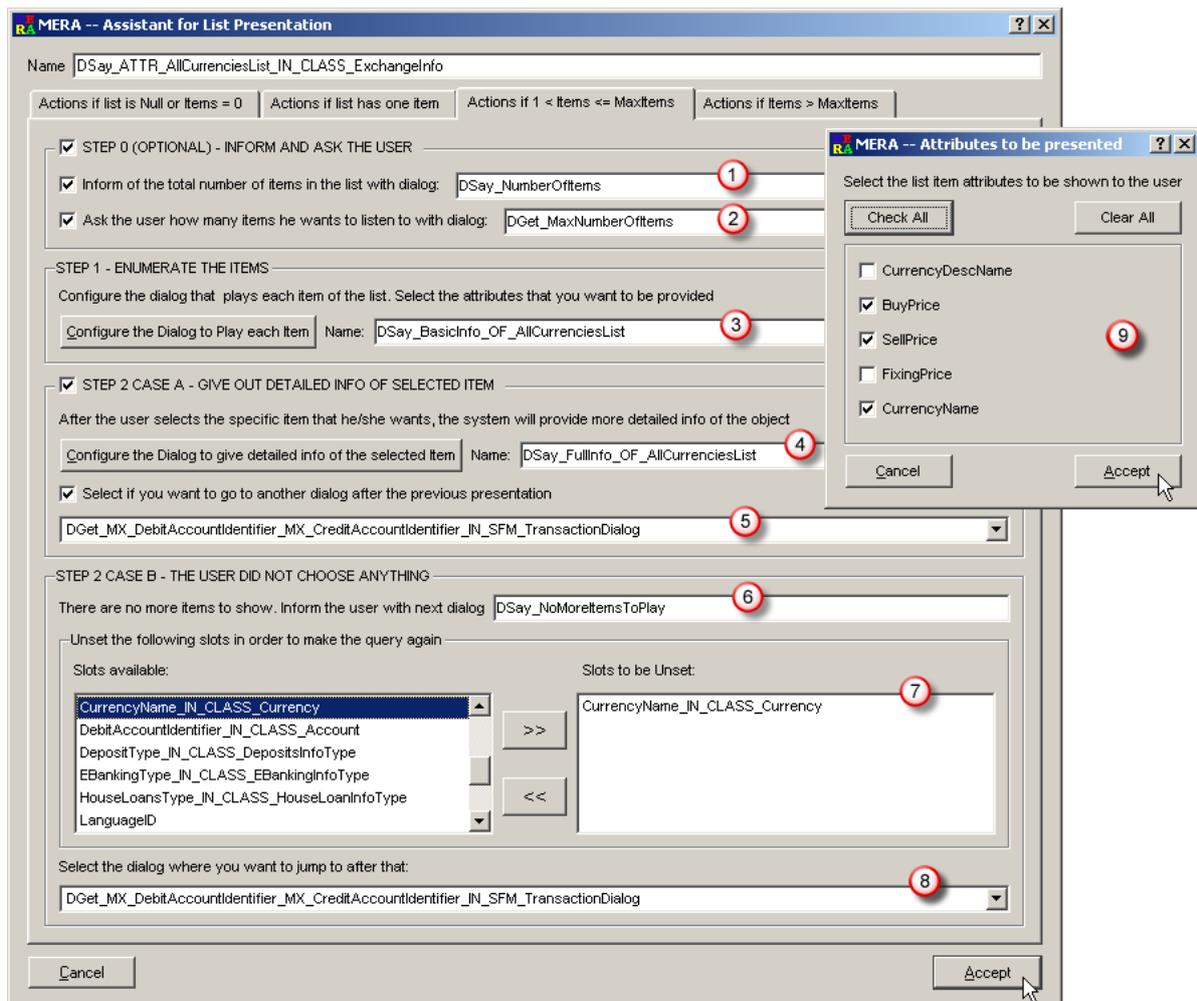


Figure 4.18. Example of the assistant window for configuring a DSay dialogue for the presentation of objects lists (case 3).

3. **More than one item and less than a maximum allowed:** This is the more complex situation, as the items have to be provided in groups. Figure 4.18 shows the window used to define this case. In the figure, number 1 and 2 define an optional step for informing the user, at the beginning of this dialogue, how many items there are in the list (using the built-in dialogue `DSay_NumberOfItems`), and to ask how many the user wants to listen to (`DGet_MaxNumberOfItems`). If the user does not provide a valid number (e.g., it is too high) the system uses the default number defined in the ADA (see section 3.2.1, page 59). As acceleration, the assistant automatically proposes default dialogues that can also be reused in other `DSay` dialogues for presentation of lists of objects.

The next step (number 3) is to configure the enumeration of the retrieved items. The pop up window, number 9, allows the designer to select the attributes to be used to configure the prompt that provides general info to the user. In this case, it is expected to select few of them and provide more information in a following step. After playing the info for each item in the group, the user is asked if he/she wants to continue to the next group, repeat the group, begin from scratch, exit, or select a specific item to receive more detailed information. In this case, the system uses universal command words as proposed by the Universal Speech Interface project [Toth et al, 2002].

In case that the user chooses one item, the system provides detailed information (number 4) of the object using a new selection of attributes (made through the pop-up window in number 9) specified by the designer (similar to the case when the list has one item).

The checkbox, number 5, allows the designer to select an optional jump to another dialogue after the presentation. By default, the system proposes the dialogue that calls the `DSay` of Lists or the initial dialogue of the service; the designer can also select a `DYesOrNo` dialogue to know if the user wants another service or not.

Another situation (step 2 case B in the figure, near number 6) that the assistant faces is when the system finishes reading the whole list and the user does not like any item or has cancelled before the end of the list. In this case, the system informs the user that there are not more items to show (the assistant proposes the dialogue `DSay_NoMoreItemsToPlay`, number 6) and then jumps back to a previous state selected by the designer (number 8), where the user is asked again for the slots that the designer has decided to unset (number 7), so that the user can answer with a different input that provides different database items. This situation is similar to case 1.

4. **More items than the maximum allowed:** As there are too many items, the search should be more restrictive. In a first step, the system uses a predefined `DSay` dialogue to inform the user about this situation. For the next steps, the assistant can handle three different situations:

First, the designer can choose that the system plays information for items from one to a maximum as in case 3.

Second, if all slots of the application are already filled, the user has to change some of them to make them more restrictive (e.g., the user wants last month's transactions but there are too many). In this case, the designer specifies which slots have to be unset (e.g., the slot containing the period of time the user wants to know) and the questions will be repeated, in a similar way as in case 1.

Finally, the third situation, if there are still some slots to be asked, the system continues with the normal dialogue flow until the next database query.

4.6.2 Confirmation Handling

One of the main problems in a dialogue system is how to cope with speech recognition errors. Unfortunately, the great variability of speech among different speakers, environments, channels, noises, etc., prevents modelling all possible variations. Therefore, the system will always need to deal with speech recognition errors, requiring the confirmation of speech recognition results by asking the user, in order to proceed in the dialogue flow (i.e., especially before retrieving/modifying any information from/in the database). The problem is that if the system confirms every single slot, the dialogue will be too slow and user satisfaction will decrease drastically. A solution of compromise is to use the confidence level provided by the speech recognizer. The confidence is a value between 0 and 1 that provides a measure of reliability of the speech recognition results: a value close to 0 means that results are not reliable, and a value close to 1 means that results are very reliable. Usually, three thresholds, τ_i , can be defined for this confidence value and several strategies can be adopted according to these thresholds [San-Segundo et al, 2001b]. The most popular strategy and the one implemented in our platform, being CV the confidence value is the following:

- No confirmation: $\tau_1 < CV \leq 1.0$. The result is accepted with no confirmation because the confidence is very high.
- Implicit confirmation: $\tau_2 < CV \leq \tau_1$. High confidence in the result. An “implicit confirmation” is applied, which means that the system provides the recognition result to the user as a fact in the next dialogue turn, speeding up the dialogue, e.g. “You want to travel to London. When do you intend to leave?” The user can say “no” or “cancel” at that point to go back in the dialogue if London was not the intended destination.
- Explicit confirmation: $\tau_3 < CV \leq \tau_2$. The confidence is intermediate. The best option is to ask the user if the result is correct, e.g. “Do you mean London?” to confirm that London is the intended destination.
- Reject: $0.0 \leq CV \leq \tau_3$. The confidence level is extremely low, so the result is clearly unreliable. The system rejects it and asks it again.

In our platform, the default confidence levels for all the dialogues are defined in the Application Description Assistant (ADA) at the beginning of the design. However, they can be modified according to different user profiles and depending on the dialogue using the User Modelling Assistant (UMA). In the MERA-Speech assistant, it is also possible to define which of the strategies described above are available or not depending on the complexity and subsequent actions given a particular dialogue. In order to do this, we have considered two confirmation profiles: Simple and Complete. Simple is recommended for dialogues that need a very high confidence, such as Yes/No or passwords questions; in this case, only two levels are allowed: no confirmation and repeat the question. However, Complete uses all the strategies described above.

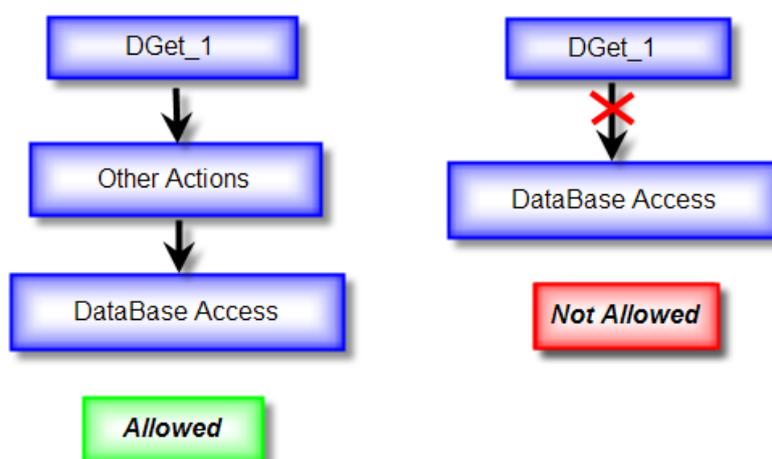
Initially, when the MERA-Speech assistant is initialized, it automatically selects, from the file generated by the RMA, all the dialogues that are used as input dialogues (e.g., the ones with prefix DGet) that need confirmation and analyzes their flow to propose the most suitable confirmation profile (Simple or Complete). However, the designer can change that proposal and the assistant checks whether the selected type is feasible. Then, after accepting

or modifying the proposals of the assistant, the designer just has to press the accept button in order to allow the system to automatically complete the confirmation flow following the algorithm described in Appendix C. Finally, the MERA-Speech uses the same common internal variable used in the UMA assistant (section 3.4.1, page 65) to store the confidence value returned by the last recognition call. Then, the final script tells the system to compare this value with the current confidence limits, stored in four fixed name variables, as defined in the UMA for each user level and specific dialogue.

In detail, the algorithm that analyzes the dialogue is as follows: first, the system examines the number and type of slots to be retrieved by the DGet dialogue, and if there is only one slot, its type is Boolean or string (as used to contain an alphanumeric password) and the number of actions in the calling dialogue is not too high, the system selects the simple profile; if all these conditions are not fulfilled, then the system selects the Complete profile.

When the algorithm allows the system to use the Complete profile, the assistant has to consider three different cases to determine if implicit confirmation can be allowed or not. If the system does not allow the implicit confirmation, explicit confirmation is used regardless of the confidence levels defined in the UMA. If the implicit confirmation is allowed, the assistant automatically sets a global variable with the name of the DGet dialogue where to jump back and unset its slots in case the user rejects the recognition in the following DGet dialogue (remind that the rejection, using implicit confirmation, is detected in the following DGet). The three conditions that have to be fulfilled in order to accept the implicit confirmation are the following ones:

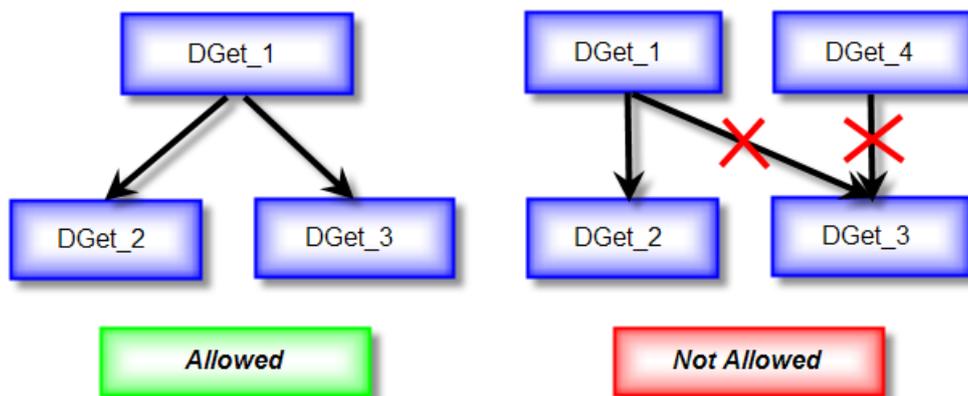
1. If the next action after calling the DGet dialogue does not correspond to a database access. The reason is to allow, in the real time system, the possibility of making the confirmation before accessing the database (i.e., in the following DGet dialogue).



2. When the selected dialogue (DGet_1) calls another dialogue (DGet_2) which in turn is not called from other dialogues (e.g. a DGet_3). The reason is that otherwise the system will not be able to set the global variable which contain the dialogue to jump back (It could be DGet_1 or DGet_3) in case of a rejection. Therefore, without this information the system cannot guarantee to jump back to the right DGet dialogue in order to repeat the last question and fill the slot again.



3. The selected dialogue (DGet_1) calls to different dialogues but they are not called from other places (e.g. if DGet_3 is called from DGet_4). In this case, the problem appears for the DGet_3 dialogue, because from that dialogue the system cannot guarantee to jump back to the right DGet (DGet_1 or DGet_4) as in case 2.



Finally, the assistant automatically generates the dialogue flow (consisting of calls to internal automatically generated dialogues for each type of confirmation and dialogue state) to carry out all the confirmation and subsequent correction. These internal dialogues are named in a smart way (using the Universal Speech Interface (USI) project, [Toth et al, 2002]) so that the designer can easily identify them when defining the grammars and prompts in the next assistant.

4.7 Strategies Applied to Other Assistants

This section describes, for consistency with this chapter, several accelerations applied to other assistants in the platform that were mainly developed during the GEMINI project.

In general, the following assistants and their accelerations were developed by other partners of the project, except when explicitly stated that they were introduced by the author of this thesis.

4.7.1 Modality and Language Extension Assistant (MEA)

As mentioned in section 3.4.3 (page 66), this assistant is the responsible of defining all the language and modality dependent aspects of the application (e.g., grammars and prompts for the speech modality, input/output concepts for the Web modality).

The main accelerations included in this assistant are: a) The possibility of creating language-dependent prompts using pre-loaded libraries and reusing prompts in other languages as configurable templates (see section 4.7.1.1). b) the creation of JSGF grammars using a wizard window, which also allows the creation of the pronunciation vocabulary used by the speech recognizer (see section 4.7.1.2), and c) the possibility of debugging JSGF files and creating n-gram based stochastic speech grammars (see section 4.7.1.3).

4.7.1.1 Setting of system prompts

For the speech modality, several prompts for each different kinds of dialogue (filling (DGet), presentation (DSay), or help) have to be defined: the default one, for the different user levels, and for all possible recognition errors (no input, no match, service timeout, etc.), and the number of times that the assistant allows every error to occur (1, 2, 3, ...) before transferring the call to an operator or exiting.

To speed up the process of typing all these prompts, the assistant offers three possibilities: 1) reuse prompts already available for the current application, 2) reuse prompts generated in previous applications and saved as libraries, or 3) reuse wording libraries saved from previous applications.

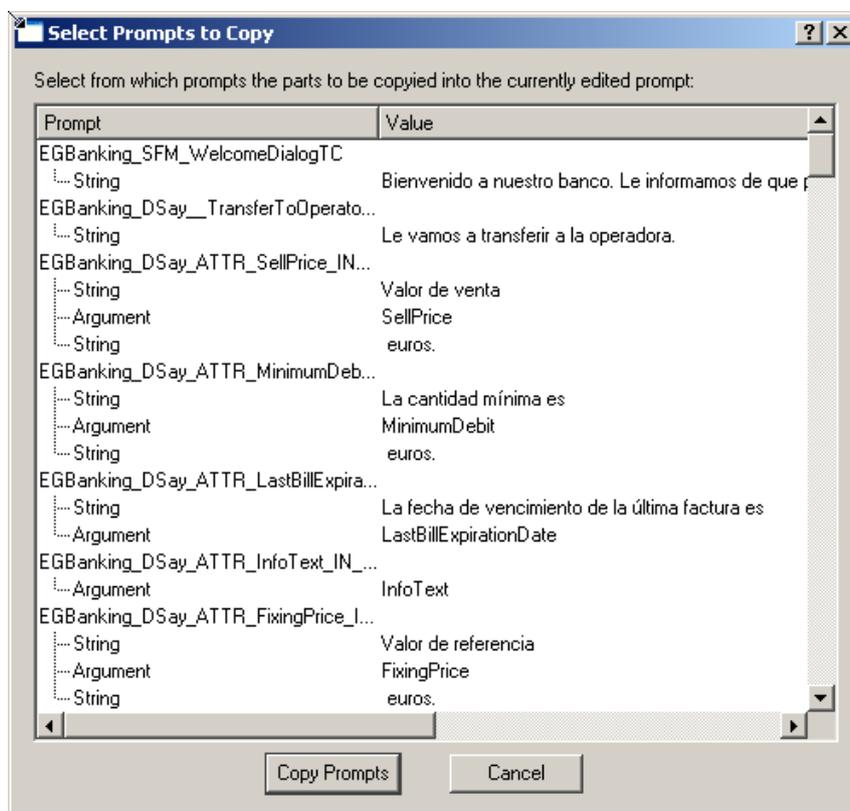
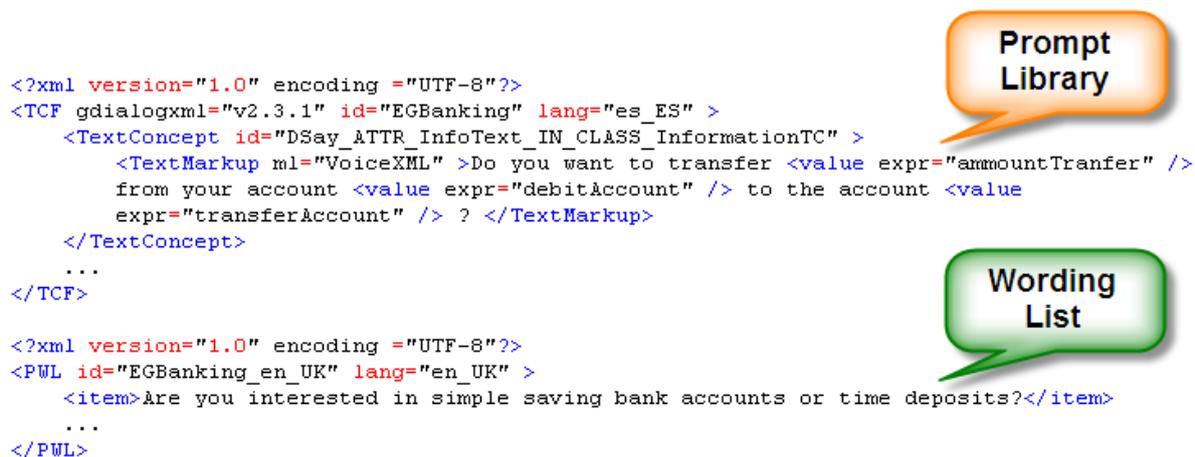


Figure 4.19. Assistant window for copying prompts

For the first and second case, the system allows the designer to copy the contents of an already defined prompt into the currently edited prompt. Figure 4.19 shows the interface for copying prompts when creating a prompt for a specific dialogue. Prompts in this window are loaded from previous saved libraries or from other states of the current service.

For the third case, the system allows the reusability of prompt wordings from old applications. The difference between wording and prompt libraries is very subtle (see Figure 4.20). While a wording only contains full or partial sentences without including prompt arguments, SSML information, or breaks, prompt libraries may contain all this information. The former are useful for cross-domain services, while the latter are useful, especially when they include prompt arguments, for in-domain services or for new versions of the same service. In our platform, in order to distinguish them, wording lists are saved with the extension pwl and the name is automatically composed concatenating the application name and the language (e.g. ApplicationName_LanguageID.pwl). By contrast, prompt libraries are saved one for each language of the service with the name tcf.xml in the corresponding folder.



```
<?xml version="1.0" encoding="UTF-8"?>
<TCF gdialogxml="v2.3.1" id="EGBanking" lang="es_ES" >
  <TextConcept id="DSay_ATTR_InfoText_IN_CLASS_InformationTC" >
    <TextMarkup ml="VoiceXML" >Do you want to transfer <value expr="amountTransfer" />
    from your account <value expr="debitAccount" /> to the account <value
    expr="transferAccount" /> ? </TextMarkup>
  </TextConcept>
  ...
</TCF>

<?xml version="1.0" encoding="UTF-8"?>
<PWL id="EGBanking_en_UK" lang="en_UK" >
  <item>Are you interested in simple saving bank accounts or time deposits?</item>
  ...
</PWL>
```

Figure 4.20. Examples of wording and prompt library files.

Once the prompts for the main language have been specified, the designer has to specify them for the additional languages. This process is accelerated by using the main language prompt as a template to edit the string parts of the language dependent prompts. These prompts can be specified either at once for one language for all dialogues, or for each dialogue for all additional languages.

When editing a prompt for an additional language, only the string and break prompt items can be edited, added or removed. Prompt arguments (i.e. slots passed as arguments to that prompt) are kept from the main language prompt template, and cannot be removed since it is supposed that they are required to provide the information to the user (e.g. Your account balance is <slot_balance> euros, in English, or El saldo de su cuenta es <slot_balance> euros, in Spanish) with independency of the language. In any case, the wizard allows the designer to change the order of the arguments throughout the sentence using the arrows in number 2.

Figure 4.21 shows the window used to create prompts for additional languages. Number 1 notifies the designer about the dialogue, user level, error type, and number of occurrence the prompt that is being edited corresponds to. Number 2 shows the prompt in the main language. This prompt is used as a template to configure the prompt in number 4 for the new language (number 3). As mentioned above, number 6 and 7 allow the designer (through the pop-up window shown in Figure 4.19) to use already defined wordings or prompts

respectively. The window also allows the designer to use new SSML tags (number 5) and the possibility of including audio files in order to allow hybrid prompts (number 8).

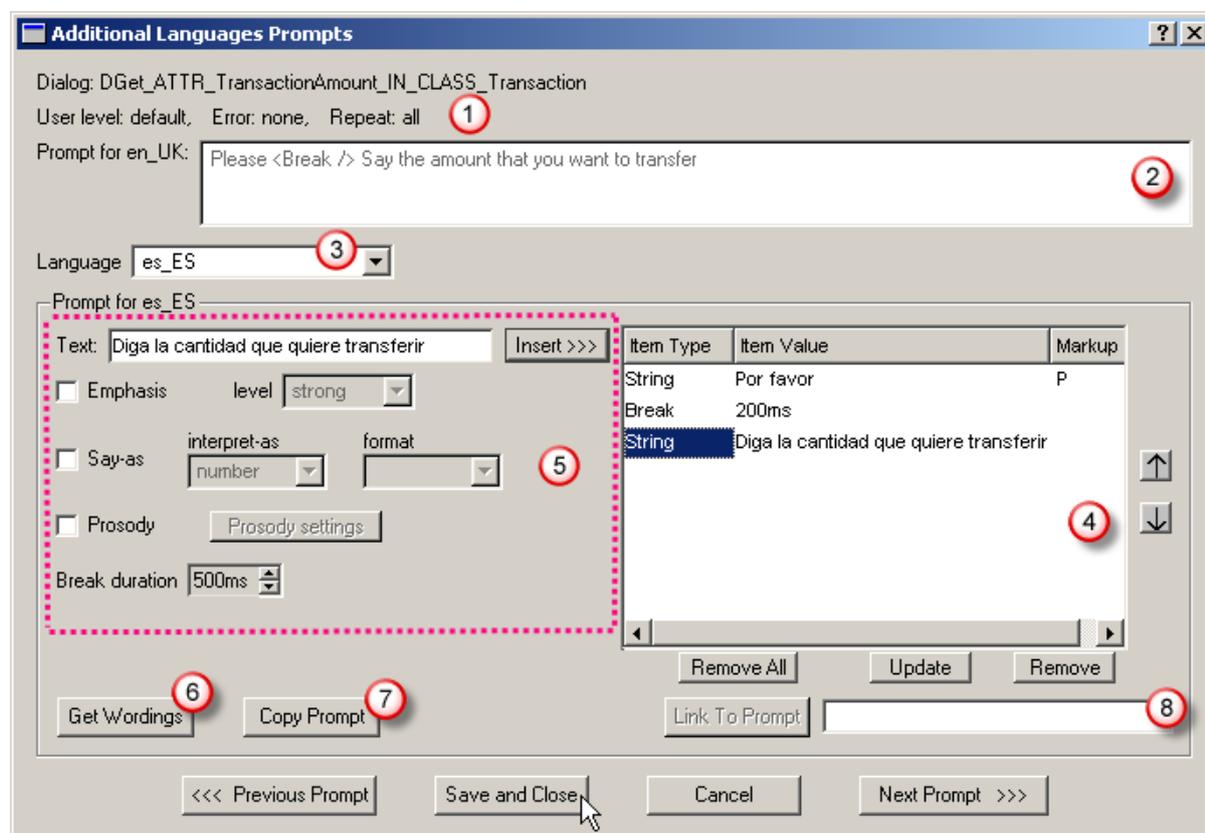


Figure 4.21. Additional languages prompts setting window.

4.7.1.2 Creation of rule-based grammars

As mentioned in section 3.4.6.2 (page 72), the Language Modelling Toolkit (LMT) is an auxiliary assistant that allows the designer to specify the language models that will be used in the runtime system to “understand” the different user answers to the system questions. This assistant helps the designer to generate and depict rule-based grammars in JSGF format, without requiring the use of third party applications. The assistant was created during the GEMINI project, and it is described in detail in [Georgila et al, 2004].

Figure 4.22 shows the main window of the assistant. In this example, the grammar name, Journey, is defined in number 1. The grammar format is selected (number 2) to be written in JSGF in Augmented Backus-Naur representation (it is also possible to select writing it in XML format). The grammar language is selected in step 3 as British English (en_UK). In step 4, the designer defines the rules that compose the grammar, specifying if they are public or private. For each rule, it is possible to define rule attributes in step 5. The rule attributes allow the definition of the slots to be used to store semantic information. In the example, destination is the name of the slot that will store the semantic interpretation given by the specific token uttered by the user (i.e. Lisbon, Paris, Athens, or Madrid). Finally, in step 6 the designer introduces, using a pop-up window not shown in the figure, the set of possible tokens the user can say in that specific rule. The pop-up window allows the designer to match the rule attribute with the corresponding semantic interpretation.

The assistant includes several accelerations that simplify the process of creating the grammars and pronunciation dictionaries for the speech recognizer. For instance, during the definition of the grammar, the assistant allows the designer to specify references to other rules, grouping and optional groupings, alternatives, rule expansions, and multi-word tokens. Besides, when creating the rules, the designer can type in words and strings of words or insert them from multiple external vocabulary files. In addition, it is possible to define all the tokens related to a rule and mapped to the same semantic attribute. For instance, in Figure 4.22, the rule <arrival> has four tokens (Lisbon, Paris, Athens, and Madrid) and all of them have the same semantic concept, “destination”. Since this information can be stored in a database consisting of hundreds or thousands of records, the assistant allows the designer to load a file containing all database records, to link them to the semantic attribute, and to decide if they are optional, or if they have to be expanded zero or more times, or expanded one or more times.

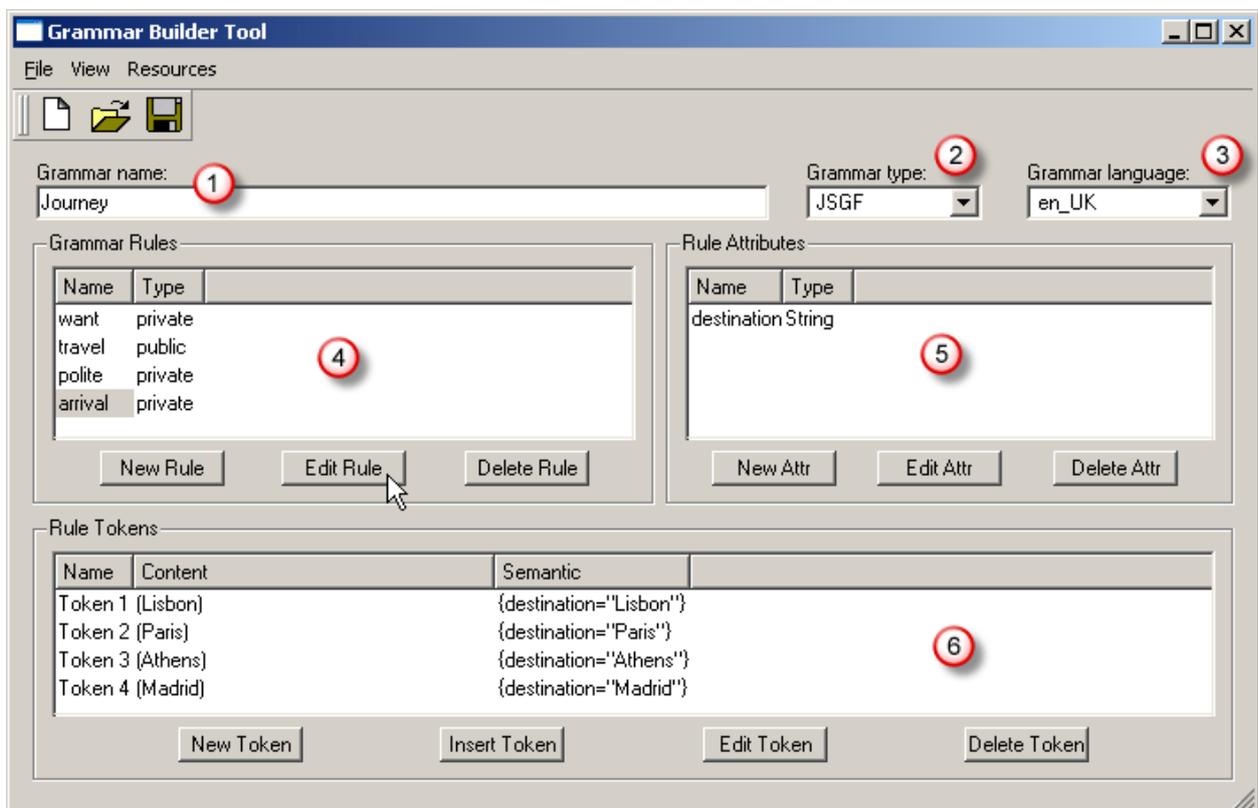


Figure 4.22. Example of the definition of a grammar rule using the Language Modelling Toolkit

Finally, the assistant also incorporates a vocabulary builder component that generates the phonetic transcriptions, in the Speech Assessment Methods Phonetic Alphabet (SAMPA⁶¹) format, of the words included in the grammar in order to create the pronunciation dictionary used by the speech recognition. Currently, the tool allows the automatic transcription for the four original languages supported by the AGP (i.e. English, German, Greek, and Spanish). However, in order to allow the transcription for other

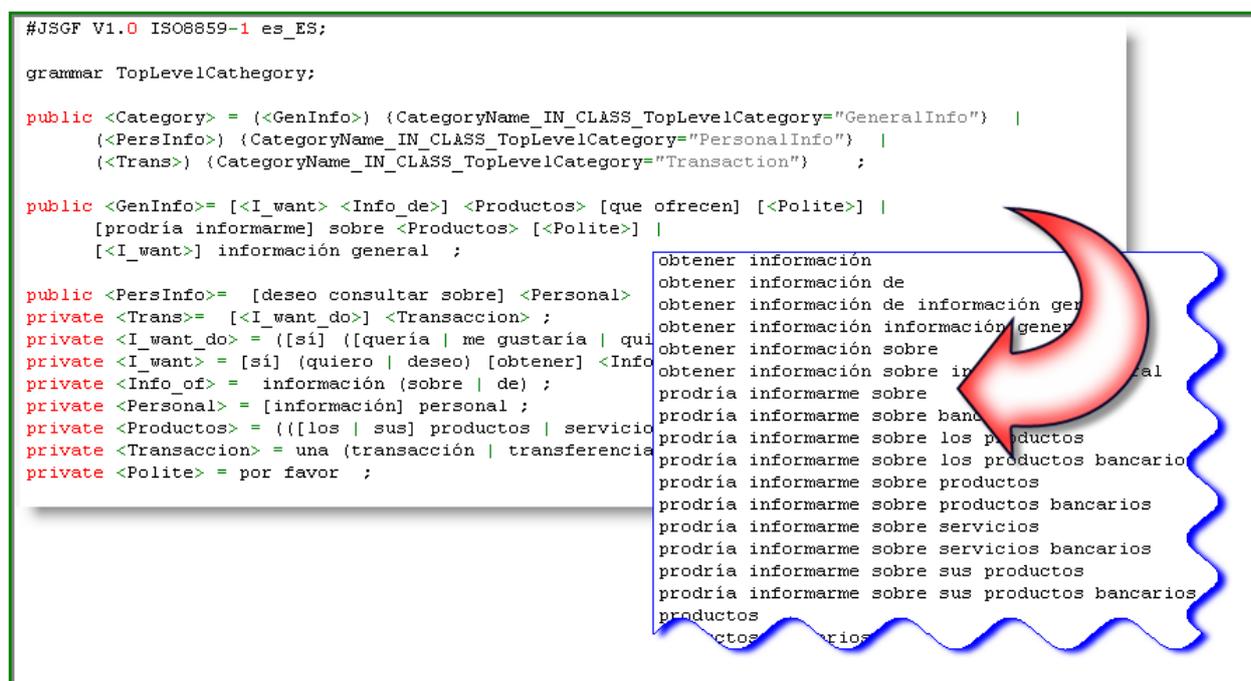
⁶¹ <http://www.phon.ucl.ac.uk/home/sampa/index.html>

languages, the tool includes a language-independent function that enables the user to write context-dependent rules for grapheme-to-phoneme and phoneme-to-grapheme conversions. Finally, [Georgila et al, 2004] report a subjective evaluation where different kind of grammars and scenarios were evaluated, showing the efficiency of the tool considering that it saves time and prevents designers from making mistakes that can be hard to locate if the grammar is generated by hand.

4.7.1.3 Creation of stochastic grammars

In addition to the possibility of creating context-free grammars using JSGF files, the platform includes two new functionalities incorporated by the author of the thesis, in order to create and debug stochastic language models. The first functionality allows the designer to obtain a text file with all the sentences that a JSGF grammar can produce when all its rules are automatically and recursively expanded. This feature allows the designer to debug the grammar providing a full list of all the sentences that the ASR can recognize. The second functionality is the automatic creation of a stochastic grammar based on word n-grams generated from the file with all the previously generated sentences. The main motivation for this functionality is to create automatically the stochastic grammars required by the ASR. This way, the ASR can support large and open vocabularies, improving the service and allowing a more robust and flexible recognition of user's utterances.

The process of creating the grammars is done using the assistant shown in Figure 4.24 after specifying the grammar and clicking in button number 1. The first step is to generate a raw text file containing all possible sentences generated from an existing rule based grammar file (currently, only JSGF files are accepted but in future releases it will be possible to use others formats) created using the built-in assistant of the platform (section 4.7.1.2) or from other platforms.



```
#JSGF V1.0 ISO8859-1 es_ES;

grammar TopLevelCategory;

public <Category> = (<GenInfo>) {CategoryName_IN_CLASS_TopLevelCategory="GeneralInfo"} |
(<PersInfo>) {CategoryName_IN_CLASS_TopLevelCategory="PersonalInfo"} |
(<Trans>) {CategoryName_IN_CLASS_TopLevelCategory="Transaction"} ;

public <GenInfo>= [<I_want> <Info_de> <Productos> [que ofrecen] [<Polite>] |
[prodría informarme] sobre <Productos> [<Polite>] |
[<I_want>] información general ;

public <PersInfo>= [deseo consultar sobre] <Personal>
private <Trans>= [<I_want_do>] <Transaccion> ;
private <I_want_do> = ([[sí] ([quería | me gustaría | qui
private <I_want> = [sí] (quiero | deseo) [obtener] <Info
private <Info_of> = información (sobre | de) ;
private <Personal> = [información] personal ;
private <Productos> = ([[los | sus] productos | servicio
private <Transaccion> = una (transacción | transferencia
private <Polite> = por favor ;
```

obtener información
obtener información de
obtener información de información gen
obtener información información gener
obtener información sobre
obtener información sobre in
prodría informarme sobre
prodría informarme sobre ban
prodría informarme sobre los productos
prodría informarme sobre los productos bancario
prodría informarme sobre productos
prodría informarme sobre productos bancarios
prodría informarme sobre servicios
prodría informarme sobre servicios bancarios
prodría informarme sobre sus productos
prodría informarme sobre sus productos bancarios
productos
ctos
rios

Figure 4.23. Example of full generation of possible sentences from a JSGF file

The process for generating the sentences is to first expand all rule names (e.g. <Category>, <I_want>, etc. in Figure 4.23) defined in the file, considering at the same time all the possible combinations with special JSGF symbols for alternatives, grouping and optional grouping (i.e. |, () and [] respectively). Although the assistant is flexible and robust enough to parse several kinds of rule names, there are some limitations regarding the standard JSGF format; for example, the assistant does not expand recursive/nested rules, incomplete or malformed rules, unary operators neither weights information. These improvements will be considered in future developments of the platform.

The second step, number two in Figure 4.24, is to generate the stochastic grammar from the sentences created in the previous step or from new texts. The designer needs to select, number 3, the text file generated in the previous step or include more files, and the name of the grammar file to be generated (number 4). By default, the assistant automatically proposes the name of the output file using the name of the JSGF file used in the previous step or from the first text file selected in step three. With this information, the assistant generates three files: 1) the grammar file (up to trigrams), 2) a vocabulary file containing all different words that appears from the selected texts, and 3) the pronunciation file, in SAMPA format, for the recognizer.

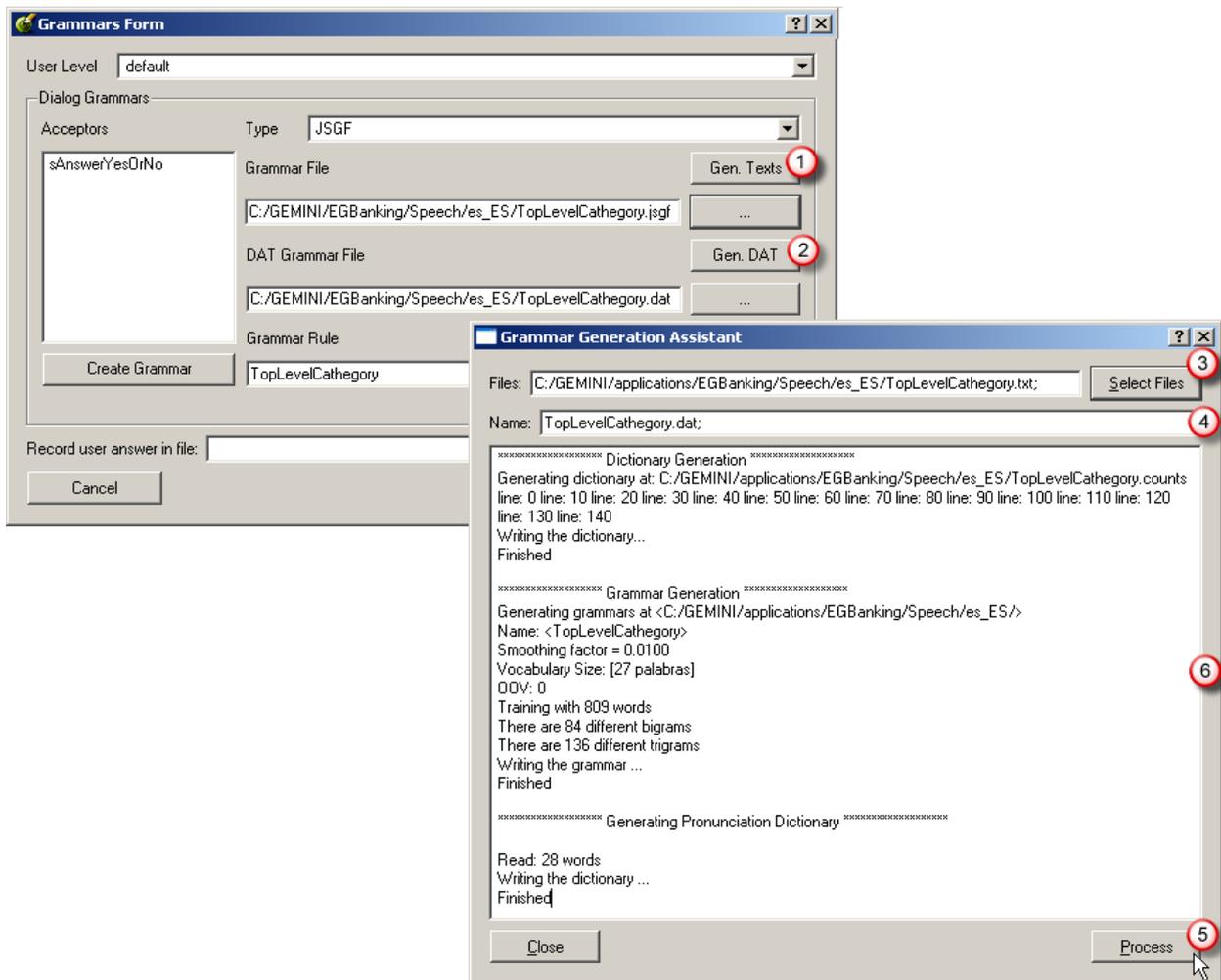


Figure 4.24. Assistant for the creation of stochastic language models

4.8 Conclusions

In this chapter, we have described all the accelerations included in the design platform in order to speed up the design and guide the designer through all the steps required to create a multimodal and multilingual dialogue service. Apart from the direct applicability of these accelerations in the design platform, and the advantages for designers, the proposed accelerations are in most cases innovative and do not exist in any of the current commercial and research platforms.

Different types of accelerations have been proposed according to the requirements, capabilities, and available information at each assistant. In some cases, the accelerations take advantage of heuristic information extracted from the contents of the backend database used in the service or from an object oriented data model structure that represents the tables, fields, and relationships between fields of the database. In other cases, the accelerations consist of the application of predefined and configurable rules that using contextual and previously defined information from other assistants, allows the generation of different kinds of proposals that simplify the process of creating and fulfilling the dialogue flow. Finally, other accelerations consist of different wizard windows or simplified processes that help designers to complete, create, or debug models (e.g., grammars, prompts, SQL commands) required by the design and runtime platform.

In order to provide some of the accelerations, an automatic procedure to extract heuristic information from the backend database was created. During this process, we had to deal with some limitations of the connecting database driver, mistakes in the definition of the fields in the database, and for mapping some field types supported by the platform but not for most database engines. These problems were solved through a set of regular expressions that were evaluated on different available databases obtaining an average correlation of 89.6 % when compared to the classification made by a human evaluator.

In relation with the accelerations included to create the data model structure we have proposed a new assistant that automatically exploits the heuristics extracted from the database contents in order to automatically propose, organize, and simplify the process of defining the classes and attributes.

In relation with the assistant that defines the prototypes of the database access functions, we have incorporated a new wizard window that allows the automatic generation and debugging of SQL queries used by the real-time system. Internally the system implements an automatic procedure that analyzes and proposes the SQL statements using information regarding the type of the input/output parameters. This acceleration contributes to reduce the necessity of learning a new programming language (SQL in this case), and goes one step forward to similar assistants in current development platforms: the queries are automatically proposed. Another acceleration, proposed and implemented by the partners of the GEMINI project, allows the definition of relations between the function arguments and the data model structure. Then, in the following assistants, these relations are exploited through different kind of automatic proposals that simplify the design.

In relation with the assistant where the state flow model is generated, during this thesis, we have contributed with a new graphical interface that simplifies the creation of the transition network, and provides a clear overview of the dialogue flow by using an automatic algorithm that reduces and reorganizes the information displayed to the designer. On the other hand, the main accelerations are the automatic generation of different state proposals that can be used to quickly create complex states, together with the possibility of using an automatic analysis of the feasibility of the slots defined in a given state of being requested

using mixed initiative forms or direct forms. Both accelerations represent an important effort for the reduction of the design time and to improve the quality of the generated flow in comparison to current development platforms. Especially interesting is the process of automatically proposing the slots to be requested using mixed-initiative forms or directed dialogues which is not provided in any other platform.

In relation with the retrieval modelling assistant, it has been the assistant where the higher number of accelerations have been proposed and implemented. Specifically we have incorporated several automatic dialogues and templates that can be used to obtain or present information to the final user. In a similar way to the previous assistant, the proposed dialogues help to reduce designer mistakes when creating new dialogues by proposing default built-in dialogues. Besides, we have incorporated an auxiliary window where the designer can find all the actions that are considered relevant for the dialogue being edited. This acceleration and the procedure to fulfil the information presented to the designer do not exist in any other platform and it is one of the most important contributions of this thesis. In addition, we have also created an automatic procedure to help the designer to connect the input/output parameters of different actions and dialogues with the local/global variables that contain or will contain the information for/from those actions and dialogues. Again, in this case the assistant applies different rules and mechanisms to automate this process that is usually made by hand in other development platforms. Finally, we have also designed a simple procedure to define dialogues with mixed-initiative or over-answering capabilities.

In relation with the assistant that defines the specific details for the speech modality, the proposed accelerations were the automatic generation of the dialogue flow required for the confirmation handling of the user answers, together with an assistant where the dialogue flow for providing the information contained in a list of retrieved results after querying the backend database can be specified. In this case, both accelerations provide innovative contributions to the design of spoken dialogue applications by proposing different procedures and dialogue flows, codified through predefined templates, considering the number of items to show to the user, the number and type of the slots to be requested, as well as the confirmation type to be used. Finally, the assistant semi-automatically proposes the information required to complete some actions or steps of the dialogue flow.

Finally, other assistants in the platform were also accelerated in order to allow the quick definition of language dependent prompts and pronunciation vocabularies used by the speech recognizer, as well as the creation and debugging of stochastic grammars.

5 EVALUATION OF THE ACCELERATION TECHNIQUES

In order to estimate the goodness of the platform, its assistants, and the different acceleration techniques, two different types of evaluations were proposed: the first one is a subjective evaluation where several persons, with different levels of experience on designing and programming dialogue applications, rated the design platform and its assistants. The second one is an objective and subjective evaluation, where another set of designers were proposed to evaluate the platform using a measurement system that provided several objective metrics when using the assistants and compared our platform with an alternative assistant with less accelerations. Finally, these evaluators were also asked to fill in a subjective evaluation form. The next sections present full details for both evaluations. Finally, in section 5.3 (page 147) we present the conclusions of the evaluations.

5.1 Subjective Evaluation

In order to rate the usability and acceptability of the platform, during the GEMINI project we carried out a subjective evaluation of all the assistants included in the Application Generation Platform (AGP). The main topics evaluated at this time were: a) the friendliness of each assistant in the platform and the whole platform interface, b) the complexity and time required to learn to use each assistant and the whole platform, c) the level of functionality of each assistant, d) the level of consistency, transparency, and intuitiveness of each assistant, and e) the willingness of the evaluators to use the platform to develop dialogue applications. Appendix D contains the detailed questionnaire the evaluators had to answer.

5.1.1 Experimental setup

Since this evaluation was carried out during the GEMINI project, we were able to evaluate the platform and its assistants with different groups of evaluators created from the partners of the project. In this case, the Greek group was created from people from the University of Patras – Wire Communication Lab (WLC⁶²) and from the Knowledge⁶³ company. The German group was created from workers at Temic SDS GmbH company (currently, Harman-Becker⁶⁴). Finally, the Spanish group was created from people working or studying at the Universidad Politécnica de Madrid (UPM). In turn, these groups were divided into three categories considering the level of knowledge and experience on designing dialogue applications. The three categories considered were: novices, i.e. testers that never developed a dialogue application before, intermediate, i.e. testers with some experience on using or designing dialogue services, and finally experts, i.e. evaluators who previously had used other platforms or languages to develop dialogue applications.

⁶² <http://www.wcl.ee.upatras.gr/>

⁶³ <http://www.knowledge.gr/>

⁶⁴ <http://www.harmanbecker.com/>

Partner	Novice	Intermediate	Expert	Total
WCL (Greece)	11	4	0	15
KNOW (Greece)	0	4	6	10
HB (Germany)	9	0	0	9
UPM (Spain)	4	3	0	7
Total	24	11	6	41

Table 5.1. Distribution of the evaluation participants for the subjective test

Table 5.1 shows the statistics and details about the forty-one participants involved in this subjective evaluation. From WCL, 15 subjects, post-graduate students, were involved. All of them having experience in at least one programming language, but most of them with no experience in dialogue applications. From this group, four were classified as intermediate level since they had some experience in Web applications. From Knowledge, 10 subjects, developers working in the company, evaluated the platform. Fifty percent of the subjects were speech application developers, whereas the other fifty percent were Web application developers. In this group, the average number of years of experience in designing dialogue applications was three. Therefore, they were categorized as intermediates and experts. From Temic, nine subjects, software engineers working in the company, evaluated the platform. All of them were classified as novices as they had never before written a dialogue application. Finally, from the UPM, seven subjects, pre-graduate students and teachers of the university, were involved. In this case, three were classified as intermediate with an average of four years of experience in speech dialogue development. The age of the users was varying in the range of 21 to 49 years old. Considering the subjects' mother tongue, the groups included Greek (61%), Spanish (17%), and German (22%) speakers. Figure 5.1 illustrates the final distribution of programming skills of the users involved in the evaluation.

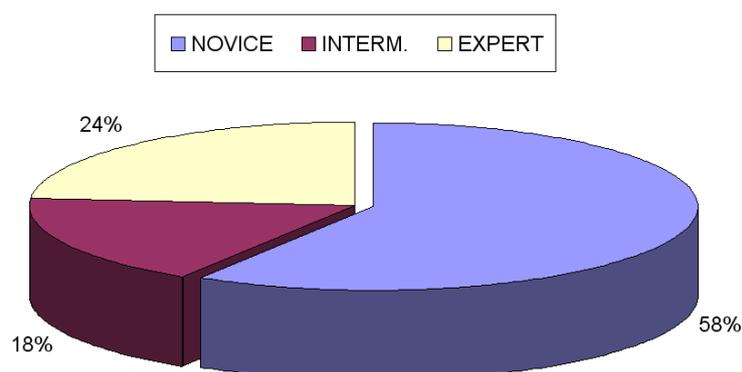


Figure 5.1. Final distribution of experience status for the evaluation participants of the subjective test

The evaluation was done in one session of 3-4 hours. This session was divided into three main blocks. To start with, an overview of the platform was presented to the participants. Including also information about some definitions and terms (e.g., multimodality, over-answering, mixed-initiative, etc.) used during the evaluation that were

not known at all by most of the participants, especially the novice users. Besides, the architecture of the platform, the sequence of the layers, and the functionality of each assistant were also explained.

During the second block, the participants received a detailed demonstration of each assistant, its interface, capabilities, accelerations, settings, and proper input and output were described. The demonstration was based on a demo version of a banking application.

Finally, in the third section, the evaluators were asked to use each assistant of the platform in order to create an appropriate model for a new dialogue service. For instance, the testers were asked to perform the following tasks: to create the data model for three complex classes, to prepare at least three database access functions for the DCMA. In addition, they had to design at least four states in the SFMA, to get the information about a house loan, to do a transaction of a certain amount between two accounts using mixed-initiative and to obtain the current value of a currency in the RMA. The final goal of all these tasks was to cover more than the 90% of the functionality of the AGP. In any case, in order to carry out all these tasks within the limitation of the 3-4 hours of the evaluation, the participants were allowed to create and reuse some predefined libraries and models without starting from scratch. After finishing the evaluation, a short discussion was carried out about the main problems that they had faced, and they were also asked to answer a questionnaire with specific questions for each assistant and for the overall appearance and behaviour of the platform.

5.1.2 Evaluation results

As we have described before, after finishing the evaluation of each assistant and the whole platform, the participants were asked to answer the questionnaire included in Appendix D. The questionnaire consists of four questions per assistant and seventeenth for the overall AGP. The scale used by most of the questions is a 10-point scale being 1 the minimum and 10 the maximum score. In this section, we will show the results obtained for each of the four questions per assistant and for the overall evaluation of the AGP.

In summary, we can say that all the assistants were rated positively, with an average score between 7.0 and 8.5, which is a very homogeneous score. Here, the ADA, DMA, SFMA, DCMA, MERA-Speech, and UMA assistants were rated, in average, between seven and eight considering the four initial questions. The RMA was rated as having a very good overall appearance and extremely good functionality. However, it had some difficulties in learning and was perceived as less intuitive. The most probable reason is that it is the assistant that provides the biggest functionality for dialogue design, as it is the place where detailed design is made, and the evaluating designers were given very little time to learn each assistant and practise with them, so in practice evaluators only read part of the documentation provided beforehand regarding the designs they were asked to do.

On the other hand, the participants rated the assistant for the modality and language extension (MEA) below seven for all the questionnaire sections. In this case, the low results are probably the result of the time required to set the prompts and grammars (text typing), especially for different language definitions where machine translation and more prompt and grammar libraries would be desirable.

- How quickly did you learn to use each assistant?

As we can see in Figure 5.2, the average score for all the assistants was 7.7, which is a very good rate considering the different functionalities and processes done by each assistant. As we have previously mentioned, the RMA obtained only a 6.4 score mainly due to the short time the participants had to use it and the big number of functionalities offered by this assistant. In the other hand, the MERA-Speech, the other assistant developed in this thesis, was highly rated since most of the complexity of the generated flow is hidden to the designers.

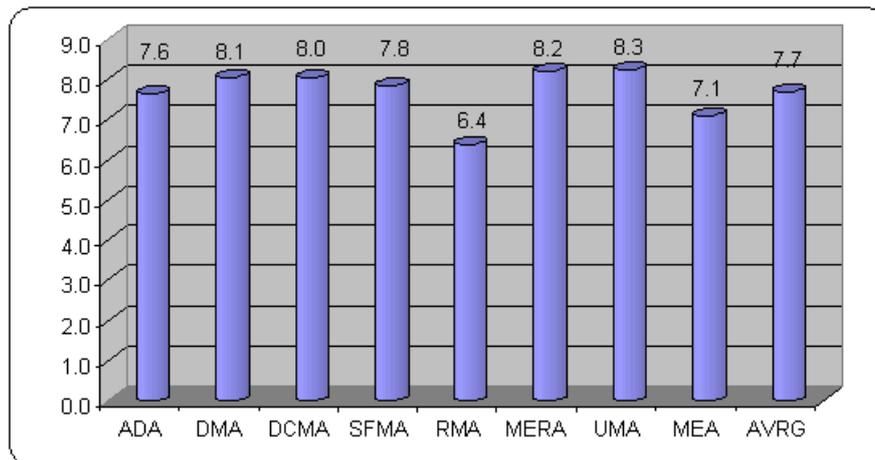


Figure 5.2. Evaluation results for the question about how quick the participants learnt to use the platform assistants

- Is the assistant easy and intuitive to use? Do you know what to do at each step?

According to Figure 5.3, the average rating for all the assistants in the platform was 7.3. Here we observe a similar behaviour regarding the previous question, i.e. the most complex assistants have the lower score. It is interesting also to observe again that the MERA-Speech is one of the most positively rated assistants. One of the reasons was that its graphical interface contained enough information to guide the designer.

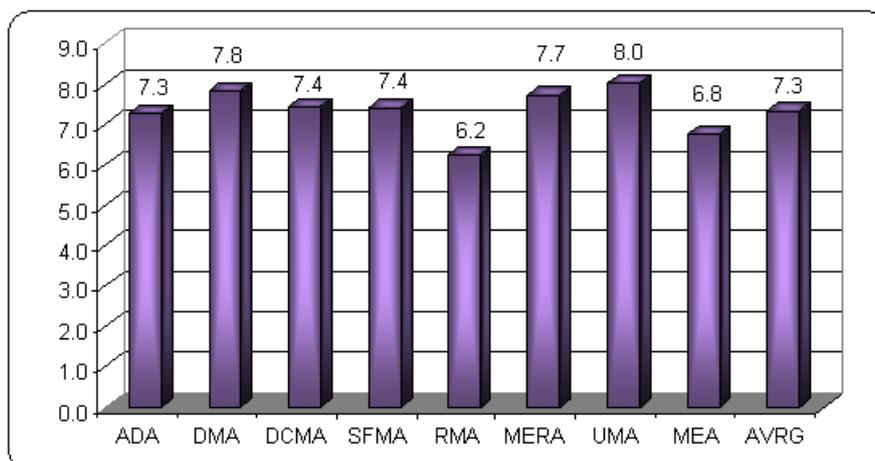


Figure 5.3. Evaluation results for the question about how easy and intuitive were the platform assistants

- Is the functionality of the assistant sufficient?

Figure 5.4 shows the results for the third question in the survey. Here we can see that all the assistants were highly rated, with an average score of 8.0. In this case, the RMA and MERA-Speech assistants obtained the maximum scores. It is important to mention that, at the time of this evaluation, most of the accelerations described and designed by the author of this thesis had not been included in the DMA, DCMA, and SFMA assistants, therefore they could have obtained a higher score. On the other hand, the MEA assistant was the assistant with the lowest score. As we mentioned above this was mainly due to the low number of accelerations included in it.

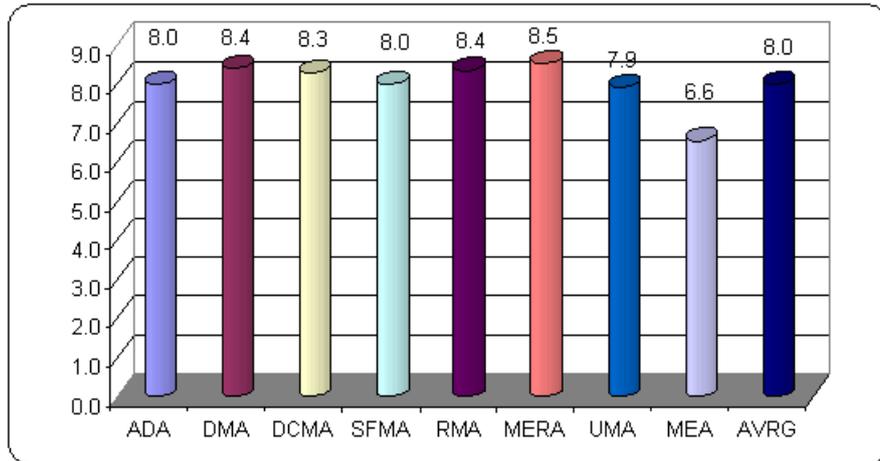


Figure 5.4. Evaluation results for the question about how sufficient was the functionality of each platform assistant

- How do you rate the appearance of the assistant (consistent, transparent, and intuitive)?

Figure 5.5 shows the results for the last question about each assistant. The average score for all the assistants is 8.0. It is interesting to observe that for this question the results were more homogenous but consistent with the previous one. Again the assistants designed and implemented by the author of the thesis were the ones that obtained the highest scores.

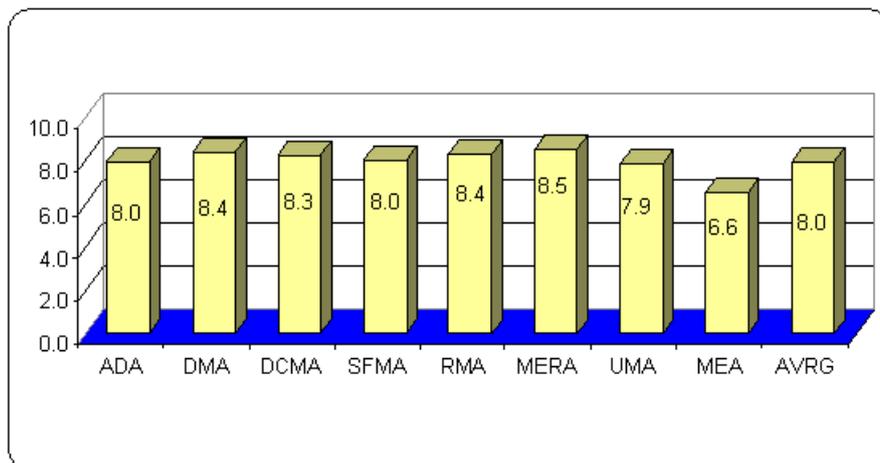


Figure 5.5. Evaluation results for the question about how consistent, transparent, and intuitive the users rated each platform assistant

Figure 5.6 shows the average results for the four general questions included in the survey considering each assistant independently.

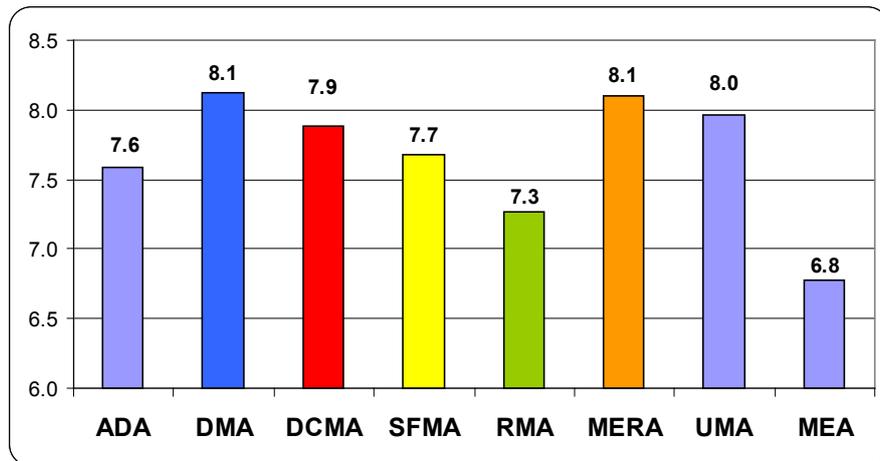


Figure 5.6. Average results of the subjective evaluation for general questions about the assistants

Finally, the questionnaire gave the participants the opportunity of providing comments and suggestions about each assistant. Next, a summary of their comments is provided.

- “All the assistants require some kind of contextual help at “mouse-over” in order to guide the designers about the capabilities and functionalities of the assistant”;
- “Some of the assistants require some kind of GUI adaptability to the monitor resolution”;
- “The SFMA and RMA functionality should eventually be integrated into one assistant, as they are using at a high degree the data model”;
- “The RMA is the most demanding regarding the time needed to understand it and use it, however it boasts a high quality GUI, desirable for the other assistants too”;
- MERA-Speech: “very nice functionality for lists handling, offering a wide range of possibilities”;
- UMA: “very simple functionality, that could eventually be integrated into one of the Speech modality assistants (MERA-Speech or MEA)”;
- MEA: some users remarked that setting additional language prompts was too slow, possible due to the expectation of automatic text translation from the main language;
- MEA: additional knowledge and learning time is required at this level (for a speech developer to get used with Web pages specific items, and for Web developers to understand speech resources design).

Finally, the participants had to answer the second part of the questionnaire to evaluate the platform as a whole. In this evaluation, 1 means very poor and 10 means excellent. Table 5.2 shows the results of this section of the evaluation. The overall score was in average: 8.37, with the maximum scores in the following aspects:

- Speeding up the development time of an application
- Over-answering and Mixed-initiative functionality
- Lists handling for speech applications

Question	Average rating
The provision of data modelling and connecting to external data sources	8.7
The provision of application state flow modelling	8.6
Easy adaptability to other languages	8.2
Easy adaptability to other modalities	7.9
Ready-made error-handling (nomatch, noinput)	8.1
Speed up of development time as compared to writing VoiceXML/+xHTML code by hand	9.0
Provision of user modelling	7.8
Provision of mixed-initiative dialogue handling	8.5
Provision of list handling	8.6
Provision of over-answering	9.0
Provision of easy connection to run-time modules	7.9

Table 5.2. Subjective evaluation of the platform

Following we include the questions and a short description of the statistics obtained for each one.

Did you learn quickly how to make applications with the AGP?

In this case, 55% of the users said yes, and 45% of the users said no. Considering the level of experience, most of the novice and intermediate participants answered “yes”, while experts answered “no” in most of the cases. The reason for this answer from the experts was that they had certain expectations of the platform and the basic training received at the beginning of the evaluation was considered as not enough.

Do you think non-experts could use the AGP efficiently?

In this case, 32% said “yes”, although they were concerned about terms that are not familiar for novice users such as over-answering, mixed-initiative, multimodality, etc. On the contrary, 68% of testers said “no”. They considered that the platform is useful only for experts; and that non-experts will not be able to design and deploy a good dialogue application even using the AGP. However, they considered that with more training, it should be possible. In any case, this is a common procedure for all speech or Web platforms.

How do you rate the overall appearance of the AGP (consistent, transparent, and intuitive)?

The average score in this question was 7.7. This low score is mainly due to the fact that for the overall rating users considered also the AGP GUI, the basic functionality of each assistant and the overall effort to use the platform, and not only the specific platform functionality (i.e. mixed-initiative, list handling, etc.).

Do you find the various assistants of the AGP are well integrated?

In this case, 88% of the users answered “yes”. However, some testers considered important to suggest a possible merging of some of the assistants (e.g., DMA, DCMA and SFMA, or RMA and SFMA).

Would you use this system in the future or recommend it to develop speech/Web applications?

Positively, 95% of the testers answered “yes”. Only two expert Web developers said “no” because they preferred to write HTML code by hand instead of using the AGP.

5.2 Objective Evaluation

Finally, the performance and usefulness of the different accelerations included in the platform were validated through an objective evaluation. Here, our idea was to obtain a set of quantitative measures obtained by different testers when they were requested to perform different tasks using the AGP and a parallel tool that does not include any of the accelerations described in this thesis. Then, these measures were used to compare the performance of each assistant and the whole platform in relation to the other tool. The next section describes the experimental setup including information about the proposed quantitative measures, the assistants and tasks that were evaluated, the tool used in the comparison with the AGP, as well as information about the participants, and the answers of the evaluators to a subjective test measuring the assistants and their accelerations.

5.2.1 Experimental setup

The first thing we had to specify was the objective measures to obtain. However, currently there is not a standard method for evaluating the accelerations proposed. [Jung et al, 2008] propose a set of quantitative measures for measuring the performance of a dialogue design platform with accelerations. In their proposal, they set different tasks that the evaluators had to carry out using the platform with accelerations and an open text editor chosen by each participant. During the evaluation, different metrics are measured such as mouse clicks, keystrokes, and elapsed time. Then these metrics are used to compare the performance of the platform with the hand-made models created using the text editor.

For evaluating our platform, we have decided to follow a similar approach, introducing some differences, obtaining a set of similar quantitative measures when different assistants of the AGP are used in order to complete a set of previously defined tasks. Then, we compared these quantitative measures with the ones obtained when annotating the same tasks in the internal language format used by the platform, i.e. GDialogXML syntax. In this case, the evaluators made the annotation using a semi-automatic assistant called Diagen (see section 3.4.6, page 71) already included in the platform. The reason to use this assistant, instead of allowing the evaluators to use any text editor of their liking, was to make a fairest comparison

between both evaluations. It is well known that writing any information in any XML-based language is a tedious and difficult task (especially if the XML document is required to be well formed and the XML tags are case-sensitive as in the GDialogXML specification). In addition, the assistant reduces the necessity of memorizing the XML specification. On the other hand, almost all developers and development platforms use some kind of tool for writing from scratch or performing fine-tuning of the code generated by the main application. Diagen is a representative example of this kind of applications.

In addition, it is important to mention that the main reasons for selecting a comparison between using the AGP assistants and Diagen, instead of comparing with other development platforms, was that we could not find any commercial or academic platform comparable to the AGP. For instance, most of these platforms create only VoiceXML applications instead of multimodal services as in the AGP, or they do not take into account the Database information neither include all the accelerations that we wanted to evaluate. Finally, most of the commercial platforms have an advanced graphical interface that we were not interested on evaluating. It is well known that the appearance of the GUI will have a great influence over the evaluators.

The evaluation was done by 9 testers which were classified in the same three levels defined for the subjective evaluation (section 5.1.1, page 125): novice, intermediate, and expert. All the evaluators had some experience in at least one programming language but most of them had no, or a very little, experience in designing dialogue applications. The evaluators, most of them pre-graduate students at our university, were then classified according to their level of experience on developing dialogue applications resulting in the following groups: 4 evaluators in the novice group, 3 in the intermediate level, and 2 in the experts group. From this group, only three participants had some knowledge of the AGP. The average age was 27 years old (from 22 to 41 years old).

Since not all the assistants in the platform were developed by the author of the thesis neither they include most of the accelerations proposed and described in this thesis, we decided to include in the evaluation only the following assistants that are the basic contribution of this thesis: DMA, DCMA, SFMA, RMA, and MERA-Speech. All these assistants were initially proposed to be also evaluated using the Diagen assistant. However, after considering the high complexity of the dialogues produced by the MERA-Speech due to the automatic templates included on it, we decided not to include this assistant in the non-accelerated evaluation, since it would involve an excessive work and, evidently, the comparison would be unfavourable.

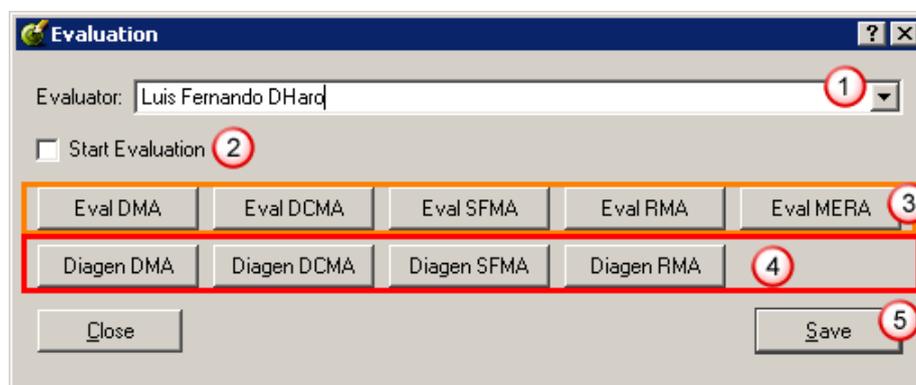


Figure 5.7. Interface used to start the evaluation of the different assistants in the AGP and using the Diagen assistant

Figure 5.7 shows the window used to perform the evaluation of the different assistants. Number 1 allows the selection of the name of the evaluator in order to save the corresponding files with the statistics. Number 2 is a check box to indicate that the evaluation can be started. This check box controls if the testers are using the assistant for training purposes or for evaluation. Number 3 and 4 allow the evaluator to select the assistant to be evaluated using the AGP or Diagen respectively. Number 5 allows the evaluator to save the quantitative measures of the current assistant being evaluated.

The evaluation was done in two sessions of 4 hours each. During each session, the testers were trained in all the assistants to be evaluated at that session. During the first session, the evaluators received a complete explanation of the whole platform, the goals of the evaluation, and the interfaces used to get the statistics. Finally, they also received instructions and evaluated the three first assistants in the evaluation: DMA, DCMA and SFMA. During the second session, the evaluators learnt how to use and evaluated the RMA and MERA-Speech assistants. In general, each assistant evaluation was divided into three main blocks: in the first one, the evaluators received instructions about the capabilities and accelerations included in the corresponding assistant through examples of use. In the second block, the evaluators were proposed to carry out an example task using the assistant in order to consolidate the knowledge acquired in the first block and to allow to answer to the doubts that could arise when practicing with the assistant. Finally, during the third block, the evaluation was carried out and later the evaluators were also requested to fill in a subjective survey to measure the acceptance, usability, intuitiveness, and most interesting features of each assistant, etc.

As we have mentioned before, the quantitative measures proposed in [Jung et al, 2008] are the number of keyboard strokes, the number of mouse clicks, and elapsed time. In this evaluation, we have included one more measure: the number of times the user presses the delete key when typing. The goal of this new metric was to provide an additional measure of the difficulty of introducing information in the assistants or writing the GDialogXML code. Besides, since the assistants reduce the number of times the designer had to type, this fact could also be reflected in the number of errors the designers could make. In order to obtain these statistics, each assistant to be evaluated was executed in parallel with a measurement system. This system captures all the events from the mouse and keyboard, differentiating the events occurring in the assistant window from events occurring in other applications. Finally, we also included an automatic and invisible mechanism (the testers were not aware of this process) for recording and saving the screen during the evaluation. The recorded videos were later reviewed in order to find out the main problems the testers found and to obtain a visual feedback of the process that the testers followed to complete the tasks. These videos also allowed us to discover if the testers used or not the accelerations included in the assistants, and the steps that took most of the elapsed time during the evaluation.

Figure 5.8 shows the program used to save the statistics. Number 1 allows the evaluator to select the step to be evaluated according to the current assistant selected in the previous form window (Figure 5.7). Each time the evaluator changes the step the system automatically checks out if the statistics has been saved or not, and asks the evaluator what to do. Number 2 is used to start or to pause the evaluation. When the evaluation is paused, neither mouse nor keyboard events are registered. Number 3 allows the evaluator to save the statistics and to go to the next step in the evaluation. Number 4 is used to clear all the registered events for a given step, starting it from scratch. Number 5 shows the different counters used in the evaluation. Number 6 and 7 allow the evaluators to reduce in one the number of clicks and

keystrokes registered. The objective of these two buttons was to allow the evaluator to correct manually an error when evaluating the assistants, and to measure the difficulty/uncertainty of a given task considering that the evaluator was probably confused. In the same way, number 7 automatically records the number of times the evaluators press the backspace or delete keys. In practice, the evaluators rarely use these two buttons, so we only used the number of times the backspace or delete keys were used.

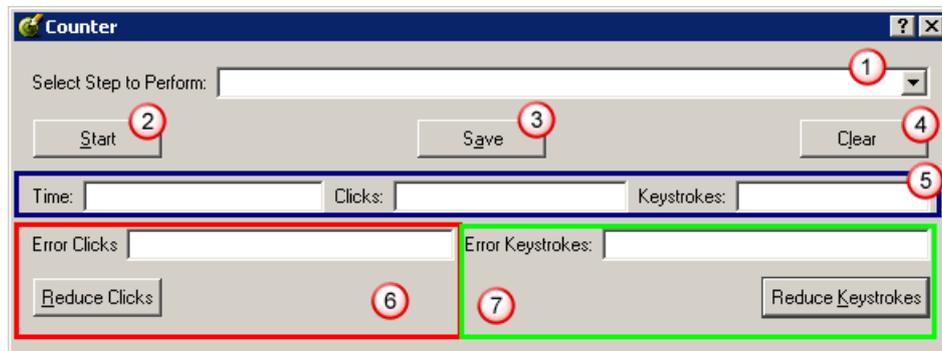


Figure 5.8. Interface to record the mouse and keyboard events during the evaluation

5.2.2 Description of the evaluated tasks and results

In order to test the different accelerations included in the platform, we designed a set of tasks to be performed by the testers using the assistants included in the AGP and using Diagen. Each of the proposed tasks could be divided into one or several steps in order to test, gradually, the different possibilities and accelerations allowed by the assistants, as well as the different kind of problems that a designer could find when developing a real application. This section will briefly describe the different tasks proposed and the steps included in each task, as well as graphics comparing the performance of the assistants of the AGP vs Diagen. In order to simplify the presentation of the evaluation and the comparison considering the different tasks, participants, and metrics, we have included a different graphic for each task and assistant evaluated. In general, these figures show the average improvements, in percentage, obtained when comparing each quantitative measure obtained using the corresponding assistant in the AGP and with Diagen for each type of participants, for all, and the average improvement considering all the metrics. In each figure, a positive value means that the assistants of the AGP perform better than Diagen, and a negative value means that Diagen outperforms the corresponding assistant. Finally, we have included in Appendix E tables with all the information represented in the graphics, the specific values for each quantitative measure obtained during the evaluation, and the results of a subjective evaluation with general questions about the evaluated assistants and Diagen.

For the DMA assistant, we proposed the evaluators to test two different steps or cases. In the first one, they were requested to create a class model with two atomic attributes. Both attributes were related to the database. In this case, it was expected that the evaluators used the assistant described in section 4.2.1 (page 89). However, during the evaluation we observed a big difference between the time used by experts and novices/intermediates to fulfil the task. For that reason, we repeated the available strategies of the assistant to the participants. After the evaluation, we reviewed the videos of each tester and confirmed that some of the novice and intermediate testers did not use the available accelerations but create each attribute using an alternative, not accelerated, method. Nevertheless, as we can see in Figure 5.9, the AGP performs better reducing in average the design time by 56.6%, the number of clicks in 30%, the keystrokes in 93.1%, and the number of keystroke errors (i.e.

using the delete or backspace keys) in 96.3%. Finally, we can see that in average for all the participants the proposed accelerations obtain a 69% improvement when compared with Diagen.

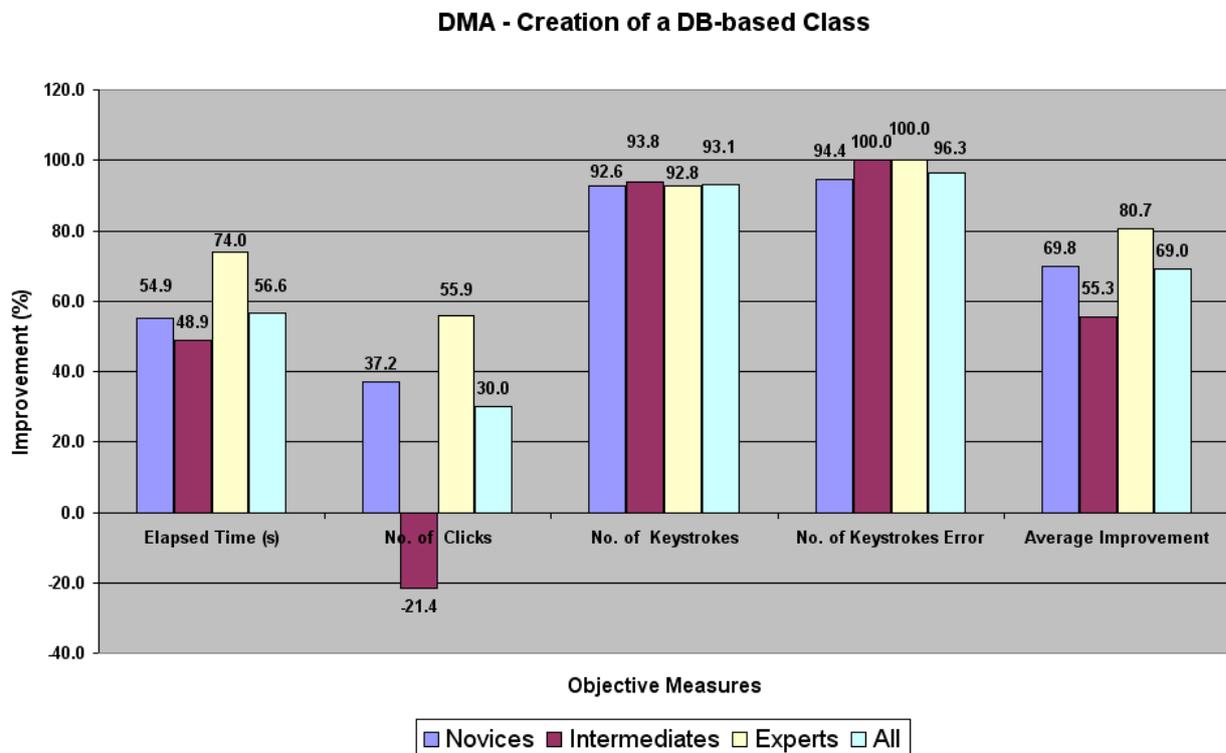


Figure 5.9. Chart with the improvements obtained when evaluating the Data Model Assistant for the creation of a class structure with database dependency

In the second step, the testers were asked to create a mixed class structure, including two atomic attributes (both related with the database and with language dependency) and one complex attribute (a list of embedded objects). This time, during the evaluation, and reviewing later the video recordings, we observed that the participants used the available wizard with accelerations to create the two atomic attributes and used the available mechanism to create the complex one. Figure 5.10 shows the improvements for this evaluation. Here we can see that there is a big improvement using the AGP for the number of keystrokes and keystroke errors. This is mainly due that for this task the GDialogXML code is more complex and the evaluators had to type much more, and the possibility of making errors was also higher. Besides, we also observed that the improvements for the elapsed time between the experts and novices was quite similar, which means that both were using better the available accelerations. Considering that this task requires a greater and more complex GDialogXML code we should expect bigger improvements in the elapsed time. However, when we inspected the video recordings we observed that the participants were getting used to the Diagen interface and therefore working faster with it. This behaviour was increasing throughout all the evaluation and it is a collateral effect that should be kept in mind. Finally, we observe that for the number of clicks there is a negative improvement (14.3%) for the intermediate participants. In this case, the video recording showed that one of the intermediate testers had some problems to create the complex attribute using the AGP, then generated more clicks than the others do. As a future solution, we propose to extend this assistant with a new wizard window specialized in creating complex attributes. Finally, the global improvement in this case was 61.9%.

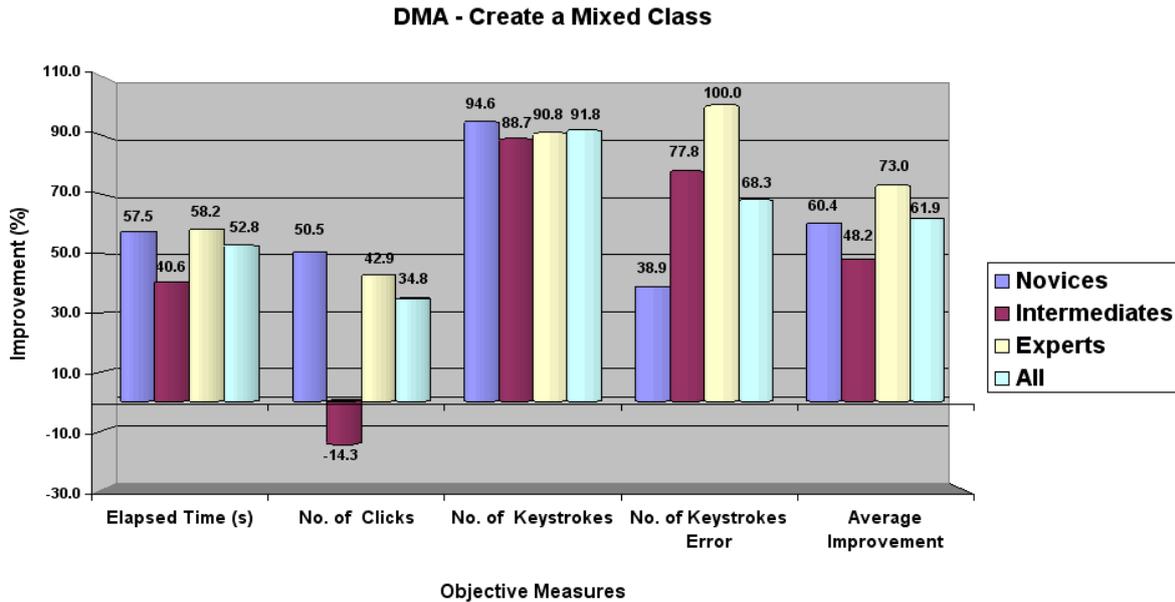


Figure 5.10. Chart with the improvements obtained when evaluating the Data Model Assistant for the creation of a mixed class

For the DCMA assistant, we proposed the evaluators one single task. It consisted on creating a function with two input arguments and one output argument. In this case, all the parameters were related to the data model. Although this assistant includes very few accelerations, Figure 5.11 shows that the design time can be reduced in 19.9%, and 16.6% in general. From these results, we propose as future improvements to define a new mechanism for defining the input/output parameters that could be especially useful for novice and intermediate users in order to reduce the keystroke errors, number of clicks, and elapsed time. Finally, it is important to mention that the participants were able to test the functionality of creating and testing SQL statements using the wizard described in section 4.3.2 (page 93). However, this process was not included in the evaluation because this kind of information is not included in the GDialogXML syntax and therefore it cannot be generated by Diagen.

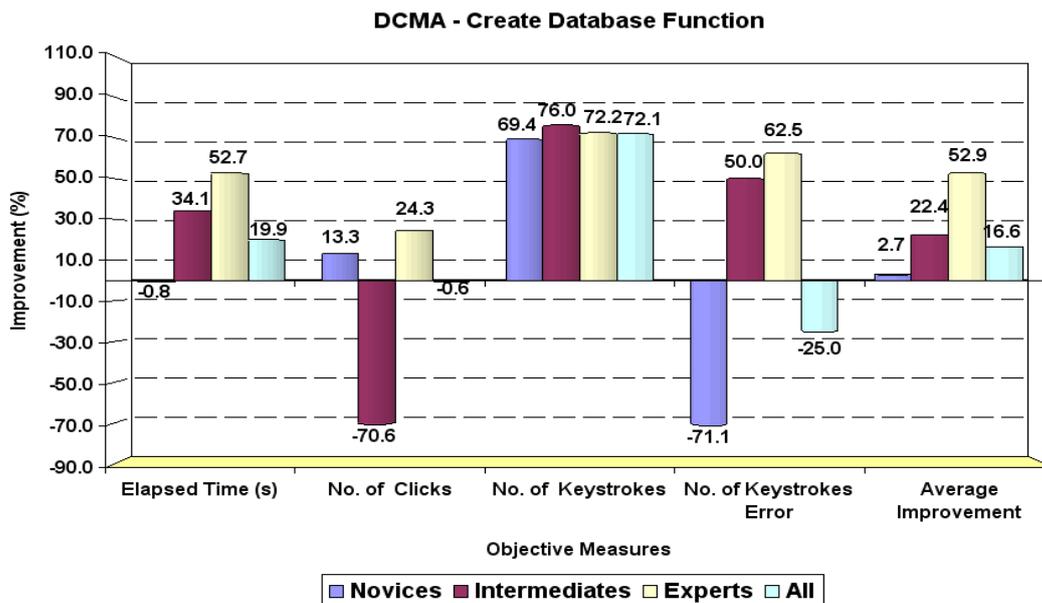


Figure 5.11. Chart with the improvements obtained when evaluating the Data Connector Model Assistant for the definition of the prototype of a database function

For the SFMA assistant, we proposed the testers to perform three steps. The first step consisted of creating a state with one slot related to the database. Here the testers had the option of creating the state using the proposal of automatic states with slots or the empty state template and then defining the slot. Figure 5.12 shows the improvement results during this step. In this case, the average improvement was 46.6%, and the elapsed time was reduced in average 56.7%. Here, the video recordings showed that all, except one of, the participants used the proposal of automatic states instead of using the empty state template.

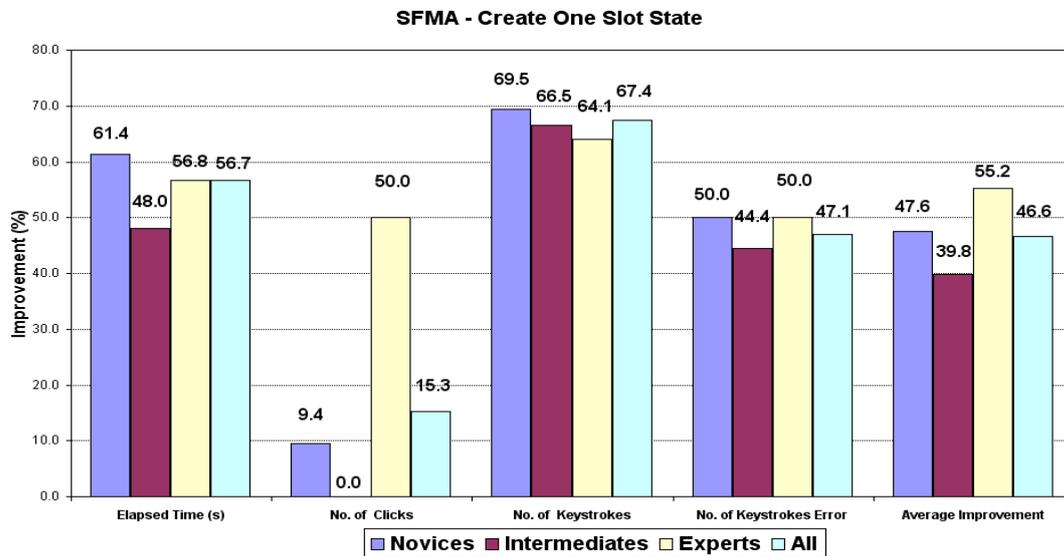


Figure 5.12. Chart with the improvements obtained when evaluating the State Flow Model Assistant for the creation of a state with one slot

For the second step, the evaluators were required to create a state with two slots, where both slots had to be set as mixed initiative. Besides, it was necessary to create a transition to other state. This step allowed the testers to check the automatic unification of slots to be requested using mixed-initiative dialogues and the automatic creation of an undefined state when it is referred as a transition state (i.e., top-down design).

The inspection of the recorded videos showed that most of the participants quickly created the state using the proposed states from the DMA. However, some of the novice testers did not use the proposal of states creating the state using the empty template and defining the slots one by one. Another fact we discovered inspecting the video recordings was that the evaluators were spending a lot of time reviewing the final state created using the AGP in order to check if it corresponded to the one specified by the evaluation. Although this behaviour is normal, we observed that for Diagen, since a lot of XML text was generated, they did not spend so much time in that revision. Therefore, the improvements should be greater in a normal case. In any case, according to Figure 5.13, the average improvement using the proposed accelerations in this assistant was 42%.

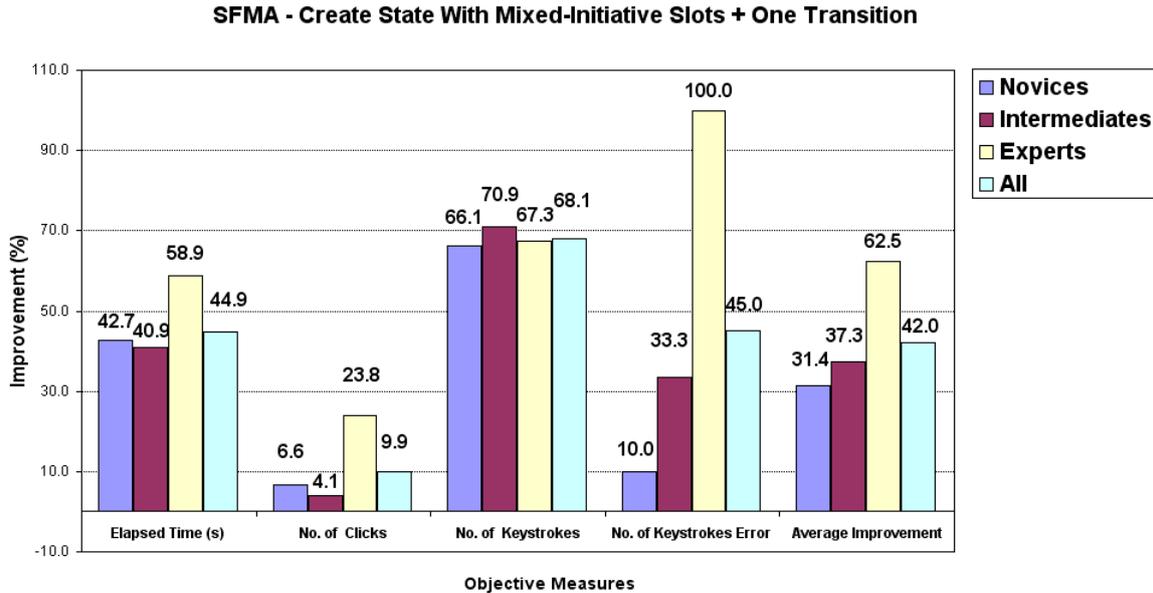


Figure 5.13. Chart with the improvements obtained when evaluating the State Flow Model Assistant for the creation of a state with mixed initiative slots and one transition

Finally, the third step was the creation of a connection between two states. This step allowed the testers to check some of the functionalities included in the graphical user interface (see section 4.4.1 and Figure 4.6, page 96). Although the final improvement was 38.1%, as we can see in Figure 5.14, it is obvious, considering the number of clicks, that for the participants it was not enough and should be simplified. This conclusion was also corroborated by the final subjective questionnaire, where most of the participants, and especially experts, agreed that the procedure to create the connections using the GUI should be changed by another one similar to the one existing in most graphics editors, i.e., connecting two blocks using anchor points. Although, they also considered that if the number of connections had been high then the proposed method would be better appreciated.

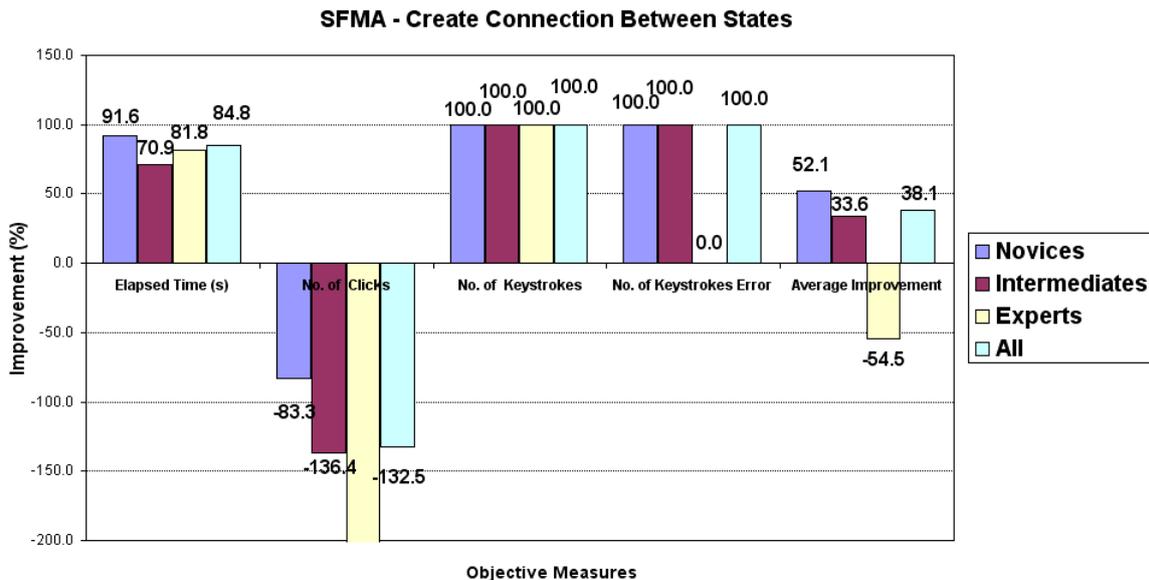


Figure 5.14. Chart with the improvements obtained when evaluating the State Flow Model Assistant for connecting two states

For the RMA assistant we proposed the evaluators three different tasks. The first one was the creation of a menu-based dialogue (see section 3.3.2, page 63). In this case, it is expected that the final user can answer, using language independent concepts, three different options: *PersonalInformation*, *GeneralInformation*, or *Transaction*. According to Figure 5.15, the participants were requested to fulfil the dialogue *GetTopLevelCategoryByName* defining the three different options. In order to get the user answer the dialogue *DGet_CategoryName* had to be selected. Then, depending on the user’s selection, which is stored in the slot *CategoryName*, the system will call different dialogues: *Personal_Information*, *General_Information*, or *Transactions*. Although, at first sight, this process looks complicated, using the proposals of dialogues (section 4.5.2, page 103) and the automatic *DGet* dialogues (section 4.5.1, page 101) the complete dialogue flow can be created in less than one minute.

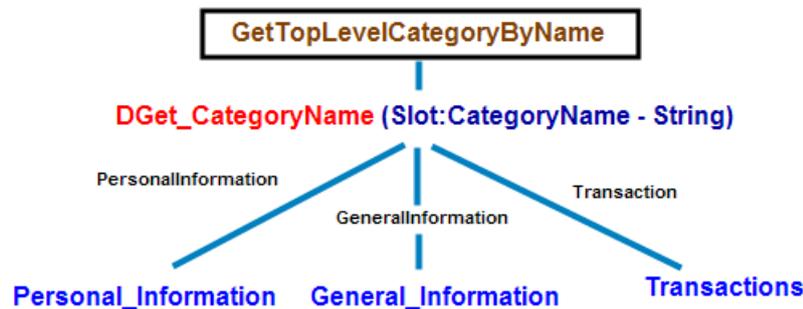


Figure 5.15. Proposed flow for the evaluation of a menu-based dialogue in the RMA

According to Figure 5.16, the average improvement was 81.5%, reducing the design time in more than an 88%. We also observe a high improvement in the number of keystrokes and keystroke errors since the XML code is more complex.

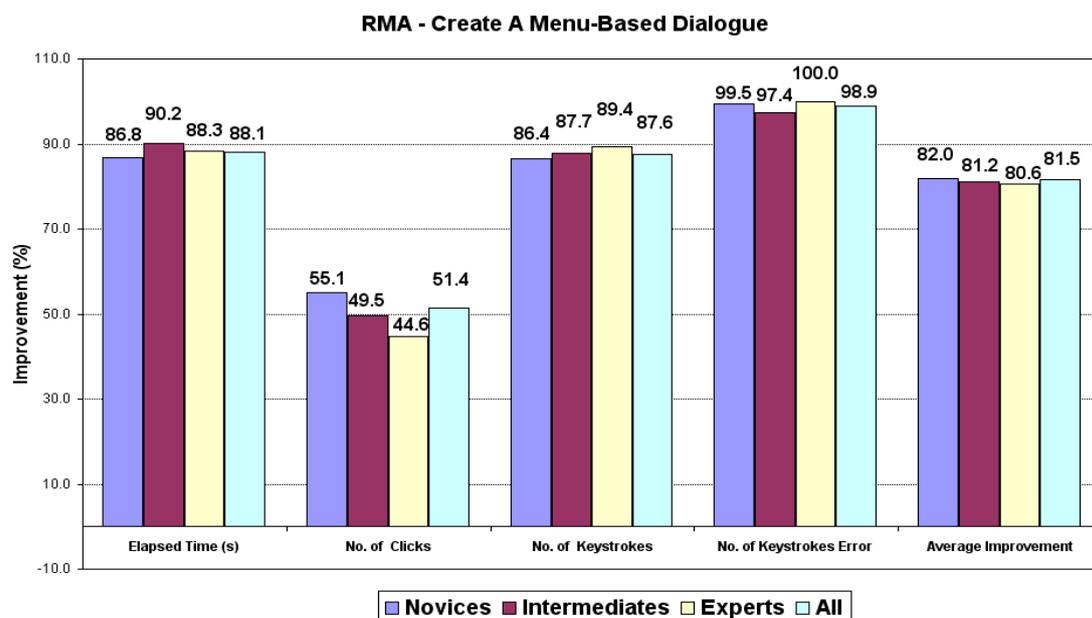


Figure 5.16. Chart with the improvements obtained when evaluating the Retrieval Model Assistant for creating a menu-based dialogue

Although the improvements of the AGP in this step are extremely high, we observe that the improvements in the number of clicks are not as remarkable. For this reason, we propose to improve the graphical interface in future releases of the platform. For instance, allowing the specification of several concepts at the same time or adapting the proposal window to the dialogue type (i.e., menu-based, sequence, while, etc.) in order to generate complex dialogue flows to be used as a template instead of using small items to build it.

In the second step, the participants were asked to create a dialogue (*SFM_GetLoansCategoryByName*) with over-answering and an IF-Then-Else condition. Figure 5.17 shows the flow proposed for this task. In this case, the designer has to use a DGet dialogue (*DGet_LoansType*) to fill in a compulsory slot (*LoansType*) and an optional slot (*HouseLoansType*). In the proposed flow, in case that the final user wants to obtain information about house loans, the compulsory slot will be set to the concept *HOUSE_LOANS*, and the system will jump to another dialogue (*GetHouseLoansSubCategoryByName*) where the other slot will be filled in (*HouseLoansType*) to save the subcategory of available house loans (e.g., for building, for repairing, for buying, etc). However, since in the current edited dialogue we define this slot as optional, it should be possible for the final user to fill it in using the over-answering capabilities of the DGet dialogue (*DGet_LoansType*). In case this slot is already filled in, it will not be requested in the posterior dialogue.

On the other hand, according to the figure, it is also possible that the final user does not want information about house loans but about other loans. In that case, the proposed flow is to call to a database access function (*PGetInformationByCategory*) that receives two input parameters (*InfoCategory* and *InfoSubCategory*) and returns a single string parameter. In the example, the first input parameter is set to 'LOANS', and the subcategory parameter is set to the concept stored in the compulsory slot. Then, the returned information is assigned to a local string variable (*InfoText*). Finally, the local variable is used as input to a DSay dialogue (*DSay_InfoText*) that plays the retrieved information to the final user. The goal of this step was to allow the evaluators to use the following accelerations: the dialogue proposals window (section 4.5.2, page 103), the automatic matching of arguments between actions (section 4.5.3, page 105), the procedure for including compulsory and optional slots (section 4.5.4, page 106), and the possibility of defining different programming structures (section 3.3.2, page 63).

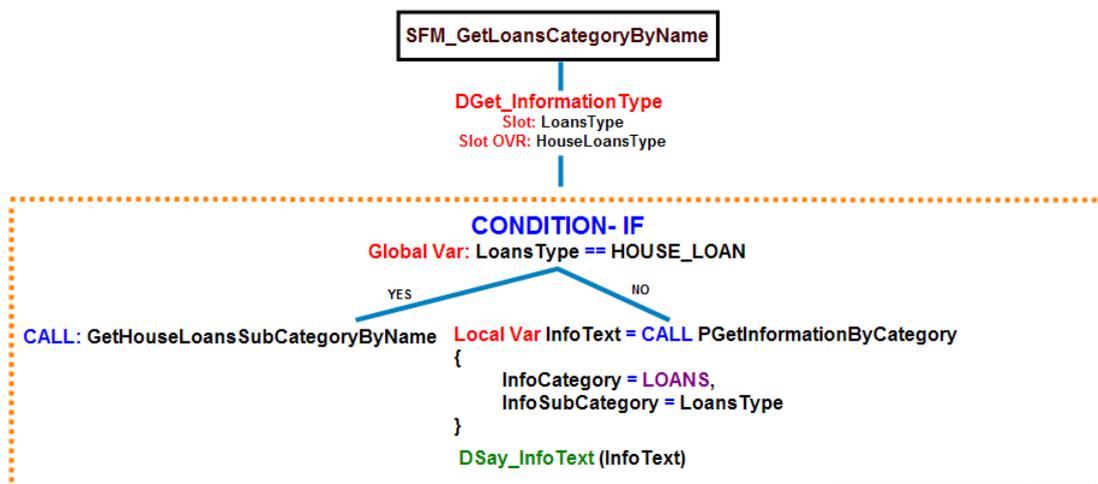


Figure 5.17. Proposed flow for evaluating a dialogue with over-answering and conditional actions

According to Figure 5.18, all the quantitative measures are positive with an overall average improvement of 88%. According to the Table E.1 in Appendix E, the average elapsed time when using Diagen was 1493 seconds (around 25 minutes), in comparison to the 140 seconds (2½ minutes) when using the AGP. In this case, the elapsed time is one order of magnitude greater than using the RMA. The main reasons for these values is the big complexity of the GDialogXML syntax when codifying the optional and compulsory slots and the low number of accelerations included in Diagen to codify conditional actions. In addition, the high number of keystrokes and keystroke errors confirms these reasons.

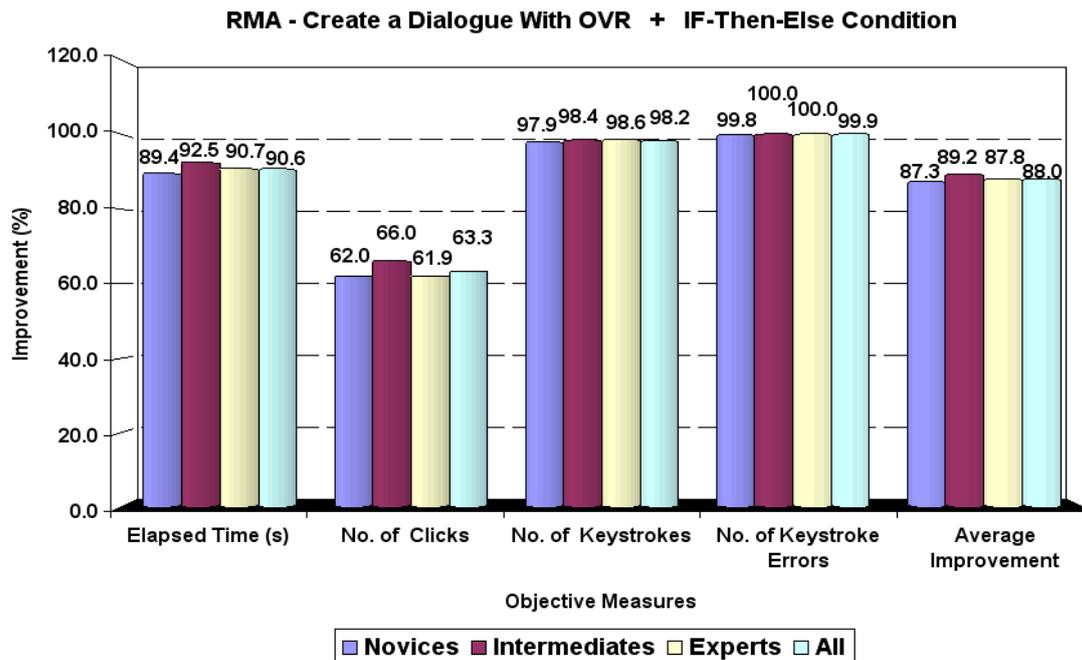


Figure 5.18. Chart with the improvements obtained when evaluating the Retrieval Model Assistant for creating a dialogue with over-answering and a conditional structure

The third task we proposed to the participants, see Figure 5.19, was the creation of a mixed-initiative dialogue (*DGet_MI_Template*) including the creation of a global variable (*Amount*) to save the information returned by a DGet dialogue (*GetTransactionAmount*). Then, the next step is to access the database using the function *PerformTransaction* that receives the credit and debit accounts and the transaction amount, returning if the transaction is performed or not. The next step in this demo version is to jump to a dialogue for asking the user if another transaction or service is desired. In a more realistic dialogue, a conditional loop should be required to guarantee that the transaction is performed or to notify the user if that is not the case and try again. However, this demo flow was enough to test the accelerations provided by the assistant for defining mixed-initiative dialogues (section 4.5.4, page 106), for matching variables (section 4.5.3, page 105), the dialogue proposals window (section 4.5.2, page 103), and for defining local/global variables.

According to the table Table E.1 in Appendix E, the average time for defining the flow using the AGP was 94 seconds (around 1½ minutes). In this case, the recording videos showed that the participants used all the available accelerations, although they spent some more time in creating the local variable since this was a new process that is not highly accelerated and that they had not practiced before. Unfortunately, we cannot provide a quantitative comparison between the AGP and Diagen, since the evaluation of this step with

Diagen could not be carried out. The reason was that at this time of the evaluation the testers were tired, and some of them reluctant to continue, after evaluating the previous task (i.e., dialogue with over-answering and conditional actions) with Diagen. Although we tried to motivate them, at the end we gave up because we thought that their results would not be accurate because of the lack of motivation, and also because this step is relatively similar to the previous one although it would have required more time to perform given its higher complexity. For that reason, we concluded that, in any case, the objective measures would show, again, the superiority of the platform over Diagen.

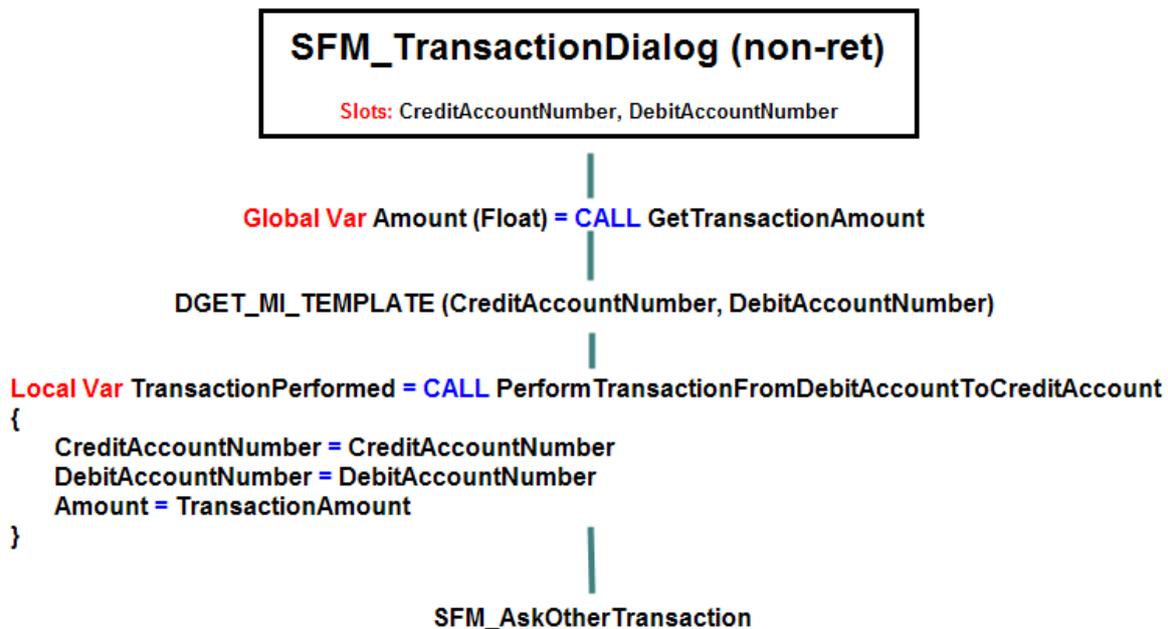


Figure 5.19. Proposed flow for a dialogue with mixed-initiative and the creation of a global variable

For the MERA-Speech assistant, we proposed a two steps task. The first step was the creation of a DSay dialogue for presenting a list of objects. In this case, the proposed dialogue provides information about the rates for selling or buying different international currencies. The average time used by the evaluators to configure this dialogue was only 89 seconds using only 17 clicks. Thanks to the different accelerations included in this assistant, the evaluators did not need to type in any information avoiding keystroke errors and reducing considerably the elapsed time. The last step was to automatically fill-in all the DGet dialogues included in the design. In this case, as we mention in section 4.6.2 (page 113), the assistant automatically proposes the strategy to fill in all the dialogues and automatically creates all the internal actions for the handling of errors in the speech recognition system. This acceleration allowed the participants to spend in average only 4 seconds, which they spent reviewing the proposed profile (simple or full) for each dialogue and clicking in the button to start the fill in process, in any case, without requiring any typewriting.

Finally, Figure 5.20 provides an overview of the average improvement considering all the tasks per assistant. As we can see, the accelerations proposed in this thesis produces an average improvement of 65.5% for defining the data model structure (DMA), a 16.6% for defining the prototypes of the database access functions (DCMA), 42.2% in the definition of

the finite state model of the application (SFMA), and a 84.8% for defining all the actions of each state of the dialogue flow (RMA). This way, we obtained an overall average improvement of 52.3% that corresponds to a 56.5% improvement in the elapsed time, 13.4% for the number of clicks, 84% in the number of keystrokes, and 55.2% in the number of keystroke errors. These results are consistent with the number and scope of the accelerations described in this thesis. We can also observe that the improvements were greater in the assistants where the more complex structures and actions are required; this way, we accelerate the design and guide the designer in the steps where it is more needed.

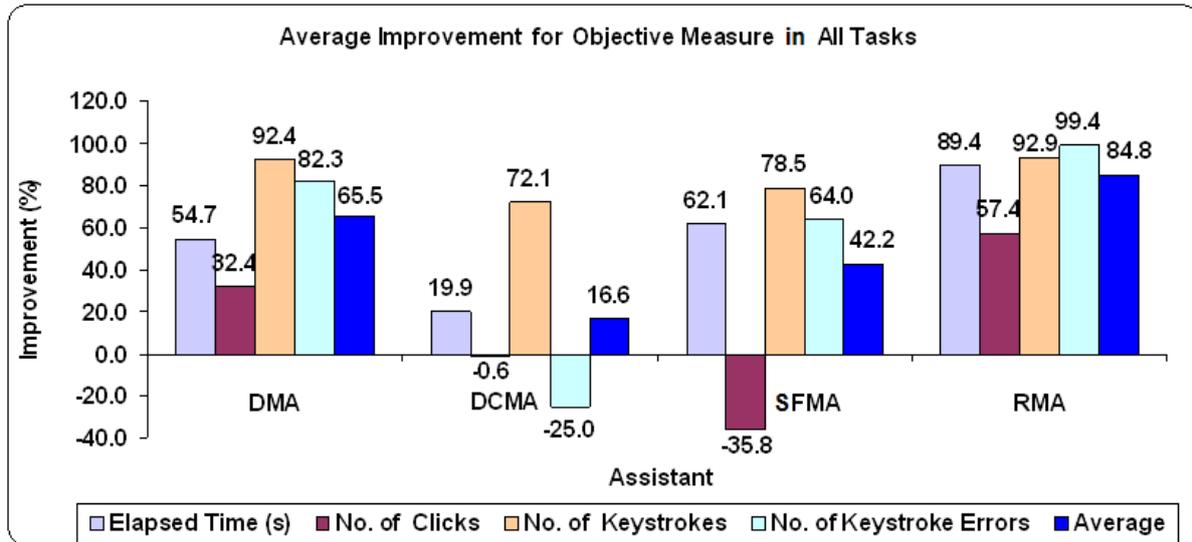


Figure 5.20. Chart with the average improvement by assistant considering all tasks

Although these results are good, we are sure that they could be better since after reviewing all the videos recorded during the evaluation, we found out that the evaluators spent too much time reviewing each step which incremented the elapsed time measured by the evaluation system. In addition, when we were carrying out the evaluation, we observed that the evaluators were quickly used to the Diagen interface since the mechanism for codifying the different steps and tasks in GDialogXML syntax were very similar (see section 3.4.6.3, page 72). In contrast, when evaluating the assistants of the AGP, the evaluators were required to learn a different methodology and accelerations at each assistant making more difficult its use. Finally, we also found that some of the evaluators did not use all the available accelerations of the assistants, but they resorted to other methodologies less straightforward.

5.2.3 Subjective survey

At the end of the two sessions of the evaluation, the evaluators were requested to fill-in a subjective survey about the different assistants and accelerations evaluated in this section. This survey was similar to the one used during the subjective evaluation described in section 5.1 (page 125), but including new specific questions about the accelerations (see Table 5.3), and including open questions to provide comments and suggestions.

In this survey, the participants were asked to answer a 4-item questionnaire per assistant with general questions about the appearance of the assistant, its level of intuitiveness, how quickly it was to learn it, and if the functionality of the assistant was enough. Then, they also answered to a 12-item questionnaire with specific questions about

the accelerations included in the AGP. In most questions users had to rate the relevant attribute or characteristic using a 10-point scale (1=minimum, 10=maximum).

Figure 5.21 shows the results of the general questions about the different assistants evaluated. In this case, we observed that these results are consistent, and better in all the cases, with the evaluation presented in section 5.1.2 (see Figure 5.6, page 130). The improvements are mainly due to the incorporation of new accelerations proposed in this thesis, together with the correction of some bugs, and the simplification of some procedures.

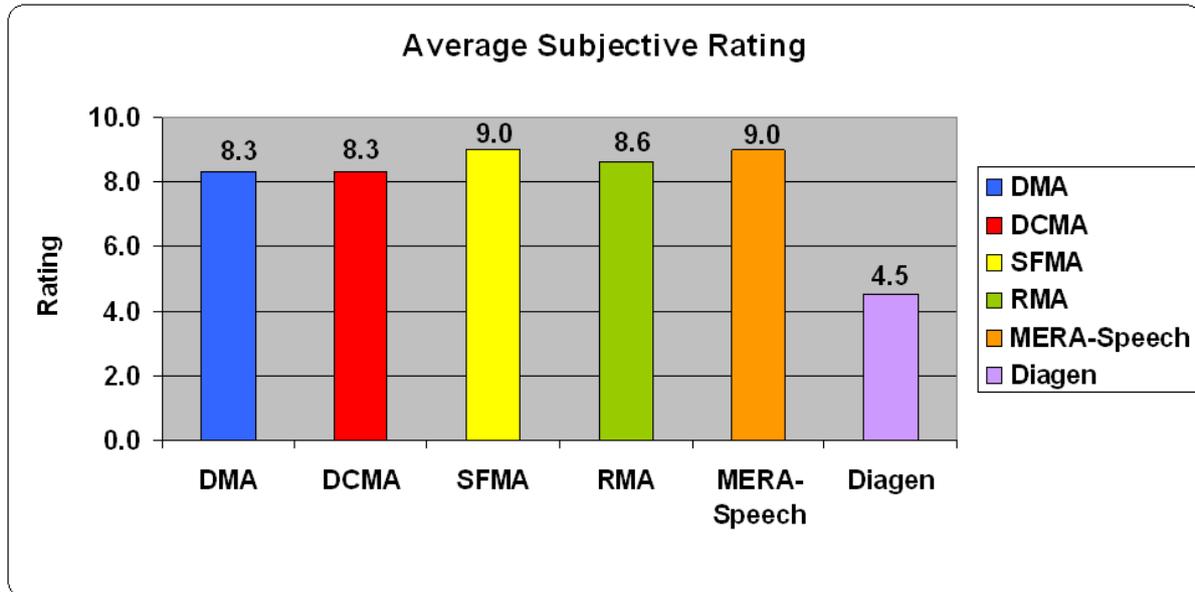


Figure 5.21. Chart with the results of the subjective evaluation for general questions about the assistants

On the other hand, Table 5.3 shows the questionnaire used to evaluate the main accelerations included in each assistant. The evaluated accelerations correspond to the ones used during the objective evaluation. This way, the participants had the possibility of using and experimenting with them, therefore their results are relevant since they are given in the heat of the moment. According to the table, all the accelerations were positively assessed with an average value of 9.0, with the maximum scores in the following accelerations: automatic generation of action proposals (section 4.5.2, page 103) and the easiness to define the state flow model (section 4.4, page 94). Finally, it is also important to highlight the last question since the participants showed an unquestionable preference for the AGP in contrast to using Diagen.

Finally, the survey included a section for comments and suggestions of the subjects with respect to the accelerations of each assistant. A summary of the main comments for each assistant is described next.

- **DMA:** A comment from one of the evaluators summarizes the most important acceleration in this assistant: “The creation of classes using the database information is extremely useful”. However, there are some aspects of the graphical interface that can be improved such as drag and drop capabilities, copy and paste, and the direct edition of the attributes using the boxes in the workspace instead of using buttons and boxes on the toolbar.

Assistant	Specific Questions About the Accelerations	Novice	Interm.	Expert	Av. All
DMA	Facility to create the data model structure using information from the DB (No useful at all – Very useful)	8.5	9.3	9	8.9
DCMA	Facility to define and test the database access functions (No useful at all - Very Useful)	8.8	9.3	9	9.0
SFMA	Facility to design the state flow model (Very Difficult - Very Easy)	9.8	8.7	9.5	9.3
RMA	Facility to define global or local variables (Very difficult - Very Easy)	8.0	8.0	9.5	8.3
	Facility to define dialogues with Mixed-Initiative (Very Difficult - Very Easy)	9.3	8.0	10	9.0
	Facility to create dialogues with Over-Answering (Very Difficult - Very Easy)	9.3	8.3	9.5	9.0
	Acceleration for passing arguments between actions (No useful at all - Very Useful)	9.3	8.0	9.5	8.9
	Automatic generation of action proposal for each dialogue (No useful at all – Very Useful)	9.5	10.0	10	9.8
	Tooltips to preview complex actions (No useful at all – Very Useful)	8.0	9.0	9	8.6
MERA-Speech	Acceleration for the management of the presentation of lists of objects (No useful at all - Very Useful)	9.3	9.3	8	9.0
	Quick configuration of error handling (nomatch, noinput, etc.) (No useful at all - Very Useful)	9.3	8.7	8.5	8.9
Diagen	Comparison of Speed development between using Diagen or the assistants of the AGP (0=I Prefer Diagen – 10=I Prefer AGP)	9.5	10.0	10.0	9.8

Table 5.3. Subjective evaluation results for specific questions about the accelerations

- **DCMA**: This assistant was considered easy to use since the menus were clear and simple. The most valuable accelerations were the automatic creation of SQL statements and the automatic proposal of data model classes/attributes and tables/fields when defining the arguments. One of the participants suggested to improve the process of defining the input/output parameters through a graphical interface instead of the text-based interface currently implemented.
- **SFMA**: The most valuable accelerations were the proposals of states from the classes and database functions, as well as the automatic unification of slots as mixed-initiative. One suggestion for this assistant was to implement a new mechanism for creating complex states in order to avoid the unification step

required when the designer selects two or more proposals of states in the auxiliary window. Another suggestion was to implement an easiest method, for instance using a right-click menu, to define the first state in the model instead of opening the state for edition (currently the designer has to specify it before saving the model since this information is compulsory to organize sequentially the states in the canvas). Finally, the procedure for connecting states was considered as non-intuitive at all, although it is easy, some evaluators have shown a preference for implementing a similar mechanism as in other graphical tools where two or more objects are connected using anchor points and drawing the connection line.

- **RMA:** In this assistant, the best accelerations were the window with the proposal of actions, the automatic creation of variables when passing arguments between actions, and the intuitive mechanism for creating conditional actions. The only suggestions were related with the graphical interface, for instance changing the position of some buttons in the assistants to make them easy to access, to avoid modal windows that prevent the designer from performing other tasks such as creating variables or editing other actions, since the mouse or keyboard focus cannot be redirected to other windows, etc.
- **MERA-Speech:** This assistant was highly appreciated since the steps when defining the presentation of lists were very clear and intuitive for all the participants. The automatic creation of the error handling was highly appreciated, although some evaluators requested the possibility of pre-visualizing and performing fine-tuning on the automatic flow created by the assistant.
- **Diagen:** Although this assistant does not include too many accelerations, the evaluators considered the process of creating any section of GDialogXML code through different pop-up windows, and pre-defined templates (see section 3.4.6.3, page 72), as easy, simple, and fulfilling most of their requirements. However, during the objective evaluation two main problems were detected: 1.) The information collected through the different pop-up windows can be lost in case of problems. Besides, sometimes it is confusing for the designer to follow the process of completing many nested items. 2) The templates were not enough for most designers, especially those unfamiliar with the GDialogXML syntax. The proposed solutions were 1.) The creation of a window listing all the actions that the designer checks using the pop-up windows. The action table would allow the designer to go back to an incomplete step before accepting the whole set of actions and GDialogXML code. 2) The templates have to be complemented by new pop-up windows and contextual help allowing more complex structures, and providing default or previously defined values. In addition, several other accelerations such as auto-completion, tags in colours, and reducing the number of times they need to delete and overwrite some default messages included in some of the templates.

5.3 Conclusions

With the objective of evaluating the performance of each of the assistants that make up the platform, as well as the accelerations proposed in this thesis we carried out a subjective and objective evaluation. In detail, we proposed for the objective evaluation the collection of different metrics such as elapsed time, number of clicks, keystrokes, and keystroke errors in order to measure the performance of each acceleration and assistant, as well as a parallel GDialogXML editor included in the platform. Then, a comparison between the assistants with accelerations and the GDialogXML editor was carried out. The results of this evaluation

confirm that the design time can be reduced in more than a 56% and the number of keystrokes in 84%. Besides, the subjective evaluation showed that all the accelerated assistants obtained a global score between 8.0 and 9.0. Therefore, both evaluations confirm the designer-friendliness of the platform, as well as its usability, and the contribution of the proposed accelerations to reduce the design time and to simplify the design process.

In relation with the accelerations included to create the data model structure the objective and subjective evaluation showed that the proposed accelerations helped to reduce the design in a 65.5%, to increase the subjective overall rating of the assistant from 8.1 to 8.3, and to score the new procedure for creating the data model structures with an 8.9.

In relation with the assistant that defines the prototypes of the database access functions, the objective evaluation showed an average improvement of 16.6% when compared to Diagen, which resulted in a reduction of 19.9% in the design time. The subjective evaluation also presents an increase in the overall rating from 7.9 to 8.3. The assistant for generating the SQL statements was rated with a 9.0.

In relation with the assistant where the state flow model is generated, the objective evaluation showed an average improvement of 42.2% when compared to Diagen, which resulted in a reduction of 62.1% in the design time and 78.5% in the number of keystroke errors. The subjective evaluation also presents an increase in the overall rating from 7.7 to 9.0. In this case, the improvements made to the graphical interface, and the state proposals and mixed initiative slots contributed the most to this new perception of the assistant.

In relation with the retrieval modelling assistant, i.e., the assistant that defines the actions to be done in each state, the subjective evaluation showed that the accelerations included in this assistant were scored in average with an 8.9, and the assistant with an 8.6. The objective metrics showed that the proposed accelerations contribute to reduce the design time by an 89.4%.

In relation with the assistant that defines the specific details for the speech modality, the automatic generation of the dialogue flow required for confirmation handling and the wizard for defining the dialogue flow for the presentation of lists of retrieved results after querying the backend obtained high subjective marks with an 8.9 and 9.0 respectively. Although the objective evaluation could not be completed with comparisons between the AGP and Diagen, given the high complexity of the GDialogXML models, the obtained results confirm that the accelerations are quite remarkable. Besides, the overall score given by the participants in the evaluation for this assistant was 9.0.

6 DEVELOPMENTS AND IMPROVEMENTS APPLIED TO THE RUNTIME SYSTEM

Besides all the previously described acceleration strategies applied to the design platform and its assistants, as well as all the efforts made to generate and provide the final service in different languages, throughout the thesis we also worked in improving two important components of the runtime system, namely an automatic language identification (LID) system and an automatic speech-to-sign language translation system. The former is important since it allows the identification of the final user's language at the beginning of the dialogue in order to load the correct acoustic and language models to be used during the call. The later is also important because it provides a mechanism for accelerating the specification of system prompts for different modalities, and to allow that the same service can be provided to different kinds of final users without requiring too much effort or specific knowledge from the designer. Both systems were selected in our effort to improve the multilingual and multimodal capabilities of the final service.

Unlike the acceleration strategies presented in the previous section, where different kind of heuristic, rule-base, and contextual information were used to accelerate the design, in this section the proposed improvements will be mainly based on using statistical information. This approach has the advantage that different automatic algorithms are applied in order to create the models used by the language identification system and the machine translation system without requiring too much participation from the designer, this way reducing also the design time and contributing to guarantee the multimodal and multilingual capabilities of the design and runtime platform.

This chapter is divided into two main sections. The first one presents an innovative and successful language modelling technique based on using an n-gram ranking of frequencies for providing long-span information to a state-of-the-art LID system based on the PPRLM technique. In this work, two main objectives were considered: a) to accurately identify the language, and b) to use a reduced audio segment in order to identify the language as soon as possible. These factors are quite important since they help to guarantee the user satisfaction, to provide reliable speech recognition results, and to allow a quick setup of the service.

The second section describes a successful technique for improving an automatic machine translation system that can be used to translate the previously defined prompts for the service into an animated representation in the sign language. In this case, the goal was the creation of an adaptation technique that can be applied to the target language model that is used by the machine translation system during the process of generating the translated sentence, in order to guarantee that the candidate sentences are grammatically and syntactically correct. Our main contribution lies on using retrieved counts from online resources to adapt the original counts of the n-grams that appear in the target language, as well as the creation of the list of n-grams to be queried on the Web. Apart from being innovative, the proposed technique is especially interesting for applications where the training data is scarce to estimate properly the language model used during the decoding process for translating sentences from one language into the other.

6.1 Language Identification System

As mentioned in the state of the art (section 2.3, page 36), many methodologies for language identification have been proposed. Among them, the most widely used and successful technique is PPRLM. In PPRLM, several parallel phone recognizers and language models are used during the identification process. This section describes a novel approach for language identification based on a text categorization technique, namely an n-gram frequency ranking, and the incorporation of different acoustic information. In our system, we use a parallel phone recognizer, the same as in PPRLM, but instead of using as phonotactic constraints the traditional n-gram language models (PPRLM_{NG}) we use a new language model which is created using a ranking with the most frequent n-grams (PPRLM_{RANK}), keeping only a fraction of them. The objective is to select the n-grams that are more discriminative between languages. Then the distance between the ranking for the input sentence and the ranking for each language is computed, based on the difference in relative positions for each n-gram. The final objective of this ranking is to be able to model reliably a longer span than the obtained using the traditional n-gram models used in PPRLM_{NG}, namely 5-gram instead of trigram, because for using this ranking it is not necessary to use any kind of smoothing technique, so it requires less training data for a reliable estimation. The results show that this approach overcomes PPRLM_{NG} thanks to the inclusion of n-grams of higher order (i.e., 4-gram and 5-gram) in the classifier. Besides, two alternatives are shown: a ranking with absolute values for the number of occurrences, and a ranking with discriminative values. In addition, we have also combined this technique with other sources of information (feature vectors in our classifier) such as acoustic scores at sentence and phoneme level, as well as phoneme duration, which in previous research experiments have also proved to be relevant and provide additional improvements.

6.1.1 System Description

6.1.1.1 Database corpus

The database used for carrying out the experiments described in this thesis is a continuous speech database, which consists of very spontaneous conversations between controllers and pilots. For speech recognition it is a very difficult database, noisy and very spontaneous, as in “*Lufthansa four two seven nine start up approved clear to Frankfurt standard departure somosierra one echo three six left squawk one zero two three report parking position*”. A big drawback with this database is that all speakers are native Spanish. Therefore, many of them do not reflect all the phonetic variations in English, and they mix Spanish words for names, airports, greetings, and goodbyes even when the rest of the sentence is in English.

The database consists of approximately 9 hours of speech for Spanish, consisting of 4998 sentences, and 7 hours for English corresponding to 3132 sentences. Since the proposed technique has to be applied to dialogue systems, only sentences with a minimum of 0.5 s and a maximum of 10 s (with an average duration of 4.5 s) were considered. This restriction is important since many of the techniques reported in the literature take advantage of using longer sentences (e.g. with an average duration of 30 s).

6.1.1.2 Parallel phone recognizer followed by language modelling (PPRLM)

As mentioned in the state-of-the-art (see section 2.3.1, page 39), the most widespread and successful technique for LID is Parallel Phone Recognition followed by Language

Modelling (PPRLM) [Zissman, 1996], which classifies languages based on the statistical characteristics of the allophone sequences. The technique consists of two stages: First, a phone recognizer takes the speech utterance and outputs the sequence of allophones corresponding to it, without using any phonotactic constraint during the Viterbi decoding. Then, the sequence of allophones is used as input to a language model (LM) module that scores the probability that the sequence of allophones corresponds to the language. In order to recognize different languages the system is made up of N parallel phone recognizers and M language models modelled for the M different languages to recognize. In theory, PPRLM allows having phone recognizers modelled for languages different from the languages that have to be identified. However, the performance of the system increases if there is a match between the input language and the language of the acoustic models, because in that case, it is possible to model explicitly the phonetic variations of each language.

During the classification step, the unknown utterance is transcribed using each of the N parallel recognisers. Then a score is calculated for each of the N transcriptions using the M language models as represented in Figure 6.1. In our system, N and M are equal to two, corresponding to the Spanish and English languages.

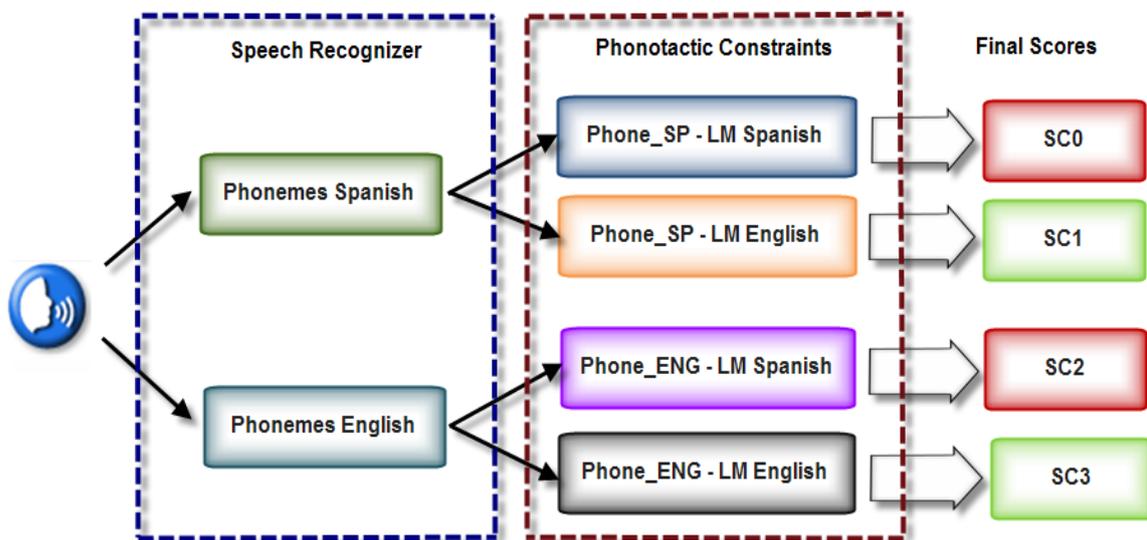


Figure 6.1. PPRLM scores used for the LID system

Finally, an overall score is calculated through an average between both scores obtained for the same language using Eq. 6.1. In this way, the target language is the one that obtains the highest overall score.

$$SC_{_SPA} = \frac{SC0 + SC2}{2}; SC_{_ENG} = \frac{SC1 + SC3}{2}$$

Eq. 6.1

Regarding the language models, in most PPRLM systems it is frequent to use back-off n-gram models or independent n-gram language models of different orders interpolated linearly using the deleted interpolation technique (see Eq. 6.2) In this equation, weights α_1 , α_2 , and α_3 correspond to the unigram, bigram and trigram model respectively; α_0 stands for the zero-gram or equally distributed probability model. Although this technique alleviates the data sparcity problem, it still exists, as described in [Cordoba et al, 2003][Cordoba et al, 2006c], where a PPRLM system using trigrams, PPRLM_{NG} in our notation, performs slightly better than another one using 4-grams.

$$S(w_{i-2}, w_{i-1}, w_i) = \alpha_3 \cdot p(w_i | w_{i-1}, w_{i-2}) + \alpha_2 \cdot p(w_i | w_{i-1}) + \alpha_1 \cdot p(w_i) + \alpha_0 \cdot p_0$$

Eq. 6.2

One problem with this approach was that the weights of the n-grams, i.e. α_0 , α_1 , α_2 , and α_3 , were difficult to integrate in our approach, the Gaussian classifier which is described in the following section, as the scores for the unigram, bigram, and trigram models were independent in our feature vector. For that reason, in [Cordoba et al, 2007b] a different formulation was proposed (see Eq. 6.3).

$$\sigma_i^{Final} = \frac{\sigma_i^{Original}}{\lambda}$$

Eq. 6.3

In this case, instead of multiplying each feature by its weight in the distance measure, it was proposed to divide the variance of the Gaussian distribution of each score by a corresponding λ_i weight (Eq. 6.3). For low values of λ_i , the final variances are increased, so the distances are smoothed (which is good for less discriminative features). This smoothing weight is quickly adjusted with good results and allowed us to maintain the models independent in our feature vector.

6.1.1.3 Gaussian classifier for LID

In spite of all its advantages, as it is described in [Ramasubramaniam et al, 2003], the general PPRLM approach has a flaw: there is the possibility of having a different bias in the log-likelihood score for the languages considered. This is even more evident when the phone recognizers have a different number of units, therefore the language with fewer units will have higher probabilities in the LM score (think of the unigram case), and the classifier will tend to select that language. Since in our system we have 61 allophones for English, and 49 allophones for Spanish, in [Cordoba et al, 2006c] we proposed to use a Gaussian Mixture Model (GMM) classifier instead of the usual decision formula applied in PPRLM. The advantage of the Gaussian classifier is that it does not suffer from the bias problem, as it does not use an absolute discriminant function. Besides, we can increase the number of Gaussians, in order to better model the distribution that represents our classes, following the classical HMM modelling approaches (i.e., Gaussian splitting and Lloyd re-estimation after each splitting with a maximum of iterations).

Therefore, with all the scores provided by every LM in the PPRLM module, we prepare a score vector. Now, the recognized language is not the one with the largest average score. Instead, the distance between the input vector of LM scores and the Gaussian distributions for every language is computed, and the distribution that is closer to the input vector is the one selected as identified language. Therefore, the LID problem can be treated as a conventional N-class classification problem (for N languages) in the score space of dimension D (D scores considered in our system). Each class is represented by a Gaussian density $N(\mu_l, \Sigma_l)$, where μ_l and Σ_l are the mean and covariance of class l. They are estimated from the training data of P vectors of class l using Eq. 6.4. Here, we have considered the weighted Euclidean distance (Σ diagonal) instead of a full covariance matrix as we are aware of the insufficient training data to estimate the full matrix.

$$\mu_l = \frac{1}{P} \sum_{p=1}^P x_{l,p} \quad \Sigma_l = \frac{1}{P} \sum_{p=1}^P (x_{l,p} - \mu_l)(x_{l,p} - \mu_l)^t$$

Eq. 6.4

A test utterance is classified as language l^* based on its score vector v using Eq. 6.5.

$$d(v, \mu_{l^*}, \Sigma_{l^*}) \leq d(v, \mu_l, \Sigma_l), l = 1, \dots, N \quad \text{Where } d(v, \mu_l, \Sigma_l) = (v - \mu_l)^t \Sigma_l^{-1} (v - \mu_l)$$

Eq. 6.5

Finally, an important conclusion presented in [Cordoba et al, 2006c] is that, instead of using absolute values for the scores, it is important to use differential scores as a normalization mechanism. This differential score is the difference between the score obtained by the LM of the same language of the acoustic models considered (Spa-Spa or Eng-Eng) and the score obtained by the other ‘competing’ language(s): SC0 – SC1 and SC3 – SC2 in Figure 6.1. In this case, we applied it to unigram, bigram and trigram separately, with six features in total that are listed in Table 6.1

Phonemes-SPA	SCO-SC1 for unigram
	SCO-SC1 for bigram
	SCO-SC1 for trigram
Phonemes-ENG	SC3-SC2 for unigram
	SC3-SC2 for bigram
	SC3-SC2 for trigram

Table 6.1. Differential score vector

Table 6.2 shows the results using PPRLM_{NG} and a Gaussian classifier for the optimum combination of weights. This system will be the baseline for the remaining of this chapter. More details about the baseline system can be found in [Cordoba et al, 2003] and [Cordoba et al, 2006c]. However, to show the benefits of the proposed technique we will first show the results using a mono-Gaussian classifier and afterwards we will present the results for the multi-Gaussian classifier.

Gaussians	LID Error rate (%)
1	3.69
2	3.74
3	3.75
4	3.75

Table 6.2. LID error rate results for PPRLM_{NG}

6.1.1.4 General conditions of the experiments

The LID system uses a front-end with PLP coefficients derived from a mel-scale filter bank (MF-PLP), with 13 coefficients including c_0 and their first and second-order differentials, giving 39 parameters per frame. For the phone recognizers, we have used context-independent continuous HMM models. For Spanish, we have considered 49 different allophones and, for English, 61 different allophones. All models use 10 Gaussians densities per state per stream.

In order to increase the reliability of the results presented in the next sections, we performed a cross-fold validation, dividing all the available material in 9 subsets. Using in each pass:

- 4 blocks for estimating the acoustic models and the Gaussian distribution for the LMs and the ranking,
- 3 blocks for estimating the language models for PPRLM, the n-gram ranking or the n-gram probabilities, and the Gaussian distribution for the acoustic scores and duration,
- 1 block for the test-set and parameter fine-tuning,
- 1 block for the validation set.

Besides, this block distribution is consistent with the one proposed in [Cordoba et al, 2006c] where it was checked that the acoustic models and the Gaussian mixtures for the LMs can be trained using the same data as it does not participate in the LM estimation, and the same can be applied for the mixture estimation of acoustic scores with the data used to train the LMs. Moreover, this distribution provides more robust and effective models since the Gaussian mixtures are estimated on different sets of the training data allowing that the distribution of the scores matches better the one that will be obtained in the evaluation set.

Another issue that we also considered when splitting the database into these blocks was that since this database consists of conversations between controllers and pilots, and that the same controller uttered a large group of sentences which were sequential in the database until there was a shift change, it was possible that the system made some kind of speaker modelling instead of language modelling, i.e., the models could be capturing the specific characteristics of a predominant controller instead of the language used. For that reason, we decided to create the lists using a random selection procedure, namely Fisher-Yates, which assures the maximum dispersion in speaker selection. In previous experiments, [Cordoba et al, 2006c], this random selection resulted in an important improvement of 16.6% in average and 5% in the minimum, showing that in fact there was some sort of implicit speaker modelling.

6.1.2 Proposed Technique: n-gram Frequency Ranking

As we have described before, one of the main problems with our current PPRLM_{NG} system is the existence of data sparsity problem and the inability to use longer span information (n-grams of higher order than trigram). In spite of all our efforts to alleviate these problems, they remain as we described in [Cordoba et al, 2006c], where our system using trigrams performs better than when we using 4-grams. In this section, we describe our efforts to reduce both problems [Cordoba et al, 2007a].

6.1.2.1 Base system: all n-grams in one ranking

In [Cavnar and Trenkle, 1994], an interesting technique that combines local information (n-grams) and long-span information (collected counts from the whole utterance) is described. In general terms, during training the technique proposes the creation of a ranked template with the N (typically 400) most frequent n-grams (up to n-grams of order five) of the character sequences in the train corpus for each language sorted by occurrence and then orthographically in case two or more n-grams contain the same occurrence (e.g., positions 10 and 11 in Figure 6.2).

During the evaluation, a dynamic ranked template is created for the phoneme sequence of the recognized sentence following the same procedure. Then a distance measure is applied between the input sentence template and each language dependent template previously trained. The distance for a given ranking T is calculated using Eq. 6.6.

$$d^T = \frac{1}{L} \sum_{i=1}^L \text{abs}(\text{pos } w_i - \text{pos } w_i^T)$$

Eq. 6.6

Where L is the number of n-grams generated for the input sentence. If an n-gram does not appear in the global ranking (meaning that it has not appeared in training or it is not in the top n-grams selected) it is assigned a maximum distance: the size of the ranking. The selected language is the one that presents the higher correlation between templates (i.e., the lower distance).

	Trained Template	Sentence Template	Distance
Most Frequent	1. f (254)	1. w (25)	1
	2. w (235)	2. f (21)	1
	3. ai (211)	3. ai (20)	0
	...	4. t_h (15)	7
	10. n (150)
	11. t_h (150)	11. n (10)	1

Least Frequent	26. 'ai_n_e (89)	30. dZ (7)	no match = max
	27. ei_h_o_n (80)	31. 'ai_n_e (7)	5

			Sum = score

Figure 6.2. Example and calculation of distance score using a ranking of n-grams as proposed by [Cavnar and Trenkle, 1994]

Figure 6.2 shows an example of one of the templates created in our system for English and the template created for the unknown sentence. Although this technique is very simple, it provides good results for language recognition of written texts (up to 93%, depending of the length of the sentence to be recognized and the size of the template).

In order to start with our experiments we decided to follow the original proposed technique and use it as our baseline system. Here, we used the same input as PPRLM_{NG}: the sequence of allophones generated by the phone recognizer, but with the difference of using this ranking instead of the interpolated n-gram based LM module considered in PPRLM_{NG}. In addition, the combination of PPRLM_{NG} with the proposed technique, PPRLM_{RANK}, allowed us to include longer span information (4-gram and even 5-gram) into the language model. However, as the information used by the classification system is very similar to PPRLM_{NG}

(i.e. frequency of occurrence of n-grams), we were afraid that results could be at most similar, but as we will show in the next sections, the proposed technique improves clearly $PPRLM_{NG}$. In any case, we will also have four independent rankings, as we had four LMs in $PPRLM_{NG}$ (see Figure 6.1), trained using the same blocks used to train the acoustic models (section 6.1.1.4). In this first experiment, L was also set to the top 400 n-grams, as proposed by [Cavnar and Trenkle, 1994], but the LID rate was 7.5% error rate, which is higher than the obtained using $PPRLM_{NG}$ (Table 6.2), therefore we decided to research other alternatives.

Our first variation from [Cavnar and Trenkle, 1994] is the application of what we called the “golf score”. As the number of occurrences of the n-grams in the input sentence is very low, most n-grams have the same number of occurrences and should have the same position in the ranking. It is the same as a ranking in golf (the sport): all players with the same number of strokes share the same position. It meant a relative improvement of 5% (from 7.5% to 6.4%). Figure 6.3 shows an example of the modification applied to the original template using the proposed “golf” score.

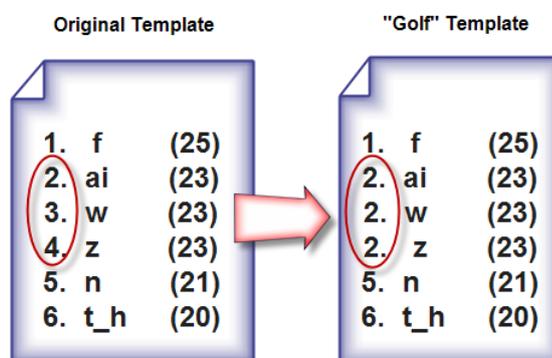


Figure 6.3. Example of the modification of a ranking template using the “golf score”

Then, we applied our Gaussian classifier to these scores as we did with $PPRLM_{NG}$ using the differential scores described in section 6.1.1.3. In Figure 6.4, we can see the results, with the optimum number of Gaussians, of using the ‘golf’ ranking and varying the ranking size. In this case, our best results are obtained using rankings with 3,000 n-grams.

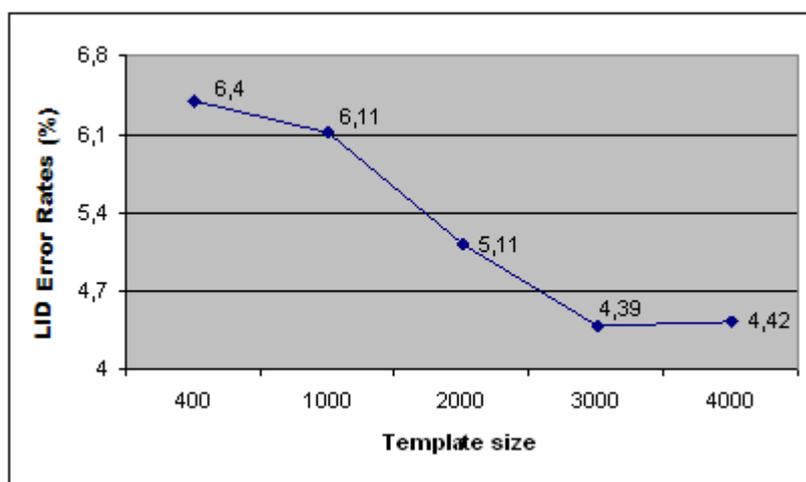


Figure 6.4. LID error rate results varying the ranking size and using the ‘golf’ ranking

6.1.2.2 N-gram specific rankings

After examining the ranking templates created following the original proposal, we arrived to the conclusion that they were not optimum for our task since the top positions were always devoted to the unigrams and bigrams that we already knew that were less discriminative for language identification. For instance, in PPRLM_{NG}, the optimum result is always obtained applying the highest weight to the trigrams. Therefore, we decided to have different rankings for each n-gram order. This introduces a small change in our Gaussian classifier, we now have 10 features in our vector, the same 6 features in Table 6.1 for unigram, bigram, and trigram, and 4 new features for 4-gram and 5-gram.

Table 6.3 shows an example of the n-gram specific ranking used in this section. The table shows the first five positions in the ranking for the given n-gram order and the number of occurrences of the respective sequence of phonemes. The table also shows an example of the “golf technique” in the four-grams $w_^\wedge_n_t$ and $w_^\wedge_n_z$, highlighted in yellow. In this case, both n-grams occur 165 times and both are ranked in position 3, however the next 4-gram, $t_u_:_w_^\wedge$, is ranked in position 5, since there is no rank 4.

Rank	Phoneme N-Grams	Counts	Rank	Phoneme N-Grams	Counts
[1-gram]			[2-gram]		
1	n	4293	1	w_^\wedge	1531
2	r	4125	2	^\wedge_n	1279
3	k	3301	3	z_'\i	701
4	s	2984	4	t_u:	600
5	t	2883	5	.._n	544
[3-gram]			[4-gram]		
1	w_^\wedge_n	1125	1	u: w_^\wedge_n	268
2	u: w_^\wedge	324	2	z_'\i_r_ou	208
3	'i_r_ou	302	3	w_^\wedge_n_t	165
4	z_'\i_r	269	3	w_^\wedge_n_z	165
5	ou_w_^\wedge	196	5	t_u: w_^\wedge	161
[5-gram]					
1	t_u: w_^\wedge_n	133			
2	w_^\wedge_n_t_u:	115			
3	w_^\wedge_n_z_'\i	112			
4	n_t_u: w_^\wedge	104			
5	^\wedge_n_t_u: w	103			

Table 6.3. Example of an n-gram specific count-based ranking for English.

According to Table 6.4, the ranking size for unigram and bigram is different between languages. For that reason, it was necessary to include an additional normalization in the distance measure. In this case, we divide it by the number of items in the set for that n-gram order. In the table, the difference in the ranking size for the unigrams corresponds to phonemes that occur in one language but not in the other (i.e. compare the size between ENG_ENG and CAST_CAST). However, even when using the same phoneme set (i.e. ENG_CAST and CAST_CAST) there are differences, this is mainly due to specific phonemes that users cannot easily pronounce in the other language.

Phone Set Lang. Model	Number of Elements			
	ENG		CAST	
	ENG	CAST	ENG	CAST
[1-gram]	61	48	56	49
[2-gram]	2550	1842	2836	1977
[3-gram]	3000	3000	3000	3000
[4-gram]	3000	3000	3000	3000
[5-gram]	3000	3000	3000	3000

Table 6.4. Ranking size for the different n-grams and languages

In Table 6.5, we can see the results using this approach. Now, the ranking size presented in the table is the maximum allowed in the ranking creation algorithm, because for unigram and bigram there are less than 3000 different items. The results show a high improvement with this approach, which is due to the fact that there is more information as more n-grams are considered globally in the system, and that the new information is estimated more reliably. Nevertheless, the performance of this method is still below PPRLM_{NG} results (Table 6.2).

Ranking size	One ranking + 'golf'	Specific ranking + 'golf'	Improvement (%)
1000	6.11	4.46	27.0
2000	5.11	3.96	22.5
3000	4.39	3.82	13.0
4000	4.42	3.96	10.4

Table 6.5. LID error rate results with n-gram specific ranking

6.1.2.3 Measure of separation between distributions

One of the main difficulties we suffered in the LID experiments is that they were very time consuming, as we had to modify many weights (one for each n-gram) and we were using the cross-fold validation technique. During the experiments with PPRLM_{NG}, we had the same problems, however at that time we just considered up to trigrams, but with the new proposed technique, that we were confident we could use up to 5-grams, the combination of weights was increased. Therefore, we decided to restrict the weights considered in the experiments using, for each feature, information regarding the separation between the pdf distributions for each candidate language. In order to do it, we applied Eq. 6.7 that is usually used in feature selection algorithms to reduce the dimensionality of the input vectors. Moreover, from our experience, we have also found that there is a very strong correlation among this measure of separation between the Gaussian distributions and the results in LID. When using the formula, a high value means that the feature is especially discriminative between languages. In the equation, μ_1 and μ_2 are the mean values for the feature considering Spanish and English input sentences respectively, and σ_1 and σ_2 are the respective covariances.

$$\frac{\mu_1 - \mu_2}{\sigma_1^2 \sigma_2^2}$$

Eq. 6.7

Table 6.6 shows the separation which is obtained with $PPRLM_{NG}$ and $PPRLM_{RANK}$ for each n-gram considered.

Order	$PPRLM_{NG}$	$PPRLM_{RANK}$
trigram	10.57	8.42
4-gram	-	6.41
bigram	8.54	5.35
5-gram	-	4.43
unigram	3.17	2.06

Table 6.6. Comparison of feature discrimination between $PPRLM_{NG}$ and $PPRLM_{RANK}$

Therefore, the discriminative power of $PPRLM_{NG}$ is higher, especially for the trigram, but the nice thing of $PPRLM_{RANK}$ is that we also obtain a nice discrimination with the 4-gram and 5-gram that could not be used in $PPRLM_{NG}$ due to insufficient training data. In addition, the table helps to understand and confirm the results in Table 6.2 and Table 6.5, where $PPRLM_{NG}$ outperforms our ranking proposal.

6.1.2.4 N-gram discriminative ranking

After considering the discriminative power offered by higher order n-grams (see Table 6.6) and that the specific n-gram rankings were not working as well as we could expect, we considered another solution. In this case, inspired in the work of [Nagarajan and Murthy, 2004], where better LID results could be obtained using the most discriminative units, we decided to give more relevance (higher positions) in the ranking to the items that are actually more specific to the language that is being identified, i.e. n-grams with a high frequency in one language but with zero or low frequency in the competing languages. In order to do it, we applied document/topic classification techniques.

The first option was to use the *tf-idf* (see section 2.2.1.3, page 31), which has been widely used for topic classification in many different fields. However, as we only have two languages, it only discriminates n-grams that appear in one language but not in the other, and very few n-grams in our database fulfil that. Therefore, we proposed a variation to *tf-idf*. In this case, after the original global rankings are created, we have the number of occurrences of each n-gram: $n_1(w)$ = occurrences of n-gram w in the current language, and $n_2(w)$ = occurrences of n-gram w in the competing language (it would be the average in the competing languages to extend this measure to multiple languages).

$$N_1 = \sum_{\forall w: w \in T_1} n_1(w) \quad N_2 = \sum_{\forall w: w \in T_2} n_2(w)$$

Eq. 6.8

Being, in Eq. 6.8, N_1 the sum of all occurrences for the current language and N_2 for the competing language, and T_1 and T_2 the ranking templates created for each language. As the number of total occurrences will be different for each language and n-gram order, before the subtraction a normalization is needed to have comparable amounts (see Eq. 6.9).

$$n'_1(w) = \frac{n_1(w) \times N_2}{N_1 + N_2} \quad n'_2(w) = \frac{n_2(w) \times N_1}{N_1 + N_2}$$

Eq. 6.9

Using these normalized values we considered several alternative formulae with the same philosophy as *tf-idf* for the final number of occurrences considered for the ranking (which we will call n_1'') and studied the separation between the Gaussian distributions for each language obtained using each formula before diving into the LID experiments (see Table 6.7). To summarize, only the average separation for all 5 n-grams is presented. First, we proposed a purely discriminative solution Eq. 6.10:

$$n_1''(w) = \frac{n'_1(w) - n'_2(w)}{n'_1(w) + n'_2(w)}$$

Eq. 6.10

According to the table, with this formulation there is a nice improvement over the non-discriminative ranking. The next proposal was to include an item frequency term in the formula (see Eq. 6.11

), but in this case we lost part of the discriminative power. For that reason, we decided to reduce the effect of the first term by taking its logarithm or the square root obtaining higher improvements.

$$n_1''(w) = n'_1(w) \times \frac{n'_1(w) - n'_2(w)}{n'_1(w) + n'_2(w)}$$

Eq. 6.11

Formula	Average Separation
Original – no discriminative	6.15
$n_1''(w) = \frac{n'_1(w) - n'_2(w)}{n'_1(w) + n'_2(w)}$	6.75
$n''(w) = n'_1(w) \times \frac{n'_1(w) - n'_2(w)}{n'_1(w) + n'_2(w)}$	6.48
$n_1''(w) = \log(n'_1(w)) \times \frac{n'_1(w) - n'_2(w)}{n'_1(w) + n'_2(w)}$	6.82
$n_1''(w) = \text{sqrt}(n'_1(w)) \times \frac{n'_1(w) - n'_2(w)}{n'_1(w) + n'_2(w)}$	7.01
$n_1''(w) = n'_1(w) \times \frac{n'_1(w) - n'_2(w)}{(n'_1(w) + n'_2(w))^2}$	7.13

Table 6.7. Average feature discrimination (several formulas)

Finally, the last formula in Table 6.7, proposed by my advisor, provided the best classification power, probably because it normalizes the values between 1 and -1. Here one means that the n-gram appears in the current language but not in the other competing ones ($n_2'=0$), indicating that it is especially relevant for that language; -1 meaning just the opposite ($n_1'=0$), so the n-gram does not appear in the current language.

Table 6.8 shows that the discrimination for the ranking trigram is now very similar to the $PPRLM_{NG}$ trigram, and with a light improvement for the 4-grams. In addition, results are better than those obtained in Table 6.6.

6.1.2.5 Threshold

One factor that has to be also addressed with these measures is that they are very prone to overtraining, i.e. n-grams that just appear once or twice in training for one language and never for the competing language(s) will be at the top position of the list, even though they are probably irrelevant.

Therefore, we decided to apply a threshold: if $(n_1' + n_2') < \theta_{ng}$, send the item to the last position in the ranking. After applying a greedy algorithm, the optimum thresholds were $\theta_{1g}=6$, $\theta_{2g}=4$, $\theta_{3g}=3$, $\theta_{4g}=2$, $\theta_{5g}=2$ for unigram, bigram, trigram, 4-gram, and 5-gram respectively.

Order	$PPRLM_{NG}$	Discriminative $PPRLM_{RANK}$
trigram	10.57	9.71
4-gram	-	6.61
bigram	8.54	7.12
5-gram	-	4.25
unigram	3.17	2.19

Table 6.8. Comparison of feature discrimination between $PPRLM_{NG}$ and discriminative $PPRLM_{RANK}$

6.1.2.6 Results using the discriminative $PPRLM_{RANK}$ system

In Table 6.9 (third column), we can see the LID error rate results using the proposed technique (in parenthesis the relative improvement in comparison to $PPRLM_{NG}$). For simplicity, the table only presents the results for a ranking size equal to 3000. We can see that, even with one Gaussian, results are better than $PPRLM_{NG}$. However, when the number of Gaussians grows the improvements are lower. Probably, the reason is that we now have a 10-feature vector instead of six with $PPRLM_{NG}$, so it is more difficult to estimate reliably several Gaussians with our training database. The improvement over $PPRLM_{NG}$ for the best results is **13.0%** (3.21 versus 3.69). Over the non-discriminative ranking, it is **16.0%** (3.21 versus 3.82) (see Table 6.5).

Gaussians	PPRLM _{NG}	Discriminative PPRLM _{RANK}
1	3.69	3.21 (13.0%)
2	3.74	3.22 (13.9%)
3	3.75	3.32 (11.5%)
4	3.75	3.24 (11.4%)

Table 6.9. LID error rate results for PPRLM_{NG} versus discriminative PPRLM_{RANK}

6.1.2.7 Longer span of the technique

We also checked the relevance of including 4-grams and 5-grams in the proposed technique for LID. According to Table 6.10, considering only up to 4-gram or up to trigram results are worse than using all n-grams. Therefore, we are clearly taking advantage of longer span information using the proposed technique.

Order	LID error rate results
Up to 5-gram	3.21
Up to 4-gram	3.36
Up to trigram	3.65

Table 6.10. LID error rate results for including incrementally long-span information

6.1.2.8 Accumulative improvements

Figure 6.5 shows the accumulative improvements obtained with the modifications proposed in this thesis (from section 6.1.2.2 to 6.1.2.5) to the basic ranking described in section 6.1.2.1 which clarifies the relevance of each alternative proposed.

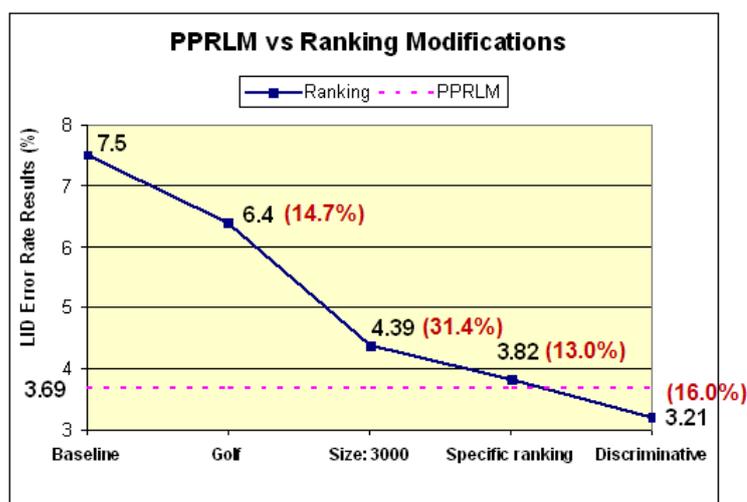


Figure 6.5. Accumulative LID error rates reductions over the original ranking technique

6.1.3 Incorporation of additional information

One drawback with PPRLM is that the basic technique only takes into account information regarding the allophone sequence. As we mentioned in the section 2.3 (page 36), other techniques such as the “GMM tokenizer” provide a good performance using both acoustic and “sequence of sounds” information. Unfortunately, the addition of new information cannot be included using the basic PPRLM formula (see Eq. 6.1). However, this, and other sources of information, can be included taking advantage of the Gaussian classifier.

Since previous experiments from my advisor, published in [Cordoba et al, 2006c] and [Cordoba et al, 2007b], proved that the fusion of $PPRLM_{NG}$ and acoustic scores provided better results using different feature vectors in the Gaussian classifier, we decided to check if the fusion of the n-gram discriminative ranking proposed in this thesis with these acoustic scores could also be used to improve the system. This section describes in detail the new sources of information as proposed in the papers, the results reported in the papers using them alone or in combination with $PPRLM_{NG}$, as well as the experiments we did in the thesis for combining them with our proposed technique.

6.1.3.1 Inclusion of the sentence acoustic score

The first additional information is the acoustic score at the sentence level, normalized by the number of frames, obtained by the phone recognizers for both languages. Since the values of the acoustic score were not homogeneous at all and the estimated distributions had a big overlap between the languages that we wanted to classify, all experiments using those scores provided worse results. The solution proposed in the papers was to use the difference between the score for the Spanish phone recognizer and the score for the English phone recognizer as feature value (“differential scores” according to section 6.1.1.3, page 152). This approach can be extended to several languages using Eq. 6.12. Here, the differential score is the difference between the acoustic score of the current language and the average acoustic score from the n-1 different languages. In this case, the overlap between the estimated distributions reduced drastically.

$$Ac_Score_i - \frac{1}{n-1} \sum_{\forall j \neq i}^n Ac_Score_j$$

Eq. 6.12

In order to estimate the acoustic score distributions we utilized the same set used to train the language models because those sentences have not been used to train the phone models. This way we trained the Gaussian distributions for allophone sequence scores and acoustic scores separately, as they use different training data for the estimation (it is very similar to the treatment of different feature vectors in HMM models).

Gaussians	$PPRLM_{NG}$	$PPRLM_{NG}$ + Sent. Acoustic	Discriminative $PPRLM_{RANK}$	Discriminative $PPRLM_{RANK}$ + Sent. Acoustic
1	3.69	3.21 (13.0%)	3.21 (13.0%)	2.79 (13.1%)

Table 6.11. Comparison of LID error rate results for including the sentence acoustic score to the $PPRLM_{NG}$ and the discriminative $PPRLM_{RANK}$ systems

As we can see in Table 6.11 (fifth column), the results are outstanding, obtaining even better results than the fusion of $PPRLM_{NG}$ + sentence acoustic scores (third column), which provided improvements (given in the column in parenthesis) in average of 13.0%. On the other hand, when comparing $PPRLM_{NG}$ with the proposed technique $PPRLM_{RANK}$ + sentence acoustic scores the results are in average 24.4% better. Finally, the inclusion of acoustic information is even better than only using the discriminative $PPRLM_{RANK}$ technique (fourth column). In this case, the relative improvements (given in parenthesis) are 13.1% in average when compared to the third column. Finally, we also did experiments with the fusion of all three models (i.e. $PPRLM_{NG}$ + Discriminative $PPRLM_{RANK}$ + sentence acoustic) and we obtained a minimum of **2.66%**, which is a nice additional improvement (4.7%) over “Discriminative $PPRLM_{RANK}$ + acoustic”.

6.1.3.2 Inclusion of the acoustic score for each phoneme

After obtaining promising results with the incorporation of acoustic information at the sentence level, we now considered to add the acoustic score for each individual phoneme, also proposed in previous papers, taking into account that this feature could also have a strong variation depending on the language. Using the Gaussian classifier, we modelled the distribution for the acoustic score of each phoneme. For each input sentence, we have its corresponding sequence of phonemes using the Spanish and English phone recognizers. Figure 6.6 shows the procedure to create the vector with the acoustic score for each phoneme.

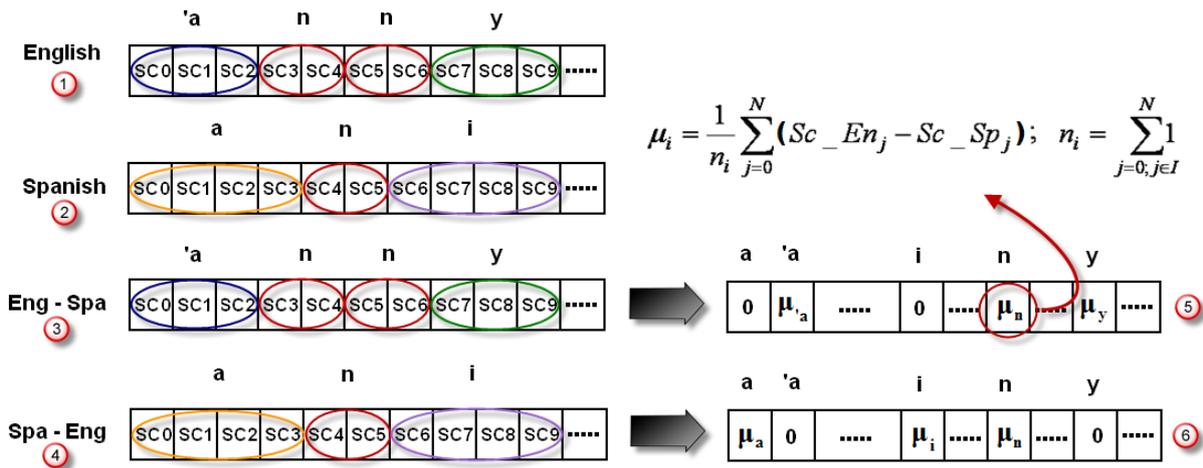


Figure 6.6. Example of the procedure to create the vector with the acoustic score for each phoneme

In the example, number 1 shows the N frames of a recognized sentence using the English phoneme set. In the figure, the first three frames correspond to phoneme /‘a/. Number 2 shows the recognized sentence using now the Spanish phoneme set. Observe that since we are using different phoneme sets the number of frames assigned to each phoneme may be different as well as the identity of the phonemes itself. Considering that in the previous experiments we found that the “differential scores” approach was a better option because these scores have a strong variability, we decided to apply the same concept here. In order to do it, a new score is calculated for each frame as $SC_{new_i} = SC_{English_i} - SC_{Spanish_j}$. In the figure, numbers 3 and 4 correspond to the new calculated differential scores considering each language. It is important to highlight that although the score changes for each frame, the recognized phonemes remain without change (i.e., the same number of frames and sequence).

The next step is to create the score vector in number 5. In this case, we have to compute the average differential score (μ_i in the figure) for each phoneme appearing in the sentence

(averaging the new calculated differential score over all frames belonging to that phoneme) obtaining a feature vector with as many features as the total number of phonemes in the system for all languages. Obviously, phonemes not appearing in the sentence do not contribute to the final score in the classifier (in the figure they obtain a 0 value). It is also important to explain that each average value for the new vectors, numbers 5 and 6, are different even when they share the same number of dimensions, i.e., total number of phonemes in both sets, since the number of frames assigned to a given phoneme and the average differential scores are different for each language.

In order to reduce the size of the feature vector it was necessary to group some allophonic variations. The first approach was to consider 34 different phonemes for each language, but even then, the vector was too large, so the estimations were unreliable. Then, it was decided to apply a feature selection algorithm to reduce the dimensionality, by using the same approach described in section 6.1.2.3 (page 158), applying Eq. 6.7, in order to determine which features were more discriminative. The next step was to test the system using the first 24, 30, and 35 features, keeping at the end 30 features as the optimum. To get an idea of the information provided by the incorporation of the acoustic score for each phoneme, we should mention that using the same equation, the discrimination for the sentence acoustic score is 6.84, whereas for the 30 features of the acoustic score for each phoneme it ranges from 3.52 to 0.54. Finally, according to the reported papers, using only this feature it is possible to obtain an 8.17% error rate in LID that is slightly better than using the sentence acoustic score (8.20%), see Table 6.12.

6.1.3.3 Inclusion of the duration for each phoneme

Finally, following the previous reported work, we also considered that phoneme duration could also be different depending on the input language and included this feature vector into the Gaussian classifier. In this case, we modelled the Gaussian distribution for the average duration of each phoneme in the system. For each input sentence, we computed the average duration for each phoneme resulting in that the feature vector had as many features as the number of phonemes.

However, it is important to mention that this feature presents an important problem because the duration produced by the recognizer is too difficult to normalize. The “differential scores” approach would be to subtract the average duration for the competing language, but, as the phoneme sets are different for each language, this subtraction is not possible. Therefore, two normalizations were considered: a) Subtract the average phoneme duration of the competing language; b) Subtract the phoneme duration of the competing language for the phoneme that had the largest part in common with the current one, so it will be the most probable “competing” phoneme. In the previous experiments (b) was the best option. Later, the feature vector was reduced using the same feature selection technique as in the previous section, keeping 22 features as the optimum value. Unfortunately, the reported results using only this new feature vector showed that the LID error result was 32.31%, which was clearly a bad result confirming that there were still normalization problems, see Table 6.12. In spite of these previous results, we decided to test this feature in combination with our technique.

6.1.3.4 Individual features

When mixing several sources of information differences are less evident. Therefore, we will first show in Table 6.12 the results of each source independently. The results show that the n-gram ranking provides a 13.0% improvement over PPRLM and prove that the phoneme

duration is the least discriminative feature, which can be due to problems in the normalization.

Gaussians	PPRLM _{NG}	Disc. PPRLM _{RANK}	Sentence Acoustic	Phoneme Acoustic	Phoneme Duration
1	3.69	3.21	8.20	8.17	32.31

Table 6.12. LID error rate results for individual feature vectors

6.1.3.5 Combination of all features

Finally, Table 6.13 shows the results when combining several feature vectors and the relative improvements over the PPRLM_{NG} and the PPRLM_{RANK} base systems considering also the discriminative power of each feature according to Table 6.12.

Row	Feature vectors	LID Rate	Improvements PPRLM _{NG}	Improvements PPRLM _{RANK}
1	PPRLM _{NG} + Sentence Acoustic	3.21	13.0%	-
2	PPRLM _{NG} + Phoneme Acoustic	3.13	15.2%	-
3	PPRLM _{NG} + Phoneme Duration	3.68	2.7%	-
4	PPRLM _{NG} + both Acoustics	3.05	17.3%	
5	PPRLM _{NG} + both Acoustics + Duration	3.25	11.9%	
6	PPRLM _{RANK} + Sentence Acoustic	2.79	-	13.1%
7	PPRLM _{RANK} + Phoneme Acoustic	2.78	-	13.4%
8	PPRLM _{RANK} + Phoneme Duration	3.08	-	4.1%
9	PPRLM _{RANK} + both Acoustics	2.67	-	16.8%
10	PPRLM _{RANK} + both Acoustics + Durations	2.59	-	19.3%
11	PPRLM _{NG} + PPRLM _{RANK}	2.85	22.8%	11.2%
12	PPRLM _{NG} + PPRLM _{RANK} + Sentence Acoustic	2.66	27.9%	17.1%
13	PPRLM _{NG} + PPRLM _{RANK} + both Acoustics	2.54	31.2%	20.9%
14	PPRLM _{NG} + PPRLM _{RANK} + both Acoustics + Durations	2.52	31.7%	21.5%

Table 6.13. LID error rate results for feature vector combinations

From the results reported in Table 6.13, we can see in rows 1 and 2 that in the PPRLM_{NG} system the acoustic information at phoneme level provides a better improvement than the information at sentence level as the individual results predicted. According to row 3, the fusion of PPRLM_{NG} and phoneme duration only provides a low improvement as expected, although it is worse than the obtained using the acoustic information. Considering rows 4 and 9, in both systems, PPRLM_{NG} and PPRLM_{RANK}, both acoustic scores improve the identification rate. This is expected since both scores are complementary. However, when we compare rows 5 and 10 we can see the results for combining all the scores; in this case, the PPRLM_{NG} systems obtains slight worse results when compared with only the acoustic information. However, for the PPRLM_{RANK} the results are slightly better.

The results reported in rows 6 to 9 show that the fusion of the PPRLM_{RANK} and the acoustic features provides similar improvements like in PPRLM_{NG}. However, these improvements are a bit lower probably because they begin from a better system. If we consider now row 11, we observe that the fusion of PPRLM_{NG} and PPRLM_{RANK} provides a nice improvement, which is surprising, as they use the same source of information, i.e. the n-grams. Observing rows 12 and 13, we can see that the fusion of PPRLM_{NG} + PPRLM_{RANK} + Acoustic scores provides further improvements, which shows again that they all provide complementary information. Finally, row 14 shows that our best system is the fusion of all the scores acoustic and durations and both PPRLM systems.

6.1.3.6 Evaluation of the Multi-Gaussian Classifier

After finishing the definition of the proposed technique and all the features that make up the input vector to the backend classifier, we decided to check if we could get higher improvements using a Multi-Gaussian classifier applying different number of mixtures to the different features proposed previously and growing up from one to four mixtures. The reason for growing up to only four mixtures was to avoid overtraining (see Table 6.2) and to reduce the time required to train the models and optimize the weights for all the features in the input vector.

Table 6.14 shows the LID error rate results using a multi-Gaussian classifier, where the number of mixtures has been selected to provide the maximum LID rate. From rows 1-5 we show the results for each feature separately. It is interesting to observe that in most of the cases, the optimal number of Gaussians is 1, except for the phoneme acoustic and phoneme duration where the high variability of these features is better modelled by a higher number of Gaussians. In general, the improvements using the multi-Gaussian classifier are small when compared to the mono-Gaussian classifier. The only exception is for the phoneme duration where the multi-Gaussian allows a considerable improvement of 21.5%.

From rows 6 to 8 we can see the results for combining the PPRLM_{NG} system with the different acoustic and duration features. In this case, we observe that increasing the number of Gaussians provides improvements to the LID rate. In any case, none of these combinations provides better results than the ones obtained using the PPRLM_{RANK} approach proposed in this thesis.

In rows 9-10 we observe the results for combining the PPRLM_{RANK} and the acoustic and duration features. In this case, the multi-Gaussian classifier provides slightly improvements as in the previous case.

Finally, we decided to check if the combination of our proposed technique with the traditional PPRLM system could provide any improvements when increasing the number of Gaussians. Row 12 shows that unfortunately the improvement is almost insignificant. However, it confirms that both systems provide complementary information.

Row	Feature vectors	Optimal Num. of Gaussians	LID Rate MonoGaussian	LID Rate MultiGaussian
1	PPRLM _{NG}	1	3.69	3.69 (0%)
2	Sentence Acoustic	1	8.20	8.20 (0%)
3	Phoneme Acoustic	4	8.17	7.85 (3.97%)
4	Discriminative PPRLM _{RANK}	1	3.21	3.21 (0%)
5	Phoneme Duration	4	32.31	25.37 (21.5%)
6	PPRLM _{NG} + Sentence Acoustic	4-1	3.21	3.06 (4.67%)
7	PPRLM _{NG} + Phoneme Acoustic	3-4	3.13	3.10 (0.98%)
8	PPRLM _{NG} + Phoneme Duration	1-4	3.68	3.49 (5.16%)
9	PPRLM _{RANK} + Sentence Acoustic	3-2	2.79	2.91 (-3.94%)
10	PPRLM _{RANK} + Phoneme Acoustic	1-4	2.91	2.70 (2.88%)
11	PPRLM _{RANK} + Phoneme Duration	1-4	3.08	3.09 (-0.32%)
12	PPRLM _{NG} + PPRLM _{RANK}	4-2	2.85	2.84 (0.35%)

Table 6.14. LID error rate results using the Multi-Gaussian classifier

6.1.3.7 Analysis of Confidence Intervals

In order to assess the reliability of the LID rates presented in the previous sections we analysed them considering the 95% confidence interval given by Eq. 6.13.

$$\frac{Interval}{2} = \pm 1,96 \sqrt{\frac{p(100-p)}{n}}$$

Eq. 6.13

In this equation, p is the obtained LID rate and n is the number of sentences used in the test set. Since we have used a cross-fold validation, n is equal to the total number of available sentences in the database, in this case, 8130. Unfortunately, even using the cross-fold validation the total number of sentences is small, therefore the confidence intervals are quite big.

In this section, we will compare the most interesting combinations of the presented systems with the different sources of information considered, checking if there is an overlap in the confidence intervals or not. Since the multi-Gaussian experiments did not provide significant improvements, we will only compare the mono Gaussian systems.

Figure 6.7 shows the comparison of LID error results obtained using the PPRLM_{NG}, PPRLM_{RANK}, and the combination of both systems. In this figure, we can see that there is an overlap in the results for PPRLM_{NG} and PPRLM_{RANK}. For that reason, we cannot be sure that both systems are significantly different. We have to take into account that the total number of sentences is small, even though we use the cross-fold validation technique.

However, the last bar shows that when both techniques are combined the new system provides better results than considering $PPRLM_{NG}$ alone. In this case, the new system provides better results thanks to the contribution of the information provided by the $PPRLM_{RANK}$ system, which, as we have already said is a very nice contribution of the proposed technique.

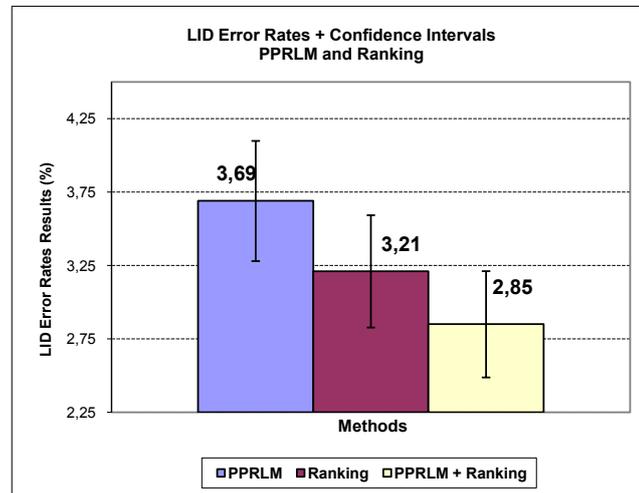


Figure 6.7. LID error rate results and confidence intervals considering the $PPRLM_{NG}$, $PPRLM_{RANK}$, and the fusion of both systems

6.1.4 Conclusions

In this section, we have demonstrated that the n-gram Frequency Ranking approach proposed in this thesis can overcome $PPRLM_{NG}$ thanks to the longer span that can be modelled. In our first attempt, we started from a widely known technique for LID on written text. Later, we introduced several new ideas and important changes to the original technique in order to obtain considerable improvements when compared with a state-of-the-art LID system, i.e. $PPRLM_{NG}$.

In relation with the ranking technique the first conclusion we arrived to was that the ranking size should be increased as much as possible when that number of different n-grams is available. In our case, it was set to 3000. Another conclusion is that instead of using a common ranking for all n-grams it should be better to use n-gram specific rankings. Moreover, we have also demonstrated that the selection of the most discriminative n-grams to create the rankings provides better results, which are able to overcome $PPRLM_{NG}$ (13% relative improvement). In this case, we have proposed different formulations in order to normalize the results and to provide better results than the widely used *tf-idf* metric.

On the other hand, we have also demonstrated that the measure of separation between pdf distributions is a good tool to reduce the number of experiments and to anticipate which features are going to be actually discriminative for the LID task.

Taking advantage of the incorporation of a Gaussian classifier to discriminate between the languages, we also included different acoustic and duration based information that resulted in an additional improvement (16.8% improvement) in the LID rates, showing that all the features proposed provide complementary information. Nevertheless, when

considering the discriminative power of each of the acoustic and duration information included, we found that the acoustic score for each phoneme is a slightly better feature than the sentence acoustic score, and the phoneme duration is the less contributing. Surprisingly, the combination of the baseline PPRLM_{NG} system, the new proposed technique, and the acoustic and duration information, all combined, provided a new improvement of 31.2% when compared with the baseline system and with a 20.9% when compared with using only the discriminative ranking.

On the other hand, the results with a multi-Gaussian classifier did not produce considerable improvements when compared with the mono-Gaussian classifier. This is mainly due to the size of our current database and possibly to the homogeneity of some of the features used.

Finally, the analysis of confidence intervals showed that the combination of the proposed discriminative ranking system with the traditional PPRLM_{NG} systems provides significant improvements when compared to a system that only uses the traditional approach. Unfortunately, when we compared the best system using the PPRLM_{NG} system (i.e. combined with acoustic information) and our best LID system (a combination of all sources, the PPRLM_{NG}, the proposed PPRLM_{RANK}, plus acoustic and duration information), we did not obtain a significant difference, although it has to be taken into account that the database size is small.

6.2 Automatic Translation of Dialogue Prompts into the Sign Language

This section describes in detail the work done for improving the multimodal and multilingual capabilities of the runtime system developed for this thesis, and for allowing the design of new kind of services and target population without too much effort for the designer. Specifically, a statistical machine translation system is proposed in order to allow the automatic translation of the prompts of the system into an animated representation using a 3D avatar. The goal of this new system is to extend the number of modalities supported by the platform but, especially, to allow the design of services for a new target population, i.e. deaf people. Since, in general, the designers do not know the sign language, the language of deaf people, this translation system alleviates this problem providing an automatic way to convert previously defined written or spoken prompts into their representation in the sign language.

Unfortunately, as we have pointed in the state-of-the-art (section 2.4.4, page 50), currently most of the commercial and research automatic translation systems are based on statistical approaches that require a huge number of parallel texts to be trained. However, as we have also indicated, currently most of the available sign language (SL) corpora consist of only a few hundred sentences that are too small or too general for training purposes. In addition, it is too hard to find such kind of corpus available from online content.

Therefore, since we wanted to include in this thesis an automatic translation system and considering that we also had a small corpora to train the translations models, we decided to address the problem of data sparseness, proposing a new language model adaptation technique. In our proposal, reported in [D'Haro et al, 2008], the idea is to create a new language model (LM) that adapts the original target LM used by the Machine Translation (MT) system. In our experiments, the source language corresponds to the Spanish sentences defined as system prompts in the platform, and the target language corresponds to the

translation of the Spanish sentences into a written representation of the corresponding Spanish Sign Language (LSE) signs (i.e., glosses).

The experiments reported in this section were done using a restricted domain corpus that consists of written sentences containing information about procedures and requirements needed to apply or renew the National Identity Document (DNI).

6.2.1 Runtime System for the Speech-to-Sign Language Translation System

Figure 6.8 shows the main components of the speech-to-Sign Language system used in this thesis. The system is made up of three main modules. The first module is a state of the art recognizer developed in our group [Cordoba et al, 2001] that captures the acoustic signal of the spoken utterance of the non-deaf users, and produces the sequence of words with the maximum a posteriori probability given by the acoustic and language models. The acoustic models provide the knowledge about acoustics, phonetics, microphone and environment variability, gender and dialect differences among speakers, etc.

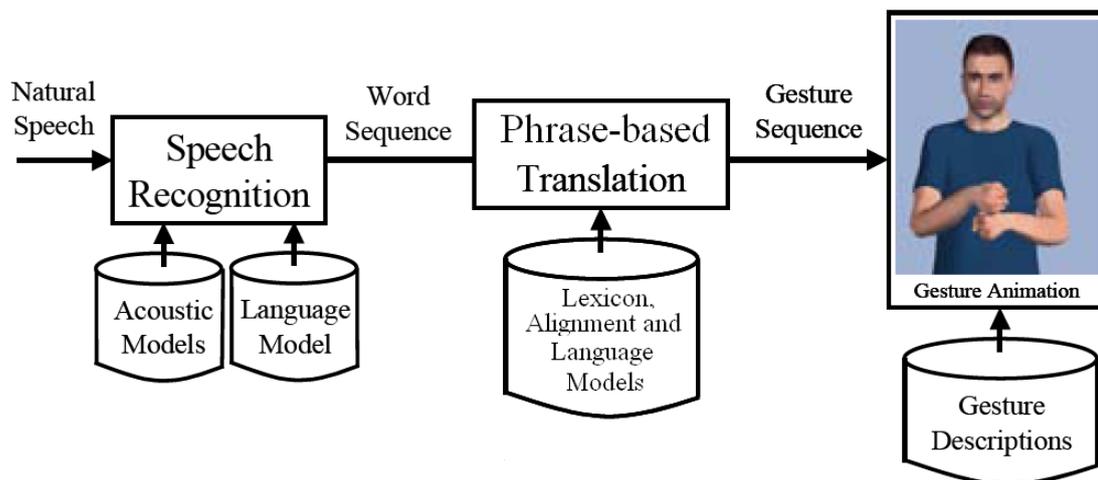


Figure 6.8. Spoken Language to Sign Language translation system

The recognizer uses context-dependent continuous Hidden Markov Models (HMMs) built using decision-tree state clustering with more than 1,800 states and 7 mixture components per state. These models were trained with more than 40 hours of speech and 4000 different speakers using the SpeechDat database in order to make it robust against the great range of final users of the service. Although SpeechDat is a telephone speech database, the acoustic models could be used in a microphone application because Cepstral Mean Normalization (CMN) and Cepstral Variance Normalization (CVN) techniques were used to compensate the channel differences. As front-end, our recognizer uses PLP coefficients derived from a Mel-scale filter bank (MF-PLP), with 13 coefficients including the energy coefficient, c_0 , and their first and second-order differentials, giving 39 parameters for each 10 ms frame. The speech recognizer uses a set of 45 allophone units of the Spanish language and 16 silence and noise models for detecting acoustic sounds (non-speech events like background noise, speaker artifacts, filled pauses, etc.) that appear in spontaneous speech.

On the other hand, the language models provide the knowledge about what constitutes a possible sentence, what words are likely to co-occur, in what sequence, the domain, the speaker style, and the lexical and grammatical complexity and variations of spoken language. In our system, we use a statistical n-gram based language model of order two (i.e., bigrams) due to the data sparseness, as there are only 266 sentences to train the model, which is too small considering the size of the vocabulary and the amount of different n-grams. Although the recognition system can generate a N-best list of recognized sentences sorted by similarity to the spoken utterance, in our work we only use the optimal word sequence. Finally, the recognizer also generates a confidence measure with values ranging from 0.0 (lowest confidence) to 1.0 (highest confidence) for each recognized word in the word sequence [Ferreiros et al, 2005].

The second module corresponds to a statistical phrase-based machine translation (SMT) system, which translates the recognized utterance into a sequence of semantic symbols, i.e., glosses, representing the grammar structure and sequence that follow the Spanish Sign Language. In order to generate the translation, the system uses three different models: the lexicon model, the alignment model, and the language model. The lexicon model provides the probability of translating one word into another and helps to disambiguate between words reducing problems due to homonymy or polysemy. The alignment model provides the probability of mapping words/group of words in the source sentence and words/group of words in the target sentence, considering in this case several factors such as phrase length, positions of the words, contexts, previous alignments, etc. Finally, the language model is used to provide knowledge about the well formedness of the translated sentence, evaluating also the syntactic and semantic structure of the candidate sentences. In sections 6.2.4 and 6.2.5 (page 174 and 177) we provide a more detailed description of the translation system and the language model used in our system.

Finally, the third module is the animated agent or avatar. The avatar is the responsible for providing the graphical output in the sign language for the deaf users. In order to do it, the avatar receives as input a gesture sequence, i.e., the translated sentence using glosses as words. Then, the system searches each gloss in a predefined dictionary that codifies the sequence of movements of the avatar to play the sign. In our system, the selected avatar was VGuido, created during the European project eSIGN (see section 2.4.4 and 3.5.2, page 50 and 75). In our case, we took advantage of the possibility of including this avatar in our runtime platform as an ActiveX. This toolkit also includes the eSIGN Editor environment that was used to define and store the signs using HamNoSys and SiGML notation. A more detailed description of this system can be found in [San-Segundo et al, 2008].

6.2.2 Bilingual Corpus

For our task, the translation system was focused on a limited domain, composed by sentences spoken by an officer when assisting deaf people in applying for, or renewing, the National Identity Document (DNI). In this context, a speech-to-sign language MT system is very useful because most of the officers do not know the Spanish Sign Language (LSE). The corpus consists of 416 sentences selected from spoken dialogues between officers and hearing users.

The main features of the corpus are summarized in Table 6.15. For both text-to-sign and speech-to-sign translation the same test set was used. In order to train the language model for the target language, it was necessary to create a dictionary of text symbols called glosses that represent each sign to be animated by the avatar. The dictionary was defined by a sign

language expert who analyzed each Spanish sentence and translated them into a sequence of glosses using the Spanish sign language grammar. For instance, a sentence like “tú tienes que pagar 20 euros de tasa” is translated into the following glosses: “FUTURO TÚ VEINTE EURO TASA PAGAR OBLIGATORIO”, or the sentence “el DNI debe ser renovado cada cinco años” is translated into “CADA CINCO PLURAL AÑO RENOVAR DNI TÚ OBLIGATORIO”. In these examples, each sign has been represented by a gloss written in capital letters. The final size of the glosses dictionary was around 320. Observe the ordering of the glosses and the semantic-like representation.

As it is also shown in Table 6.15, the size of the vocabulary in comparison to the overall amount of running words in the training set is very high (17%). In addition, the perplexity of the test set is high considering the small size of the vocabulary. Both values are unquestionable signs of data scarcity, which is likely to cause a high dispersion when estimating the parameters of the statistical translation models.

Training	Spanish	LSE
Sentences Pairs	266	
Number of Words	3153	2952
Vocabulary	532	290
Dev and Test	Spanish	LSE
Sentences Pairs	150	
Number of Words	1776	1688
OOV	90	30
Vocabulary	427	250
Perplexity (3-grams)	15.4	10.7

Table 6.15. Corpus statistics summary

In these circumstances, the high amount of unknown words in the test set (OOVs, Out-of-vocabulary) represents another important issue. In this task, there are 90 OOVs out of 532 (16.9%). Since the current statistical system does not use any morpho-syntactic parser to analyze the unknown words, it can hardly cope with them. So far, in the literature only naïve methods have been implemented to face the translation of OOVs. The usual adopted solution displays the unknown input word itself, without any change, in the output language. This solution is successful on the assumption that most of the unknown words are proper names, numbers, etc. That is, OOVs correspond to tokens that can be transcribed in the same way in any language. However, in this task, most of the OOVs correspond to non-proper names, which indeed do not match any symbol in the Sign Language. Therefore, the usual solution was not totally useful under this framework, and thus, new solutions had to be proposed in future developments. In [San-Segundo et al, 2007], we presented some solutions to this

problem, although they were not applied in the present thesis, considering only the usual solution of maintaining the same word in both languages.

Finally, the sentences were randomly divided into three sets, with 266 phrases for training. With the remaining sentences, we created three-fold cross validation sets leaving 50 sentences for development and 100 for test each time. For both text-to-sign and speech-to-sign translation experiments the same test and development sets were used.

6.2.3 Speech Recognition Results

As we have mentioned, one of our goals with this system is the possibility of automatically translating written or spoken prompt sentences defined in the platform. In the former case, 416 sentences were collected from spoken dialogues between officers and hearing users when applying or renewing the DNI document as we have described in the previous section. 15 speakers were recorded (eight men and seven women), each one uttering 50 sentences from the test and development sets, obtaining 750 utterances in total. This way, each test sentence was uttered by five different speakers. The objective of recording all these speakers was to obtain realistic results, as the speech recognizer must be speaker independent.

WER (%)	Ins (%)	Del (%)	Sub (%)
26.39	3.53	6.92	15.95

Table 6.16. Speech recognition results.

Table 6.16 shows the recognition results using the system described in section 6.2.1 and a bigram language model. The WER is high which is probably due to the low number of sentences available to train the model. The main reasons for these poor recognition results are also the big influence of the OOV rate (see Table 6.15) over the WER results (16.9% vs. 26.39%), the poorly trained LM, and the fact that some speakers uttered some sentences with a low volume. Without these problems, the same recognition system has a 4.2% WER in a similar task [Cordoba et al, 2005].

6.2.4 Statistical Machine Translation System

As we have described in the state-of-the-art, in automatic language translation, the goal is to translate a text, given in some source language, into a target language. Given a source string, in Spanish for this task, $f_1^J = f_1 \dots f_J$, it must be translated into a target string, in Spanish Sign Language, $e_1^I = e_1 \dots e_I$. Among all possible target strings, the system will choose the string with the highest probability that is given by Bayes decision rule:

$$\hat{e}_1^I = \arg \max_{e_1^I} \{ \Pr(e_1^I | f_1^J) \} = \arg \max_{e_1^I} \{ \Pr(f_1^J | e_1^I) \cdot \Pr(e_1^I) \}$$

Eq. 6.14

the program ‘*phrase-extract*’ included in the Pharaoh⁶⁶ toolkit, setting as heuristic to create the phrases the *grow-diag-final* algorithm. This algorithm creates an initial phrase based on using reliable alignment points after intersecting the word alignments created from the source-to-target language and the target-to-source language. Then, the algorithm adds less reliable neighbouring points based on using the union between both parallel word alignments. Finally, the algorithm prunes the inconsistent alignments based on the interactions of the EM algorithm and the maximum phrase size. Figure 6.9 shows an example of an alignment template used for the creation of the phrase table where the maximum phrase size was set to three to simplify the selection of the source n-grams that needed to be used to query the Web in our proposed technique (see section 6.2.5, steps 1 and 2).

- **Phrase scoring:** In this step, the phrase translation probabilities, $p(\bar{f}_i | \bar{e}_i)$ and $p(\bar{e}_i | \bar{f}_i)$, and lexical weights, i.e., the probability of the phrase given the word alignment, are computed for all phrase pairs using the program ‘*phrase-score*’ included in the Pharaoh toolkit. Although in our proposed adaptation technique we only use the phrase translation probabilities, the algorithm calculates both at the same time.
- **Minimum Error Training:** After training the phrase table, the next step is to optimize the values of the different parameters used during the decoding process. The weights are optimized on the BLEU score using the sentences that appear in the development set. Specifically, the algorithm optimizes the weight for the language model, the phrase translation model, the distortion model, and the word penalty. The two former correspond to the phrase translation model trained in the previous step and the language model trained using the SRILM toolkit [Stolcke, 2002] using a baseline trigram model or through the proposed technique in the next section. The two latter correspond to a factor, α in Eq. 6.15, that modifies the phrase translation probability as a function of the relative distance between the positions of the current and previous translation phrases (i.e., $start_i$ and end_{i-1}), and a weight w that penalizes too short sentences. Eq. 6.15 shows the formula used to select the best translation sentence.

$$\hat{e}_1^I = \arg \max_{e_1^I} p(e_1^I | f_1^J) = \arg \max_{e_1^I} \left[\prod_{i=1}^I p(\bar{f}_i | \bar{e}_i) \cdot \alpha^{|start_i - end_{i-1}|} \right] \cdot p_{LM}(e_1^I) \cdot w^{length(e_1^I)}$$

Eq. 6.15

Finally, during the evaluation, the optimized weights are used for the Pharaoh decoder in order to produce the N-best translations; in our case, we selected only the first candidate. Then, different programs provided by the toolkit allow the calculation of the different quality metrics explained in section 2.4.3 (page 47), in our case: WER, PER, BLEU, and NIST.

⁶⁶ <http://www.isi.edu/licensed-sw/pharaoh/>

6.2.5 Proposed Adaptation Technique

According to the Bayes decision rule, Eq. 6.14, the target language LM, $\Pr(e_1')$, is used for ensuring that the translated sentences are well formed and fluent. In order to obtain good results, it is necessary that the target language model is reliably trained using a large corpus to provide good estimations of the occurrences of the different n-grams that appear in the training data. However, in most applications it is difficult to have or to obtain such kind of corpus available. In addition, since our corpus contains only a few sentences, the target language model could not be estimated properly. Therefore, it was obvious that some kind of adaptation technique had to be applied to overcome this problem. Therefore, we had to find solutions for two problems: the first one was the creation of the background corpus to adapt with the in-domain data, and the second one, the selection of a successful technique for performing the adaptation.

In order to solve the first problem, we considered that an interesting solution for the small corpus available was to start creating a background corpus for the source language and then to apply the translation model, $\Pr(f_1' | e_1')$, to translate the new collected sentences into the target language obtaining this way new target data to adapt with.

In section 2.2.1.5 (page 34), we described several methodologies for gathering new training data. Among the proposed techniques, an interesting alternative method to generate the background corpus is to collect Web frequency counts using information retrieval (IR) techniques. [Keller and Lapata, 2003] and [Zhu and Rosenfeld, 2001] report different experiments that confirm that LMs estimated using Web frequency counts can be used for adaptation purposes providing comparable or better results than the ones obtained retrieving full sentences from online pages, and with the big advantages of reducing the system latency and avoiding the incorporation of undesirable sentences.

However, before continuing we have to solve one important issue: how to create the list of selected keywords to retrieve from the Web. In this case, we first decided to create a list of target n-grams for which we wanted to obtain more reliable counts. Obviously, we wanted to include all the n-grams from all orders, especially trigrams. However, if the number of n-grams is too high we have to impose some threshold in order to reduce the number of selected n-grams. In our case, after considering different possibilities, we set a threshold based on the phrase translation probability $p(\bar{f}_i | \bar{e}_i)$, i.e., the probability of translating one target-side n-gram (LSE) into a source-side n-gram (Spanish). Our goal is to consider a given target n-gram only if the retrieved source-side n-gram is highly correlated with it. In order to obtain the translation probability, we took advantage of the phrase translation table created during the training of the statistical machine translation system (see section 6.2.4). During this process, the phrase table is created using the Pharaoh toolkit, modifying the maximum phrase size from the default value of seven to three in order to simplify the selection of the n-grams to be used to query the Web.

After solving the first problem, we focused on finding the adaptation framework. In this case, we were especially interested in methodologies operating at the count level since the proposed methodology for creating the background corpus relies on retrieving Web frequency counts instead of full sentences. In [Bellegarda, 2004] several methods to overcome this problem are described. In most cases, the adaptation consists of building two LMs, one trained from the in-domain corpus and another one from a background corpus (out-of-domain, or less specific corpus which is expected to be bigger than the in-domain one), and then applying an adaptation formula that modifies the well estimated background model

using information from the in-domain model. Among the best adaptation techniques proposed in the literature we decided to use the Maximum A-Posteriori (MAP) [Bacchiani et al, 2006] method. In this technique, the adaptation is made at the frequency count level using Eq. 6.16.

$$p(w_q | h_q) = \frac{\alpha \cdot C^I(w_{i-N+1}^i) + \beta \cdot C^O(w_{i-N+1}^i)}{\alpha \cdot C^I(w_{i-N+2}^i) + \beta \cdot C^O(w_{i-N+2}^i)}$$

Eq. 6.16

Here, C^I and C^O are the frequency counts for the in-domain and out-of-domain corpora respectively, α and β are weight factors, estimated empirically to reduce the bias of the estimators and to apply a different weight to each component model.

The next step was to retrieve the frequency counts from the Web for the selected source-side n-grams and converting them back into target-side n-grams. In this case, we again used the phrase translation table but in the opposite direction, $p(\bar{e}_i | \bar{f}_i)$, i.e., the probability of translating a source-side n-gram into a target-side n-gram. After that, it was possible to apply the adaptation framework, MAP, using the original target-side n-grams counts and the ‘translated’ n-grams retrieved from Internet, in order to generate a new language model that could be used to adapt with the original one. Finally, with the new adapted LM we were able to evaluate the quality of the new translated sentences.

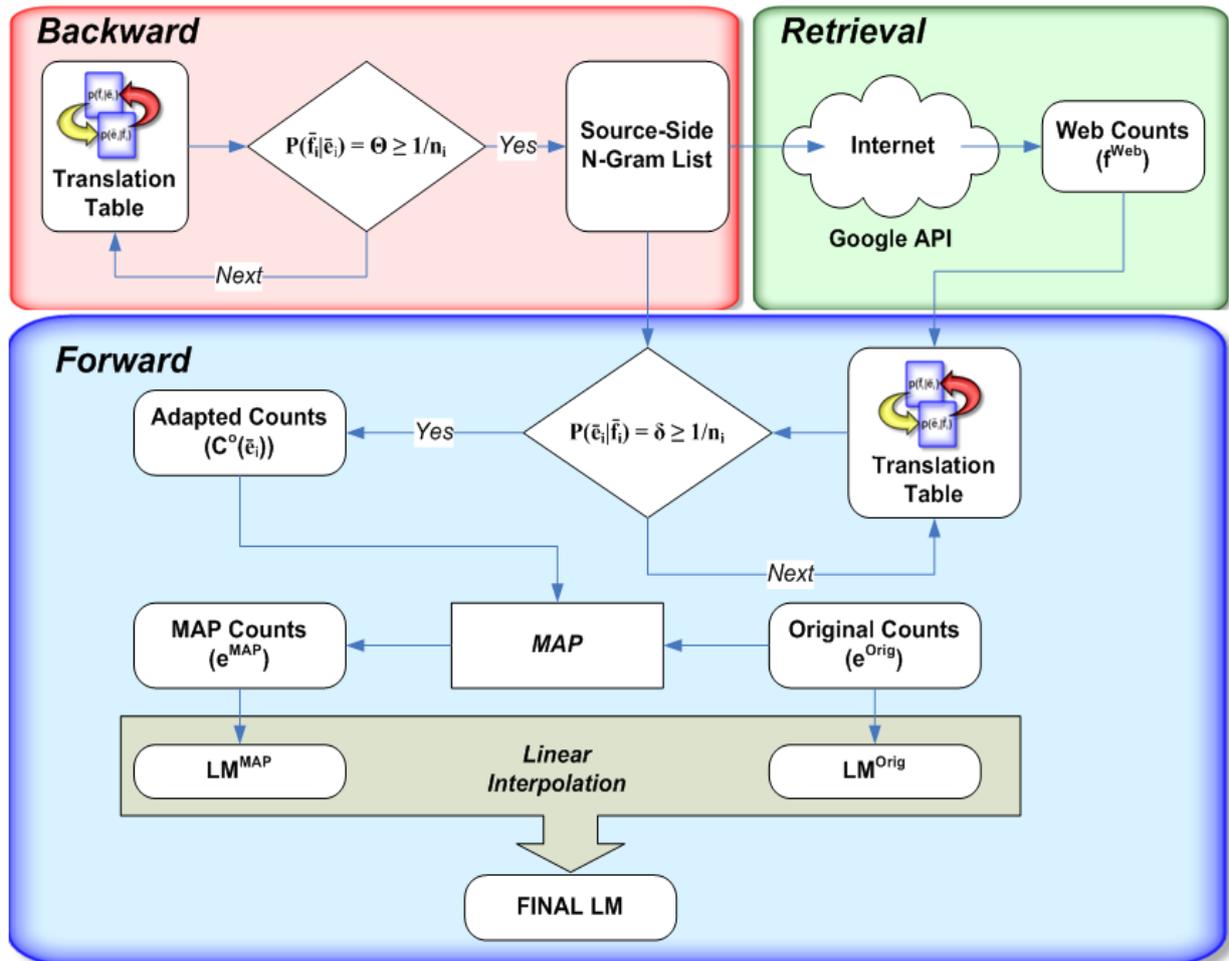


Figure 6.10. Flow diagram of the proposed adaptation technique

Figure 6.10 shows the process of the methodology proposed in this thesis. According to this figure, the adaptation is done in three steps:

1. **Backward:** To start with, the system uses the phrase pairs table created independently during the training of the translation probability $\Pr(f_1^J | e_1^1)$ (Eq. 6.14). The table consists of a list of n-gram pairs that are consistent translations between the source and target language, with their probabilities $p(\bar{f}_i | \bar{e}_i)$ and $p(\bar{e}_i | \bar{f}_i)$, and lexical weights [Koehn et al, 2003]. Using this table, the system creates the list of source-side n-grams, used in the next step, that satisfy $p(\bar{f}_i | \bar{e}_i) \geq \theta$. Here, the threshold θ is used to reduce the number of n-gram pairs to query the Web and to guarantee that the source-side n-grams are reliable translations of the target-side n-grams that we wanted to improve. After experimenting with different values and options, θ was finally set to $1/n_i$, where n_i is the number of reverse translations for \bar{f}_i . However, it could be fixed as a function of the corpus size and the translation model quality. The final list consisted of 1270 source-side n-grams (410 unigrams, 497 bigrams, and 362 trigrams).
2. **Information Retrieval (IR):** Using the n-gram list, the system queries the internet to obtain Web frequency counts using the Google-API⁶⁷. In this case, we had to deal with some limitations of the API; mainly, we were limited to perform only 100 queries at day per registered keyword, and considering that the total number of n-grams in the list was 1270, we decided to ask Google for a less restrictive license. In this case, the new keyword allowed us to perform 10,000 queries at day. This way, we were able to perform the retrieving process in just one day. Then, a new source LM was created interpolating the original LM (in-domain) and the MAP-adapted source LM created applying Eq. 6.16 between the retrieved source-side n-gram counts and the original counts.
3. **Forward:** Finally, the translation table is applied again, but on the opposite direction, to obtain the n-gram frequency counts on the target side. The conversion is done taking each n-gram pair in the list, \bar{f}_i , multiplying the retrieved Web count, $N^{web}(\bar{f}_i)$, by the phrase translation probability, $p(\bar{e}_i | \bar{f}_i)$, and summing up all the contributions that satisfy $p(\bar{e}_i | \bar{f}_i) \geq \delta$ to obtain the counts for the target n-gram, $C^O(\bar{e}_i)$ (see Eq. 6.17). Then, Eq. 6.16 is applied to merge the counts from the original target-side corpus with the ‘translated’ counts. Finally, as in step 2, a new LM is created from the linear interpolation of the original target-side LM and the MAP-adapted target LM.

$$C^O(\bar{e}_i) = \frac{\sum_{\forall \bar{f}_i: p(\bar{e}_i | \bar{f}_i) \geq \delta} N^{web}(\bar{f}_i) * p(\bar{e}_i | \bar{f}_i)}{\sum_{\forall \bar{f}_i: p(\bar{e}_i | \bar{f}_i) \geq \delta} p(\bar{e}_i | \bar{f}_i)}$$

Eq. 6.17

⁶⁷ <http://code.google.com/apis/ajaxsearch/>

Table 6.17 shows an example, in English, of the phrase table used to create the n-gram list, as well as the process followed to create the new adapted counts. In this case, for the target-side trigram: “YOU MUST DELIVER” there are three suitable translations (n_i) on the source-side. Given the condition, $p(\bar{f}_i|\bar{e}_i) \geq \theta = 1/n_i = 0.333$, the system only selects the n-grams pairs one and three during the backward step. For the bigram: “YOU MUST”, $n_i=4$ then given the condition $p(\bar{f}_i|\bar{e}_i) \geq \theta = 1/4 = 0.25$, the system would select n-grams pairs b and d to create the list of n-grams to query the Web.

The next step is to retrieve the counts from the Web that are presented in column 5 in the table. Observe that the system only retrieves the counts of the n-grams that fulfil the condition $p(\bar{f}_i|\bar{e}_i) \geq \theta$. With these counts, we also create a new MAP-adapted source-side LM that interpolated with the original source-side LM allowed us to obtain the perplexity results shown in Table 6.18. For the source side, the weight factors from Eq. 6.16 were optimized on the dev sets (cross-fold) running a downhill simplex algorithm, resulting in the following average values for the source side: $\beta_s = 0.000417$ and $\alpha_s = 36.7$. The interpolation weight was set to $\lambda_s = 0.51$.

Source (\bar{f}_i)	Target (e_i)	$p(\bar{e}_i \bar{f}_i)$	$p(\bar{f}_i \bar{e}_i)$	Web Counts	Original Counts	Final Counts
1.) you must deliver	TRIGRAM: <i>YOU MUST DELIVER</i>	0.5	1.0	135000	12	587
2.) you must bring		0.1	0.2	$p(\bar{f}_i \bar{e}_i) < \theta$		
3.) you have to provide		0.4	0.5	80420		
a.) you should	BIGRAM: <i>YOU MUST</i>	0.18	0.071	$p(\bar{f}_i \bar{e}_i) < \theta$	76	3664
b.) you have to		0.364	0.739	148000		
c.) you need		0.046	0.143	$p(\bar{f}_i \bar{e}_i) < \theta$		
d.) you must		0.410	0.952	179000		

Table 6.17. Example of n-grams in the phrase translation table

During the forward step, we use Eq. 6.17 in order to convert the retrieved counts into ‘translated’ counts. In this case, in the example the original count for the trigram gloss is 12, for the bigram gloss is 76, and setting the MAP weights (α and β) to the optimum values obtained using the development data (in this case $\alpha_t = 48$ and $\beta_t = 0.0001$), the out-of-domain target-side trigram count is

$$C_{\text{MAP}}^{\text{O}}(\text{YOU MUST DELIVER}) = 48 * 12 + 0.0001 * \left[\frac{135000 * 0.5 + 80420 * 0.4}{0.5 + 0.4} \right] = 587,$$

And the bigram adapted count is

$$C_{\text{MAP}}^{\text{O}}(\text{YOU MUST}) = 48 * 76 + 0.0001 * \left[\frac{148000 * 0.364 + 179000 * 0.410}{0.364 + 0.410} \right] = 3664.$$

Using these adapted values, we can train a new target LM using the SRILM toolkit⁶⁸ in order to generate the MAP-adapted target-side LM.

Finally, the MAP-adapted target-side LM is interpolated with the original target-side LM using an interpolation value calculated optimized on the dev sets (cross-fold) running the downhill simplex algorithm. In this case, the interpolation weight was set to $\lambda_t = 0.52$.

6.2.6 Language Model Experiments

Table 6.18 shows the perplexity results provided by the baseline LMs and the adapted ones for the train, development, and test sets. The results for the test and development sets correspond to the averaged perplexities for the three-fold cross validation. The baseline LMs are backoff trigram with Good-Turing discount. The perplexities on both sides correspond to the adapted LMs. Values in parenthesis are relative improvements over the baseline perplexities.

	Train		Dev		Test	
	Source	Target	Source	Target	Source	Target
Baseline	5.65	5.02	15.34	10.8	15.37	10.7
Adapted	3.01 (46.7%)	3.16 (37.1%)	11.92 (22.4%)	8.75 (18.7%)	12.45 (18.9%)	9.04 (15.5%)

Table 6.18. Perplexity (PPL) results using the corresponding LM

According to these results, the proposed adaptation reduces perplexities in both sides, with improvements over 15%, so a nice reduction in WER can be expected, according to the rule of thumb for reductions in perplexity (see section 2.2.1, page 25)

In spite of the good improvements in both sides, we can also observe that the improvements are higher on the source side than on the target side. In this case, the reduction in the target side is due to process of ‘translating’ the counts, i.e., the forward step, since the translation table introduces some mismatch that reduces the improvement on the target side from 18.9% to 15.5% in the test set.

6.2.7 Machine Translation Experiments

As we have explained before, one interesting characteristic of the Bayes decision rule used for the translation system, Eq. 6.14, is that the language and the translation models provide independent information, so they can be trained individually. Consequently, in the experiments presented in this section the proposed language models were not used at all

⁶⁸ <http://www.speech.sri.com/projects/srilm/>

during the process of creating the translation model, therefore the effect of the language models can only be measured in the evaluation step.

It is also important to mention that for training the phrase-based translation model we considered only the sentences from the training set and optimized it on the development set, using clean sentences and not the recognized texts. This fact is important since the translation model is created assuming that the input text is syntactically and grammatically correct. The reasons for using the clean text was that we did not have recordings for the train sentences but only for the test and development data, and because we wanted to separate the effects of the speech recognition errors from the translation model. Finally, in order to create the translation model we set the maximum phrase size, in the Pharaoh toolkit, to the default value of seven in order to look for better alignments.

Table 6.19 shows the averaged MT results for the text-to-sign and speech-to-sign experiments on the test set for the three different conditions that we have considered:

- **Exp 1:** In this case, the system uses the original language model generated considering only the sentences from the training set. This is the baseline system.
- **Exp 2:** For this experiment, the language model is generated with the proposed technique considering only the training set.
- **Exp 3:** Finally, in this experiment the language model is trained considering all available sentences, i.e., using train, development, and test sets, without including any adaptation. Since this model has all the available information, it corresponds to the top performance that it is possible to obtain only due to the LM component and without the effect of OOVs.

		WER	PER	BLEU	NIST
Text-to-Sign	Exp 1	34.74	29.59	0.50	6.30
	Exp 2	33.79 (2.73%)	29.1 (1.68%)	0.51 (2.61%)	6.36 (1.06%)
	Exp 3	32.62 (6.1%)	28.06 (5.48%)	0.55 (9.91%)	6.57 (4.23%)
Speech-to-Sign	Exp 1	42.87	38.94	0.43	5.65
	Exp 2	42.53 (0.78%)	38.57 (0.95%)	0.44 (3.75%)	5.70 (0.89%)
	Exp 3	41.43 (3.36%)	37.8 (2.9%)	0.47 (9.96%)	5.86 (3.62%)

Table 6.19. Average Machine translation results for the test set (Exp 1-3)

In order to assess the quality of the obtained translations, the four usual evaluation measures in machine translation have been taken into account (see section 2.4.3, page 47):

- WER (Word Error Rate),
- PER (Position Independent WER),
- BLEU (BiLingual Evaluation Understudy),
- NIST

The former two are error measures (the higher the value, the worse the quality) whereas the latter two are accuracy measures (the higher, the better). We have used BLEU and NIST scores since they present high correlation with human translation. In our case, since the target corpus in the sign language was manually created by experts, we considered important to try to obtain similar translations to the ones created by the experts (see section 6.2.2) when translating new sentences.

According to the table, for the text-to-sign MT system, the results show that the proposed technique is able to reach approximately half (2.73%) of the maximum improvement (6.1%) in WER that it is possible to obtain due only to the LM component. Considering the high ratio of OOVs (10,4%) and the small size of the training data, the result is outstanding.

From these experiments, it is possible to guess that the quality of the translation model limits significantly the improvement reached by better LMs. This intuition was confirmed when we tested an optimal MT system, i.e. trained using all the available sentences. In this case, the WER for Exp3 was 13.06% instead of the 32.62% presented in the table, the WER for Exp2 was 15.3%, and the baseline, Exp1, was 16.0%. In this case, using a better phrase translation model, the proposed adaptation also produces a relative improvement of 4.4 % over the baseline. In this case, the improvement is low, but this is not surprising at all since we are using a better translation model. Therefore, during the optimization of the different weights for the decoder, the language model receives a smaller weight compared to the translation model.

The second part of the table shows the results for the speech-to-sign language translation. Here, we observe that, unfortunately, the improvements are lower. The most probable explanation is that the speech recognition introduces errors that are not modelled by the translation model since we trained it using clean text. In addition, the decoder does not take advantage of the better estimated and high order n-grams ‘translated’ from the Web counts because most of the n-grams in the translated sentence do not correspond to the ones re-estimated with our technique.

6.2.8 Conclusions

In this section, we have described the incorporation of an automatic machine translation system that can be used to convert the previously defined written or spoken prompts of a dialogue application into animated prompts in the sign language. This way, the design platform is extended to support new modalities and a new kind of final users, in this case deaf people.

Then, we have presented a successful technique to adapt the target-side language model used for a machine translation system especially in situations where there are very scarce resources to obtain reliable models. The technique uses information from the source language, Spanish in our task, and from the independently trained phrase-based translation matrix in order to create a new LM, estimated using Web frequencies, which adapts the counts of the target-side language model through the Maximum A Posteriori method (MAP).

For the evaluated task, the proposed technique provided a relative improvement of 18.9% and 15.5% in perplexity over the base system for the source and target language respectively. In this case, the difference between both improvements were mainly due to the mismatch introduced by the translation table used to convert the frequencies retrieved from the Web into frequencies on the target side.

In relation with the machine translation experiments, the results for the text-to-sign experiment showed that the proposed adaptation provides a 2.73% relative reduction on WER that is near to half the performance that it is possible to achieve when only the LM is optimized. However, the results for the speech-to-sign experiments did not produce considerable improvements, which was probably due to the effect of recognition errors in the Web counts for the n-grams with errors and to the fact that the translation model was trained using only clean texts instead of using recognized sentences, so it does not model the effect of the recognition errors.

7 CONCLUSIONS AND FUTURE WORK

In this chapter, we present a summary of the main conclusions (more details can be found at the end of each chapter), future work, and contributions of this thesis. Since in this thesis we have tackled a wide number of topics and systems, we have organized this chapter according to the three main components discussed in this dissertation: the dialogue design platform, the language identification system, and the machine translation system.

A first contribution of this thesis is the analysis of the state-of-the-art regarding platforms for the design of dialogue applications. In the thesis, we have described most commercial and research platforms to find out their characteristics, positive aspects, and limitations, in order to contribute with new ideas to the field and to be able to offer a complete, innovative, and up to date alternative development platform.

The main conclusions for all systems presented in the thesis are detailed below.

7.1 CONCLUSIONS

7.1.1 Dialogue Platform

In this thesis, we have described all the accelerations included in a multimodal and multilingual design platform in order to speed up the design and guide the designer through all the steps required to create dialogue services. The proposed accelerations are, in most cases, innovative without a direct correspondence to the ones offered by the current commercial and research platforms.

Different types of accelerations have been proposed according to the requirements, capabilities, and available information at each assistant that makes up the platform. The proposed accelerations take advantage of heuristic information extracted from the contents of the backend database and from an object-oriented representation of the data model structure, in order to generate different kinds of proposals that simplify the process of creating and completing the dialogue flow. Other accelerations consist of different wizard windows or simplified processes that help designers to complete, create, or debug models (e.g., grammars, prompts, SQL commands) required by the design and runtime platform in order to provide the service.

In order to study the usability and acceptability of the different assistants of the platform, as well as the proposed accelerations we carried out a subjective and objective evaluations with participants from different countries, mother tongues, and levels of experience in programming dialogue applications. The results showed that the proposed accelerations reduced the design time by more than 56%, and obtained a subjective score that ranges from 8.0 to 9.0. In addition, the whole platform was rated with an average score of 8.0 that also confirmed the high performance of the platform and its assistants.

The first process required to provide the accelerations was the automatic extraction of heuristic information from the database. In order to correct some limitations of the connecting database driver, mistakes in the definition of the fields in the database, and to allow a correct

mapping between the field types supported by the platform and the ones supported by the database we defined a set of regular expressions that were able to correct an 89.6% of the errors.

In relation with the accelerations included to create the data model structure we have proposed a new assistant that simplifies the definition of the object-oriented classes taking advantage of the heuristic information and previous data model libraries.

In relation with the assistant that defines the prototypes of the database access functions, we proposed an automatic process for generating and debugging SQL statements used by the real-time system based on the analysis of the input/output and the type of the parameters defined in the function prototypes. Another acceleration, proposed and implemented by the partners of the GEMINI project, allowed the definition of relations between the function arguments and the data model structure, which is used in the following assistants through different kind of automatic proposals that simplify the design.

Regarding the state flow model assistant, the main accelerations included in this thesis were the automatic generation of different state proposals that can be used to quickly create complex states, together with the possibility of using an automatic analysis of the feasibility of the slots defined in a given state of being requested using mixed initiative or direct dialogues.

By far, the retrieval modelling assistant is the assistant with the highest number of accelerations. Here, we have proposed several automatic dialogues and templates that can be used to obtain or present information to the final user, the incorporation of an innovative auxiliary window where the designer can find all the actions that are considered relevant for the dialogue being edited, and an automatic procedure to help the designer to connect the input/output parameters of different actions and dialogues with the local/global variables that contain or will contain the information for/from those actions and dialogues. Finally, we have also designed a simple procedure, not present in most design platforms, to define dialogues with mixed-initiative and over-answering capabilities. The subjective evaluation showed that the accelerations included in this assistant were scored in average with an 8.9, and the assistant with an 8.6. The objective metrics also showed that the proposed accelerations contributed to reduce the design time by an 89.4%.

Considering the assistant that defines the specific details for the speech modality, the proposed accelerations were the automatic generation of the dialogue flow required for the confirmation handling of the user answers, together with an assistant where the dialogue flow for providing the information contained in a list of retrieved results after querying the backend database can be specified. This assistant, given its simplicity and the high level of acceleration offered was rated during the subjective evaluation with a 9.0 score.

Finally, other assistants in the platform were also accelerated in order to allow the quick definition of language dependent prompts and grammars used by the speech recognizer. In this case, we proposed an automatic procedure for creating stochastic grammars from a finite state grammar in JSGF format. Other accelerations proposed by the partners of the GEMINI project included the possibility of automatically creating pronunciation dictionaries and an assistant for creating prompts in different languages using the prompts for the default language as template.

7.1.2 LID System

In this thesis, we have proposed a new language model for applying phonotactic constraints to a PPRLM-based LID system. The proposed technique is based on using a frequency ranking of discriminative n-grams that outperforms the traditional approach based on using a deleted interpolation between n-grams of different orders. In our case, we obtained an accumulative relative improvement of 13.0% (from 3.69% to 3.21%).

Our first contribution was the introduction of several new ideas and important changes to the original n-gram frequency ranking proposed in the literature for LID on written text. For instance, we have arrived to the conclusion that the ranking size should be increased as much as possible when that number of different n-grams is available. In our case, it was set to 3000. Besides, we have demonstrated that instead of using a common ranking for all n-grams it should be better to use n-gram specific rankings as it provides better results. Finally, the selection of the most discriminative n-grams was an important factor to obtain better LID results; in this case, we proposed new formulations based on the widely used *tf-idf* metric in order to normalize the results.

Besides, we have also demonstrated that the fusion with the traditional PPRLM approach and the incorporation of different acoustic and duration based information in addition to the proposed n-gram frequency ranking resulted in additional improvements in the LID rates. In this case, we obtained an accumulative relative improvement of 31.7% (from 3.69% to 2.52%)

On the other hand, we have also demonstrated that the measure of separation between pdf distributions of the Gaussian classifier is a good tool to reduce the number of experiments and to anticipate which features are going to be actually discriminative for the LID task.

Finally, it is important to highlight that one of the critical aspects that had contributed the most to obtain these large improvements was the incorporation of the Gaussian classifier. In this case, the Gaussian classifier allowed us the fusion of different sources of information, as well as a reduction of the bias/normalization problem present in traditional classifiers. Unfortunately, the size of our current database did not allow us to exploit all the possibilities of our multi-Gaussian classifier since the results with a variable number of Gaussian mixtures did not produce considerable improvements in the LID rate.

7.1.3 Machine Translation System

Finally, in this thesis we have also proposed a successful adaptation technique that takes advantage of the possibility of obtaining better estimations from the source-side language of the translation system and from the phrase-based translation model in order to create a new LM for the target-side that can guarantee better translations. In our proposal, the new LM was estimated using Web frequencies that adapted the counts of the target-side language model through the Maximum A Posteriori method (MAP). The proposed technique was evaluated on a restricted domain where deaf people could obtain information for applying or renewing the National Identity Document. Our proposed technique provided improvements in perplexity and translation for speech-to-sign language and text-to-sign language. Unfortunately, the results for the speech-to-sign language translation did not produce considerable improvements probably due to the effect of recognition errors that did not take advantage of the high order n-gram counts retrieved and adapted from the Web.

7.2 FUTURE WORK

In this section, we will describe the main course of action proposed for each system described in the thesis, following the same order as for the conclusions.

7.2.1 Dialogue Platform

Considering the good results that we obtained during the subjective and objective evaluations, the growing demand of better and more complex dialogue systems, and to broaden the functionality of this kind of tools, several interesting ideas can be considered in order to improve the platform. This section describes all these ideas that we will classify by the different assistants considered in the thesis. In general, the proposals shown in this section do not necessarily correspond to improvements on the graphical interface although according to the comments of the participants of the subjective and objective evaluations it would be nice to have them.

7.2.1.1 Data model assistant (DMA)

To allow the automatic creation of complex data model structures created for each table in the database and to allow the possibility of including complex attributes using the relationships defined in the database between different fields and tables. Here, the assistant could also use the heuristics in order to select by default the most probable tables and fields to be used as attributes in the new classes.

To include a new XML tag that specifies when a given attribute in a class will be used in the following assistants to provide or to obtain information to/from the user. The idea is to accelerate the definition of the input/output parameters when defining the database access functions, to reduce the number of proposed dialogues, states, and slots in the SFMA and RMA assistants. In addition, the assistant could automatically propose the content of this tag based on the contents of the database table and field used to generate the class and the attribute.

7.2.1.2 Data connector model assistant (DCMA)

To improve the process of defining the input/output parameters of the function prototypes through a graphical interface and a toolbar with objects instead of the text-based interface currently implemented.

To extend the capabilities of generating and integrating the SQL statements and the script used to connect the database with the VoiceXML server at runtime. This can be critical if the complexity of the generated SQL statements is increased.

7.2.1.3 State flow model assistant (SFMA)

To extend the possibilities of the current toolbar in the main window in order to provide more generic templates that allow the creation of different kinds of states (e.g., template for single slot state, mixed slot states, complex states, etc.) and to provide a palette of different actions that can provide a simpler mechanism for connecting two or more states through the graphical interface.

To study the possibility of merging this assistant and the RMA given that both share many functionalities. However, in order to take advantage of the predefined separation of both assistants (i.e., the RMA uses the information provided by the SFMA to generate action proposals for each dialogue and to automatically create DGet/DSay dialogues) we propose the creation of a unified two layered assistant that the designer can switch to according to the design process. This way, using the first layer the designer specifies the same information as in the SFMA, and in the second layer the same information as in the RMA. Although the process seems easy, we will have to deal with the possibility of switching between layers without requiring a complete definition of the whole dialogue flow or state information.

7.2.1.4 Retrieval model assistant (RMA)

To reduce the number of automatic generated dialogues in order to simplify the main interface. The results of the subjective and objective evaluation showed that most of the proposed dialogues in this window were not used at all. In this case, we propose the incorporation of heuristic information to remove the less likely used dialogues to obtain or provide information. For instance, a field with too many words is a clear candidate to provide information instead of requesting it. Besides, the proposed tag in the DMA assistant should also contribute to this process.

7.2.1.5 Modality extension retrieval assistant for speech (MERA-Speech)

In this assistant, the main proposal is to allow advanced designers to get access to the automatic flow proposed for the DGet and DSay dialogues making possible the modification of the default behaviour for the proposed dialogues.

In addition, it is expected to include new confirmation profiles for the DGet dialogues depending on the number and type of the slots to be requested to the user. In this case, the incorporation of heuristic information should also contribute to extend the current profiles.

7.2.1.6 User modelling assistant (UMA)

For this assistant we plan the incorporation of an innovative methodology for proposing the default values for the confidence levels used by each DGet dialogue. In this case, we propose to use the heuristics of the database and a set of rules that can be used to modify the values specified by the designer in the first stages of the design (i.e., in the ADA assistant).

7.2.1.7 Common improvements or extensions to other assistants

For all the assistants, the number of libraries available could be increased. Most of the commercial platforms include a set of common libraries such as yes/no, phone numbers, SSN, credit card numbers, time of day, etc., as well as more complex libraries for proper names, alpha-numeric spelling, addresses, etc.

New strategies to reduce design time in the generation of grammars and prompts can be implemented. For instance, allowing the semi-automatic translation of prompts for the default language to the other languages supported by the platform. Besides, the incorporation of heuristic information from the database could be used to automatically accelerate the generation of the pronunciation vocabularies and to support dynamic grammars. Finally, we also suggest the possibility of creating/importing/exporting the speech grammars in JSGF format into/from other standard formats in order to allow the platform to reuse grammars created with other development platforms.

The current two modalities could be merged so that they can work at the same time using the X + V standard.

The incorporation of a new assistant to debug the service in text mode, i.e. without a complete underlying speech recognizer or synthesizer. This assistant will be especially important for the RMA and MERA-Speech assistants.

7.2.2 LID System

In this thesis, we have proposed the incorporation of a long-span language model technique that was combined with different acoustic and duration information into a multi-Gaussian classifier. However, we found problems with the acoustic and duration information mainly due to normalization problems. We could research a new normalization approach to include the duration of each phoneme into the feature vector. In our current approach, we found that this feature did not produce a good separation of the Gaussian distributions therefore producing bad results for the LID, although intuitively it should provide additional and complementary information to discriminate among languages.

Another work we want to propose is to explore new techniques to reduce the size of the feature vector used as input for the Gaussian classifier. Currently we have applied an automatic clustering of the allophones for each language, but we should also try other approaches as selecting the most discriminative ones. In [Lucas-Cuesta et al, 2008], we have started experiments using Linear Discriminative Analysis (LDA) obtaining promising results.

Finally, we could merge our current system with a classical GMM-based system that uses Shifted Delta Cepstral (SDC) coefficients to better model temporal information. The combination of these systems provides also good results according to the literature.

7.2.3 Machine Translation System

In this thesis, we have proposed the incorporation of a new adaptation technique for the language model used in the decoder for scoring the translation candidates. This approach was based on the Bayes decision rule where the main components are the translation model and the language model. However, most of the current decoders combine the translation and language models via a log-linear model that allows the incorporation other arbitrary features as well. As our current decoder also supports the log-linear combination, we propose the incorporation of new models based on automatic word-classes or POS-based models. Besides, instead of using the traditional linear interpolation for combining the background and the adapted language models we propose to explore other techniques such as the geometric interpolation or unigram rescaling.

We also propose to carry out new experiments to test the proposed technique on a larger database in order to check the effect of the available number of sentences for training over the performance of the method and the number of queries to retrieve using Google. Besides, experiments with other languages can show the effect of the Google index on different languages.

We could update our current translation system from the Pharaoh toolkit to the more recent Moses toolkit⁶⁹. The advantage of this new toolkit is that it allows the incorporation of morphological, syntactic, or semantic information through a more complex feature vector representing different levels of annotation. The toolkit allows many possibilities such as

⁶⁹ <http://www.statmt.org/moses/>

including only words, in this case producing the same phrase-based models we have used in this thesis, including words plus POS tags, including morphemes, including lemmas, etc. In addition, the toolkit implements new tools for better decodings, support for new language models including also the possibility of combining them in different ways, support for confusion network decoding, etc., which we believe can provide new ways to improve our current approach and obtain a better translation model.

Finally, we propose to extend the current database of signs in order to allow the creation of new services without requiring too much effort for the designer.

BIBLIOGRAPHY

PUBLICATIONS GENERATED BY THE THESIS

- [Cordoba et al, 2007a] Cordoba, R., D'Haro, L. F., Fernandez-Martinez, F., Macias-Guarasa, J., and Ferreiros, J. 2007. *Language Identification based on n-gram Frequency Ranking*. Interspeech 2007, pp. 354-357.
- [Cordoba et al, 2007b] Cordoba, R., D'Haro, L. F., Fernandez-Martinez, F., Montero, J. M., and Barra, R. 2007. *Language Identification using several sources of information with a multiple-Gaussian classifier*. Interspeech 2007, pp. 2137-2140.
- [Cordoba et al, 2006a] Cordoba, R., Ferreiros, J., San-Segundo, R., Macías-Guarasa, J., Montero, J. M., Fernández, F., D'Haro, L. F., and Pardo, J. M. 2006. *Cross-Task and Speaker Adaptation in a Speech Recognition System for Air Traffic Control*. IEEE Aerospace and Electronic Systems Magazine, Vol. 21, No 9, pp. 12-17.
- [Cordoba et al, 2006b] Cordoba, R., D'Haro, L. F., San-Segundo, R., Macías Guarasa, J., Fernández, F., and Plaza, J. C. 2006. *A Multiple-Gaussian Classifier for Language Identification Using Acoustic Information and PPRLM scores*. Actas IV Jornadas en Tecnología del Habla, pp. 45-48.
- [Cordoba et al, 2006c] Cordoba, R., San-Segundo, R., Macías-Guarasa, J., Montero, J. M., Barra, R., D'Haro, L. F., Plaza, J. C., and Ferreiros, J. 2006. *Integration of acoustic information and PPRLM scores in a multiple-Gaussian classifier for Language Identification*. IEEE Odyssey 2006: The Speaker and Language Recognition Workshop, pp.1-8.
- [Cordoba et al, 2004a] Cordoba, R., Fernández, F., Sama, V., D'Haro, L. F., San Segundo, R., Montero, J. M., Macías, J., Ferreiros, J., and Pardo, J. M. 2004. *Realización de sistemas de diálogo en una plataforma compatible con VoiceXML: Proyecto GEMINI*. Procesamiento del lenguaje natural N° 33, pp. 103-110. ISSN:1135-5948.
- [Cordoba et al, 2004b] Cordoba, R., Fernández, F., Sama, V., D'Haro, L. F., San-Segundo, R., and Montero, J. M. 2004. *Implementation of Dialogue Applications in an Open-Source VoiceXML Platform*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. I-257-260.
- [D'Haro et al, 2008] D'Haro, L. F., San-Segundo, R., Cordoba R., Bungeroth, J., Stein, D., and Ney, H. 2008. *Language Model Adaptation for a Speech to Sign Language Translation System Using Web Frequencies and a MAP framework*. Interspeech 2008, pp. 2119-2202.
- [D'Haro et al, 2006] D'Haro, L. F., Cordoba, R., Ferreiros, J., Hamerich, S.W., Schless, V., Kladis, B., Schubert, V., Kocsis, O., Igel, S., and Pardo, J. M. 2006. *An advanced platform to speed up the design of multilingual dialogue applications for multiple modalities*. Speech Communication Vol. 48, Issue 8, pp.863-887.

- [D'Haro et al, 2004a] D'Haro, L. F., Cordoba, R. de, San-Segundo, R., Montero, J. M., Macías-Guarasa, J., and Pardo, J. M. 2004. *Strategies to reduce Design Time in Multimodal/Multilingual Dialogue Applications*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. IV-3057-3060.
- [D'Haro et al, 2004b] D'Haro, L. F., Cordoba, R., Ibarz, I., San-Segundo, R., Montero, J. M., Macías-Guarasa, J., Ferreiros, J., and Pardo, J. M. 2004. *Plataforma de Generación Semiautomática de Sistemas de Diálogo Multimodales y Multilingües: Proyecto GEMINI*. Revista de Procesamiento del Lenguaje Natural No 33, pp. 119-126. ISSN: 1135-5948.
- [Hamerich et al, 2004a] Hamerich, S. W., Cordoba, R. de, Schless, V., D'Haro, L. F., Kladis, B., Schubert, V., Kocsis, O., Igel, S., and Pardo, J. M. 2004. *The Gemini Platform: Semi-Automatic Generation of Dialogue Applications*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. IV-2629-2632.
- [Hamerich et al, 2004b] Hamerich, S. W., Schubert, V., Schless, V., Cordoba, R., Pardo, J. M., D'Haro, L. F., Kladis, B., Kocsis, O. and Igel, S. 2004. *Semi-Automatic Generation of Dialogue Applications in the Gemini Project*. 5th SIGdial Workshop on Discourse and Dialogue, pp 31-34.
- [Lucas-Cuesta et al, 2008] Lucas, J.M., Cordoba, R., and D'Haro, L. F. 2008. *Applying feature reduction analysis to a PPRLM-multiple Gaussian language identification system*. V Jornadas de Tecnología del Habla, pp. 29-32.
- [San-Segundo et al, 2008] San-Segundo, R., Barra, R., Cordoba, R., D'Haro, L. F., Fernández Martínez, F., Ferreiros, J., Lucas, J.M., Macías-Guarasa, J., Montero, J. M., and Pardo, J. M. 2008. *Speech to sign language translation system for Spanish*. Speech Communication Vol. 50, pp.1009–1020, ISSN: 0167-6393.
- [San-Segundo et al, 2007] San-Segundo, R., Pérez, A., Ortiz, D., D'Haro, L. F., Torres, M. I., Casacuberta, F. 2007. *Evaluation of Alternatives on Speech to Sign Language Translation*. Interspeech 2007, pp 2529-2532.
- [San-Segundo et al, 2006] San-Segundo, R., Barra, R., D'Haro, L. F., Montero, J. M., Cordoba, R., and Ferreiros, J. 2006. *A Spanish speech to Sign Language translation system for assisting deaf-mute people*. Interspeech 2006, pp. 1399-1402.
- [Vilar et al, 2006] Vilar, D., Xu, J., D'Haro, L. F., and Ney, H. 2006. *Error analysis of statistical machine translation output*. Intern. Conf. on Language Resources and Evaluation (LREC), pp. 697–702.

GENERAL BIBLIOGRAPHY REFERRED IN THE THESIS

- [Abdel-Fattah, 2005] Abdel-Fattah, M. A. 2005. *Arabic Sign Language: A perspective*. Journal of Deaf Studies and Deaf Education 10: 2, pp. 212-221.
- [Allen et al, 2001] Allen, J., Byron, D., Dzikovska, M, Ferguson, G., and Galescu, L. 2001. *Towards Conversational Human-Computer Interaction*. AI Magazine, 22(4). pp. 27–37.
- [Allen et al, 1999] Allen, J., Guinn, C., and Horvitz, E. 1999. *Mixed-Initiative Interaction*. IEEE Intelligent Systems, 14(5), pp. 14–23.
- [Almeida et al, 2002] Almeida, L., Amdal, I., Beires, N., Boualem, M., Boves, L., den Os, E., Filoche, P., Gomes, R., Knudsen, J. E., Kvale, K., Rugelbak, J., Tallec, C., and Warakagoda, N. 2002. *Implementing and evaluating a multimodal and multilingual tourist guide*. Intern. CLASS Workshop on Natural, Intelligent and Effective Interaction in Multimodal Dialogue Systems, pp. 1-7.
- [Araki and Tachibana, 2006] Araki, M., and Tachibana, K. 2006. *Multimodal Dialogue Description Language for Rapid System Development*. 7th SIGdial Workshop on Discourse and Dialogue. pp 109-116.
- [Atherton, 1999] Atherton, M. 1999. *Welsh today BSL tomorrow*. Deaf Worlds 15(1), pp. 11-15.
- [Bacchiani et al, 2006] Bacchiani, M., Riley, M., Roark, B., and Sproat, R. 2006. *MAP adaptation of stochastic grammars*. Computer Speech and Language, Volume 20(1), January 2006, pp 41-68.
- [Balci et al, 2007] Balci, K, Not, E., Zancanaro, M., and Pianesi, F. 2007. *Xface open source project and SMIL-agent scripting language for creating and animating embodied conversational agents*. ACM Multimedia 2007, pp. 1013-1016.
- [Banerjee and Lavie, 2005] Banerjee, S. and Lavie, A. 2005. *METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments*. Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization (ACL-2005), pp. 65-72.
- [Bangalore and Riccardi, 2000] Bangalore, S. and Riccardi, G. 2000. *Stochastic finite-state models for spoken language machine translation*. NAACL-ANLP 2000 Workshop on Embedded machine translation systems - Vol. 5, pp. 52-59.
- [Beasley et al, 2001] Beasley, R., Farley, K. M., O'Reily, J., and Squire, L. H. 2001. *Voice Application Development with VoiceXML*. Sams Publishing, 400 p. ISBN: 0-672-32138-6.
- [Bellegarda, 2004] Bellegarda, J. R. 2004. *Statistical language model adaptation: review and perspectives*. Speech Communication, vol. 42, pp. 93–108.
- [Bellegarda, 2000a] Bellegarda, J. R., 2000. *Exploiting latent semantic information in statistical language modelling*. Proc. IEEE 88 (8), pp. 1279–1296.
- [Bellegarda, 2000b] Bellegarda, J. R., 2000. *Large vocabulary speech recognition with multi-span statistical language models*. IEEE Trans. Speech Audio Proc. 8 (1), pp. 76–84.

- [Bennett et al, 2002] Bennett, C., Llitjós, A. F., Shriver, S., Rudnicky, A. and Black, A. W. 2002. *Building VoiceXML-Based Applications*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 2245-2248.
- [Besling and Meier, 1995] Besling, S., Meier, H-G. 1995. *Language model speaker adaptation*. European Conference on Speech Communication and Technology (Eurospeech), pp.1755-1758.
- [Bielefeld, 1994] Bielefeld, B. 1994. *Language identification using shifted delta cepstrum*. 14th Annual Speech Research Symposium.
- [Bigi et al, 2004] Bigi, B., Huang Y., and De Mori, R. 2004. *Vocabulary and Language Model Adaptation using Information Retrieval*. Intern. Conf. on Spoken Language Processing (ICSLP), pp.1361–1364.
- [Birch et al, 2007] Birch, A., Osborne, M., and Koehn, P. 2007. *CCG Supertags in Factored Statistical Machine Translation*. 2nd Workshop on Statistical Machine Translation, pp. 9–16.
- [Blei et al, 2003] Blei, D., Ng, A., and Jordan, M. 2003. *Latent Dirichlet Allocation*. Journal of machine Learning Research 3, pp. 993-1022.
- [Bohus and Rudnicky, 2003] Bohus, D., and Rudnicky, A. I. 2003. *RavenClaw: Dialogue Management Using Hierarchical Task Decomposition and an Expectation Agenda*. 8th European Conference on Speech Communication and Technology (Eurospeech), pp. 597-600.
- [Broman and Kurimo, 2005] Broman, S. and Kurimo, M. 2005. *Methods for Combining Language Models in Speech Recognition*. Interspeech, pp. 1317-1320.
- [Brown et al, 1993] Brown, P.F., Della Pietra, and Mercer, R. L. 1993. *The mathematics of statistical machine translation: Parameter estimation*. Computational Linguistics, Vol 19, No. 2, pp. 263-311.
- [Brown et al, 1992] Brown, P. F., Della Pietra, V. J., de Souza, P. V., Lai, J. C. and Mercer, R. L. 1992. *Class-based n-gram models of natural language*. Computational Linguistics, 18(4), pp. 467 - 479.
- [Bungeroth et al, 2006] Bungeroth, J., Stein, D., Dreuw, P., Zahedi, M., and Ney, H. 2006. *A German Sign Language Corpus of the Domain Weather Report*. 5th Intern. Conf. on Language Resources and Evaluation (LREC), pp 2000-20003.
- [Casacuberta and Vidal, 2006] Casacuberta, F. and Vidal, E. 2006. *Learning finite-state models for machine translation*. Machine learning, pp. 69-91. Ed. Springer Netherlands, ISSN 0885-6125.
- [Cassell et al, 2002] Cassell, J., Stocky, T., Bickmore, T., Gao, Y., Nakano, Y., Ryokai, K., Tversky, D., Vaucelle, C., and Vilhjálmsón, H. 2002. *MACK: Media lab Autonomous Conversational Kiosk*. Imagina: Intelligent Autonomous Agents, Monte Carlo, Monaco.
- [Cavnar and Trenkle, 1994] Cavnar, W. B. and Trenkle, J. M. 1994. *N-Gram-Based Text Categorization*. 3rd Symposium on Document Analysis and Information Retrieval, pp. 161-175.
- [Chelba and Jelinek, 2000] Chelba, C., and Jelinek, F. 2000. *Structured language modelling*. Computer, Speech, and Language 14 (4), pp. 283–332.

- [Chen et al, 1998] Chen, S. F., Beeferman, D., and Rosenfeld, R. 1998. *Evaluation metrics for language models*. DARPA Broadcast News Transcription and Understanding Workshop, pp. 275–280.
- [Chen and Goodman, 1998] Chen, S. F. and Goodman, J. 1998. *An Empirical Study of Smoothing Techniques for Language Modelling*. TR-10-98, Computer Science Group, Harvard University.
- [Chen, 2004] Chen, Y. 2004. *EVITA-RAD: an Extensible Enterprise Voice PorTAL – Rapid Application Development tool*. Interspeech 2004, pp. 3053-3056.
- [Chiu et al, 2007] Chiu, Y.-H., Wu, C.-H., Su, H.-Y., and Cheng, C.-J. 2007. *Joint Optimization of Word Alignment and Epenthesis Generation for Chinese to Taiwanese Sign Synthesis*. IEEE Trans. Pattern Analysis and Machine Intelligence, 29(1):28–39.
- [Chou and Juang, 2003] Chou, W., and Juang, B. H. eds. 2003. *Pattern recognition in Speech and Language Processing*. CRC Press. 416 pps. ISBN 0849312329.
- [Christopoulos and Bonvillian, 1985] Christopoulos, and C. Bonvillian, J., 1985. *Sign Language*. Journal of Communication Disorders, 18, pp. 1-20.
- [Chung, 2004] Chung, G. 2004. *Developing A Flexible Spoken Dialogue System Using Simulation*. 42nd Annual Meeting on Association for Computational Linguistics (ACL), pp. 63-70.
- [Clarkson and Robinson, 1997] Clarkson, P. and R., Robinson, A. J. 1997. *Language model adaptation using mixtures and an exponentially decaying cache*. Intern. Conf. on Acoustics, Speech, Signal Processing (ICASSP), pp. 799–802.
- [Cole et al, 2003] Cole, R., Van Vuuren, S., Pellom, B., Hacioglu, K., Ma, J., Movellan, J., Schwartz, S., Wade-Stein, D., Ward, W., and Yan, J. 2003. *Perceptive Animated Interfaces: First Steps toward a New Paradigm for Human Computer Interaction*. IEEE Transactions on Multimedia: Special Issue on Human Computer Interaction, Vol. 91(9), pp. 1391-1405.
- [Cole, 1999] Cole, R., 1999. *Tools for research and education in speech science*. Intern. Conf. of Phonetic Sciences (ICPhS), pp. 1277-1280.
- [Cordoba et al, 2005] Cordoba, R., Macías-Guarasa, J., Sama, V., Barra, R., and Pardo, J. M. 2005. *New Advances in Cross-Task and Speaker Adaptation for Air Traffic Control Tasks*. Revista de Procesamiento del Lenguaje Natural N° 35, pp. 21-27. ISSN:1135-5948.
- [Cordoba et al, 2003] Cordoba, R., Prime, G., Macías-Guarasa, J., Montero, J. M., Ferreiros, J., and Pardo, J. M. 2003. *PPRLM Optimization for Language Identification in Air Traffic Control Tasks*. Eurospeech, pp. 2685-2688.
- [Cordoba et al, 2002] Cordoba, R., Montero, J. M., Gutiérrez Arriola, J. M., Vallejo, J. A., Enríquez, E., and Pardo, J. M. 2002. *Selection of the most significant parameters for duration modelling in a Spanish text-to-speech system using neural networks*. Computer Speech and Language, Vol.16(2), pp 183-203.
- [Cordoba et al, 2001] Cordoba, R., San-Segundo, R., Montero, J. M., Colás, J., Ferreiros, J., Macías-Guarasa, J., and Pardo, J. M. 2001. *An Interactive Directory Assistance Service for Spanish with Large-Vocabulary Recognition*. 7th European Conference on Speech Communication and Technology (Eurospeech). Vol. II, pp. 1279-1282.

- [Dagan et al, 1993] Dagan, I., Church, K., and Gale, A. 1993. *Robust bilingual word alignment for machine-aided translation*. Workshop on very large corpora, pp. 1-8.
- [Denecke, 2002] Denecke, M. 2002. *Rapid Prototyping for Spoken Dialogue Systems*. 19th Int. Conf. on Computational Linguistic (COLING'02). pp. 1-7.
- [Deerwester et al, 1990] Deerwester, S., Dumais, S.T., Furnas, G. W., Landauer, T. K., Harshman, R., 1990. *Indexing by latent semantic analysis*. Journal of the American Society for Information Science, Vol 41, pp. 391–407.
- [Doddington, 2002] Doddington, G. 2002. *Automatic Evaluation of Machine Translation Quality Using N-gram Co-Occurrence Statistics*. 2nd Intern. Conf. on Human Language Technology Research, pp. 138 – 145.
- [Dybkjær and Dybkjær, 2006] Dybkjær, H. and Dybkjær, L. 2006. *DialogDesigner: tools support for dialogue model design and evaluation*. Language Resources and Evaluation, Vol. 40 (1), pp. 87-107.
- [Dybkjær and Dybkjær, 2005] Dybkjær, H. and Dybkjær, L. 2005. *DialogDesigner – A Tool for Rapid System Design and Evaluation*. 6th SIGdial Workshop on Discourse and Dialogue, pp. 227-231.
- [Eberman et al, 2002] Eberman, B., Carter, J., and Goddeau, D. 2002. *Building VoiceXML Browsers with OpenVXI*. 11th Intern. Conf. on World Wide Web, pp. 713 – 717.
- [Engberg-Pedersen, 2003] Engberg-Pedersen, E. 2003. *From pointing to reference and predication: pointing signs, eyegaze, and head and body orientation in Danish Sign Language*. Pointing: where language, culture, and cognition meet. Edited by Sotaro Kita, Mahwah, NJ: Lawrence Erlbaum Associates, pp. 269-292. ISBN:0805840141.
- [Federico, 1996] Federico, M. 1996. *Bayesian estimation methods for N-gram language model adaptation*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 240–243.
- [Feng et al, 2003] Feng, J., Bangalore, S., Rahim, M. 2003. *WEBTALK: Mining Websites for Automatically Building Dialogue Systems*. Workshop on Automatic Speech Recognition and Understanding (ASRU '03). pp. 168-173.
- [Ferreiros et al, 2005] Ferreiros, J., San-Segundo, R., Fernández, F., D'Haro, L.F., Sama, V., Barra, R., and P. Mellén. 2005. *New Word-Level and Sentence-Level Confidence Scoring Using Graph Theory Calculus and its Evaluation on Speech Understanding*. Interspeech, pp 3377-3380.
- [Flippo et al, 2003] Flippo, F., Krebs, A., and Marsic, I. 2003. *A framework for Rapid Development of Multimodal Interfaces*. 5th Intern. Conf. on Multimodal Interfaces, pp. 109 – 116.
- [Galescu et al, 1998] Galescu, L., Ringger, E. K., and Allen, J. F. 1998. *Rapid language model development for new task domains*. ELRA First Intern. Conf. on Language Resources and Evaluation (LREC), pp. 807-812.
- [Gauvain et al, 2004] Gauvain, J. L., Messaoudi, A., and Schwenk, H. 2004. *Language Recognition using Phone Lattices*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. I-25-28.
- [Georgila et al, 2004] Georgila, K., Fakotakis, N., and Kokkinakis, G. 2004. *A graphical tool for handling rule grammars in Java speech grammar format*. 4th Intern. Conf. on Language Resources and Evaluation.

- [Gildea and Hofmann, 1999] Gildea, D., and Hofmann, T., 1999. *Topic-based language modelling using EM*. Eurospeech, pp. 2167-2170.
- [Glass and Weinstein, 2001] Glass, J. and Weinstein, E. 2001. *SPEECHBUILDER: Facilitating Spoken Dialogue System Development*. European Conference on Speech Communication and Technology (Eurospeech), pp. 1335-1339.
- [Gleason and Zissman, 2001] Gleason, T. P., and Zissman, M.A.. 2001. *Composite background models and score standardization for Language Identification Systems*. Intern. Conf. Acoustics, Speech, Signal Processing (ICASSP), pp. 529-532.
- [Good, 1953] Good, I. J. 1953. *The population frequencies of species and the estimation of population parameters*. Biometrika, 40(3 and 4), pp. 237–264.
- [Goodman, 2001] Goodman J. T. 2001. *A bit of progress in language modelling*. Computer Speech and Language, Vol. 15(4), pp. 403-434(32).
- [Granström et al, 2002] Granström, B., House, D., Beskow, J., 2002. *Speech and Signs for Talking Faces in Conversational Dialogue Systems*. Multimodality in Language and Speech Systems. Kluwer Academic Publishers, pp 209-241.
- [Gustafson et al, 2000] Gustafson, J., Bell, L., Beskow, J., Boye, J., Carlson, R., Edlund, J., Granström, B., House, D., and Wiren, M. 2000. *AdApt – A multimodal conversational dialogue system in an apartment domain*. Intern. Conf. on Spoken Language Processing (ICSLP). pp. II -134–137.
- [Gustafson et al, 1998] Gustafson, J., Elmberg, P., Carlson, R. and Jonsson, A. 1998. *An educational dialogue system with a user controllable dialogue manager*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 33-37.
- [Hamerich, 2008] Hamerich, S. W. 2008. *From GEMINI to DiaGen: Improving Development of Speech Dialogues for Embedded Systems*. 9th SIGdial Workshop on Discourse and Dialogue - Association for Computational Linguistics (SIGdial - ACL), pp. 92-95.
- [Hamerich et al, 2003] Hamerich, S. W., Wang, Y.-F., Schubert, V., Schless, V., and Igel, S. 2003. *XML-Based Dialogue Descriptions in the Gemini Project*. Berliner XML-Tage, pp. 404-412.
- [Heeman, 1999] Heeman, P. 1999. *POS Tags and Decision Trees for Language Modelling*. Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pp 129-137.
- [Heidel et al, 2007] Heidel, A., Chang, H-a., and Lee, L-S. 2007. *Language Model Adaptation Using Latent Dirichlet Allocation and an Efficient Topic Inference Algorithm*. Interspeech, pp.2361-2364.
- [Herrero-Blanco and Salazar-García, 2005] Herrero-Blanco, Á., and Salazar-García, V. 2005. *Non-verbal predicability and copula support rule in Spanish Sign Language*. Casper De Groot & Kees Hengeveld (eds.) Morphosyntactic Expression in Functional Grammar (Functional Grammar Series, 27). Berlín: Mouton de Gruyter, pp. 281-315.
- [Hofmann, 1999] Hofmann, T. 1999. *Probabilistic Latent Semantic Indexing*. 22nd Annual International SIGIR. Conference on Research and Development in Information Retrieval (SIGIR-99), pp.50-57.

- [Hurtado et al, 2005] Hurtado L. F., Blat F., García F., Grau S., Griol D., Sanchis E., Segarra E., and Torres F. 2005. *Sistema de diálogo para el Proyecto DIHANA*. Revista del Procesamiento del lenguaje natural N° 35, pp. 453-454.
- [Hutchins, 2005] Hutchins, J. 2005. *Towards a definition of example-based machine translation*. Workshop on Example-Based Machine Translation, pp.63-70.
- [Ito et al, 2006] Ito, A., Shimada, K., Suzuki, M., and Makino, S. 2006. *A User Simulator based on VoiceXML for evaluation of spoken dialogue systems*. Interspeech, pp 1045-1048.
- [Iyer and Ostendorf, 1999] Iyer, R. and Ostendorf, M. 1999. *Modelling long distance dependence in language: Topic mixture vs. dynamic cache models*. IEEE Trans. Speech Audio Processing, vol. 7, pp. 30–39.
- [Jelinek and Mercer, 1980] Jelinek, F. and Mercer, R.L. 1980. *Interpolated estimation of Markov source parameters from sparse data*. Gelsema, E.S and Kanal, L.N. (Eds). Workshop on Pattern Recognition in Practice, pp. 381-397.
- [Jelinek et al, 1991] Jelinek, F., Roukos, S., Merialdo, B., and Strauss, M. 1991. *A dynamic language model for speech recognition*. DARPA Workshop on Speech and Natural Language, pp. 293–295.
- [Jelinek, 1990] Jelinek, F. 1990. *Self-organized language modelling for speech recognition*. Readings in speech recognition, pp. 450 - 506.
- [Jemni and Elghoul, 2007] Jemni, M., and Elghoul, O. 2007. *Towards Web-Based automatic interpretation of written text to Sign Language*. 1st Intern. Conf. on ICT and Accesibility (ICTA) 2007, pp. 43-48.
- [Jiang, 2005] Jiang, H. 2005. *Confidence measures for speech recognition: A survey*. Speech Communication, Vol. 45, Issue 4, April 2005, pp 455-470.
- [Johnston et al, 2007] Johnston, M., D’Haro, L.F., Levine, M., and Renger, B. 2007. *A Multimodal Interface for Access to Content in the Home*. 45th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 376-383.
- [Johnston et al, 2002] Johnston, M., Bangalore, S, Vasireddy, G., Stent, A., Ehlen, P., Walker, M., Whittaker, S. Maloor, P. 2002. *MATCH: An architecture for multimodal dialogue systems*. 40th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 376–383.
- [Jung et al, 2008] Jung, S., Lee, C., Kima, S., and Geunbae Lee, G. 2008. *DialogStudio : A Workbench for Data-driven Spoken Dialogue System Development and Management*. Speech Communications, 50 (8-9), pp. 683-697.
- [Jurafsky and Martin, 2008] Jurafsky, D., and Martin, J. 2008. *Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall; 2nd edition, 1024 pages. ISBN: 0131873210
- [Katsurada et al, 2003] Katsurada, K., Nakamura, Y., Yamada, H., and Nitta, T. 2003. *XISL: a language for describing multimodal interaction scenarios*. 5th Intern. Conf. on Multimodal interfaces. pp. 281–284.
- [Katsurada et al, 2002] Katsurada, K., Ootani, Y., Nakamura, Y., Kobayashi, S., Yamada, H., and Nitta, T. 2002. *A Modality Independent MMI System Architecture*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 2549-2552.

- [Katz, 1987] Katz, S. 1987. *Estimation of probabilities from sparse data for the language model component of a speech recognizer*. IEEE Transactions on Acoustics, Speech and Signal Processing, ASSP-35(3), pp. 400–401.
- [Keller and Lapata, 2003] Keller, F. and Lapata, M. 2003. *Using the Web to obtain frequencies for unseen bigrams*. Computational Linguistics. Vol. 29(3), pp. 459-484.
- [Klakow and Peters, 2002] Klakow, D., and Peters, J. 2002. *Testing the correlation of word error rate and perplexity*. Speech Communication, Vol.38(1-2), September 2002, pp 19-28.
- [Klemmer et al, 2000] Klemmer, S.R., Sinha, A K., Chen, J., Landay, J. A., Aboobaker, N., Wang, A. 2000. *SUEDE: a Wizard of Oz prototyping tool for speech user interfaces*. In: CHI Letters, ACM Symposium on User Interface Software and Technology (UIST), Vol. 2 (2), pp. 1–10.
- [Kneser et al, 1997] Kneser, R., Peters J. and Klakow, D. 1997. *Language Model Adaptation Using Dynamic Marginals*. European Conference on Speech Communication and Technology (Eurospeech), Vol 4, pp. 1971-1974.
- [Kneser, 1996] Kneser, R. 1996. *Statistical Language Modelling Using a Variable Context Length*. 4th Intern. Conf. on Spoken Language Processing, Vol. 1, pp. 494-497.
- [Kneser and Ney, 1995] Kneser, R. and Ney, H. 1995. *Improved backing-off for m-gram language modelling*. IEEE Intern. Conf. on Acoustics, Speech and Signal Processing, volume 1, pp 181–184.
- [Kneser and Ney, 1993] Kneser, R. and Ney, H. 1993. *Improved Clustering Techniques for Class-Based Statistical Language Modelling*. European Conference on Speech Communication and Technology (Eurospeech), pp. 973-976.
- [Knight, 1999] Knight, K. 1999. *A statistical machine translation workbook*. Unpublished. Available online at <http://www.isi.edu/~knight/#pubs> [12/12/08].
- [Koehn et al, 2006] Koehn, P., Federico, M., Shen, W., et al. 2006. *Open source toolkit for Statistical Machine Translation: Factored Translation Models and Confusion Network Decoding*. Final report of the 2006 Language Engineering Workshop. Available online at <http://www.clsp.jhu.edu/ws2006/groups/ossmt/> [12/12/2008]
- [Koehn, 2005] Koehn, P. 2005. *Europarl: A Parallel Corpus for Statistical Machine Translation*. MT Summit X.
- [Koehn, 2004] Koehn, P. 2004. *Pharaoh: a Beam Search Decoder for Phrase-Based Statistical Machine Translation Models*. 6th Conference of the Association for Machine Translation in the Americas (AMTA-04), pp. 115-124.
- [Koehn et al, 2003] Koehn, P., Och, F. J., and Marcu, D. 2003. *Statistical Phrase-Based Translation*. Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (HLT/NAACL). Vol. 1, pp. 48-54.
- [Komatani et al, 2003] Komatani, K., Ueno, S., Kawahara, T., and Okuno, H. G. 2003. *User Modelling in Spoken Dialogue Systems for Flexible Guidance Generation*. European Conference on Speech Communication and Technology (Eurospeech), pp. 745-748.
- [Kuhn, 1988] Kuhn, R. 1988. *Speech recognition and the frequency of recently used words: A modified Markov model for natural language*. 12th Intern. Conf. Computational Linguistics, pp. 348–350.

- [Lamel et al, 2000] Lamel, L., Rosset, S., Gauvain, J. L., Bennacef, S., Garnier-Rizet, M., and Prouts. B. 2000. *The LIMSI ARISE System*. Speech Communication, Vol 31(4): pp. 339-354.
- [Larsson and Traum, 2000] Larsson, S. and Traum, D. 2000. *Information state and dialogue management in the TRINDI dialogue move engine toolkit*. Natural Language Engineering Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering, Cambridge University Press, U.K., pp.323-340.
- [Lau et al, 1993] Lau, R., Rosenfeld, R., and Roukos, S., 1993. *Trigger-based language models: a maximum entropy approach*. Intern. Conf. Acoustics, Speech, and Signal Process (ICASSP), pp. 1145–1148.
- [Lehtinen et al, 2000] Lehtinen, G., Safra, S., Gauger, M., Cochard, J.-L., Kaspar, B., Hennecke, M.E., Pardo, J.M., Cordoba, R., San-Segundo, R., Tsopanoglou, A., Louloudis, D., and Mantazas, M. 2000. *IDAS: interactive directory assistance service*. Proc. COST249. ISCA Workshop on Voice Operated Telecom Services (VOTS), pp. 51–54.
- [Levin et al, 2000] Levin, E., Narayanan, S. Pieraccini, R., Biatov, K., Bocchieri, E., Di Fabbrizio, G., Eckert, W., Lee, S. Pokrovsky, A., Rahim, M., Ruscitti, P., and Walker, M. 2000. *The AT&T-DARPA communicator mixed-initiative spoken dialogue system*. Intern. Conf. on Spoken Language Processing (ICSLP). Vol. 2, pp. 122-125.
- [Li and Lin, 2006] Li, Y-X, and Lin, N-W. 2006. *Voice Composer: A Development Tool for Voice Applications*. International Computer Symposium, pp. 304-309.
- [Li et al, 2006] Li, J., Yaman, S., Lee, C.-H., Ma, B., Tong, R., Zhu, D., and Li, H. 2006. *Language Recognition Based on Score Distribution Feature Vectors and Discriminative Classifier Fusion*. IEEE Odyssey 2006: The Speaker and Language Recognition Workshop. pp. 1-5.
- [López-Cozar and Araki, 2005] López-Cózar, R., and Araki, M. 2005. *Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment*. 262 pp. Published by John Wiley & Sons, ISBN: 0-470-02155-1.
- [López-Cozar et al, 2005] López-Cózar, R., Callejas, Z., Gea M., and Montoso, G. 2005. *Multimodal, Multilingual and Adaptive Dialogue System for Ubiquitous Interaction in an Educational Space*. ISCA Workshop (ITRW) on Applied Spoken Language Interaction in Distributed Environments, ISSN 0908-1224.
- [López-Cozar and Granell, 2004] López-Cózar, R., and Granell, R. 2004. *Sistema de Diálogo Basado en VoiceXML para Proporcionar Información de Viajes en Tren*. Procesamiento del lenguaje natural, N°. 33, pp. 171-178.
- [López-Moreno et al, 2008] López-Moreno, I., Ramos, D., González-Rodríguez, J., and Toledano, D. T. 2008. *Anchor-Model fusion for language recognition*. Interspeech 2008, pp. 727-730.
- [Ma et al, 2005] Ma, B., Li, H., and Lee, C-H. 2005. *An acoustic segment modeling approach to automatic language identification*. Interspeech 2005. pp. 2829-2832.
- [Ma et al, 2002] Ma, J., Yan, J., and Cole, R. 2002. *CU animate tools for enabling conversations with animated characters*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 197-200.

- [Manning and Schütze, 1999] Manning, C. D., and Schütze, H. 1999. *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA. 680 pp. ISBN: 0262133601.
- [Masataka, et al, 2006] Masataka, N, Ohnishi, T., Imabayashi, E., Hirakata, M., and Matsuda, H. 2006. *Neural correlates for numerical processing in the manual mode*. Journal of Deaf Studies and Deaf Education 11(2), pp. 144-152.
- [McTear, 2004] McTear, M. 2004. *Spoken Dialogue Technology: Towards the conversational user interface*. Published by Springer Ed. 432 pp. ISBN: 1-85233-672-2.
- [McTear, 2002] McTear, M. 2002. *Spoken Dialogue Technology: Enabling the Conversational User Interface*. ACM Computing Surveys, 34(1). pp. 90–169.
- [McTear, 1999] McTear, M. 1999. *Software to Support Research and Development of Spoken Dialogue Systems*. European Conference on Speech Communication and Technology (Eurospeech), pp. 339-342.
- [McTear, 1998] McTear, M. 1998. *Modelling Spoken Dialogues with State Transition Diagrams: Experiences with the CSLU Toolkit*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 1223-1226.
- [Meurant, 2004] Meurant, L. 2004. *Anaphora, role shift and polyphony in Belgian sign language*. Intern. Conf. on Theoretical Issues in Sign Language Research 8, pp. 113-115.
- [Mohri, 2000] Mohri, M. 2000. *Minimization algorithms for sequential transducers*. Theoretical Computer Science, Vol 234 (1–2), pp. 177–201.
- [Monserrat and Gallardo, 2004] Monserrat, V., and Gallardo, B. 2004. V., *Estudios Lingüísticos sobre la lengua de signos española*. Universidad de Valencia. Ed. AGAPEA. ISBN: 8437055261.
- [Montero et al, 2003] Montero, J. M., D’Haro, L.F., Cordoba, R., Vallejo, J. A., Gutiérrez-Arriola, J., and Pardo, J. M. 2003. *ANN F0 Modelling for Female-Voice Synthesis in Spanish: restricted and non-restricted domains*. 15th International Congress of Phonetic Sciences (ICPS), pp. 563-566.
- [Moore et al, 2006] Moore, R. C., Yih, W., and Bode, A. 2006. *Improved discriminative bilingual word alignment*. 21th Intern. Conf. on Computational Linguistics and 44th Annual Meeting of the ACL, pp. 513-520.
- [Morrissey and Way, 2005] Morrissey, S., and Way, A. *An Example-Based Approach to Translating Sign Language*. Workshop Example-Based Machine Translation (MT X05), pp. 109-116.
- [Nagarajan and Murthy, 2004] Nagarajan, T., and Murthy, H. A. 2004. *Language Identification Using Parallel Syllable-Like Unit Recognition*. Intern. Conf. Acoustics, Speech, and Signal Processing (ICASSP), pp. I-401-404.
- [Navratil, 2001] Navratil, J. 2001. *Spoken Language Recognition – A Step toward Multilinguality in Speech Processing*. IEEE Transactions on Speech and Audio Processing, Vol. 9(6), pp. 678-685.
- [Navratil and Zühlke, 1997] Navratil, and J. Zühlke, W. 1997. *Double bigram-decoding in phonotactic language identification*. Intern. Conf. on Acoustics, Speech, Signal Processing (ICASSP), Vol. 2, pp. 1115–1118.

- [Ney et al, 1994] Ney, H., Essen, U., and Kneser, R. 1994. *On structuring probabilistic dependences in stochastic language modelling*. Computer, Speech and Language. Vol 8, pp. 1-38.
- [Nigay and Coutaz, 1993] Nigay, L., and Coutaz, J. 1993. *A design space for multimodal systems - concurrent processing and data fusion*. INTERCHI - Conference on Human Factors in Computing Systems, pp. 172-178.
- [Nyst, 2004] Nyst, V. 2004. *Verbs of motion in Adamorobe Sign Language*. Intern. Conf. on Theoretical Issues in Sign Language Research 8. pp. 127-129.
- [Och and Ney, 2004] Och, F.J., and Ney, H. 2004. *The Alignment Template Approach to Statistical Machine Translation*. Computational Linguistics, Vol 30 (4), pp. 417-449.
- [Och and Ney, 2003] Och, F.J., and Ney, H. 2003. *A Systematic Comparison of Various Statistical Alignment Models*. Computational Linguistics, Vol. 29(1), pp. 19-51.
- [Och and Ney, 2002] Och, F.J., and Ney, H. 2002. *Discriminative training and maximum entropy models for statistical machine translation*. 40th Annual Meeting of the Association for Computational Linguistics (ACL), pp 295–302.
- [Och and Ney, 2000a] Och, F.J., and Ney, H. 2000. *Improved Statistical Alignment Models*. 38th Annual Meeting of the Assoc. for Computational Linguistics (ACL), pp. 440–447.
- [Och and Ney, 2000b] Och, F. J., and Ney, H. 2000. *A comparison of alignment models for statistical machine translation*. 18th Intern. Conf. on Computational Linguistics, pp. 1086-1090.
- [Och, 1999] Och, F. J. 1999. *An Efficient Method for Determining Bilingual Word Classes*. 9th Conf. of the European Chapter of the Association for Computational Linguistics; (EACL), pp. 71-76.
- [Och et al, 1999] Och, F.J., Tillmann, C., and Ney, H. 1999. *Improved alignment models for statistical machine translation*. Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora, pp 20-28.
- [Oviatt et al, 2000] Oviatt, S. L., Cohen, P., Wu, L., Vergo, J., Duncan, L., Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J., and Ferro, D. 2000. *Designing the user interface for multimodal speech and gesture applications: state-of-the-art systems and research directions*. Journal of Human Computer Interaction, Vol 15(4), pp. 263-322.
- [Padró and Padró, 2004] Padró Cirera, L., Padró, M. 2004. *Comparing methods for language identification*. Procesamiento del lenguaje natural, N° 33, pp. 155-161.
- [Papineni et al, 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W. J. 2002. *BLEU: a Method for Automatic Evaluation of Machine Translation*. 40th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 311-318.
- [Pardo et al, 1995] Pardo, J. M., Giménez de los Galanes, F. M., Vallejo, J. A., Berrojo, M. A., Montero, J. M., Enríquez, E., and Romero, A. 1995. *Spanish text-to-speech, from prosody to acoustics*. 15th International Congress on Acoustics, pp. 133–136.
- [Pargellis et al, 2004] Pargellis, A. N., Kuo, H. J., and Lee, C. 2004. *An automatic dialogue generation platform for personalized dialogue applications*. Speech Communication Vol. 42, pp. 329-351.
- [Parkhurst and Parkhurst, 2007] Parkhurst, S., and Parkhurst, D. 2007. *Spanish Sign Language Survey*. SIL Electronic Survey Reports 2007-008, 85 p.

- [Pereira and Riley, 1997] Pereira, F.C., and Riley, M., 1997. *Speech recognition by composition of weighted finite automata*. Roche, E., Schabes, Y. (Eds.), *Finite-State Language Processing*. MIT Press, Cambridge, MA, pp. 431–453.
- [Polifroni and Walker, 2006] Polifroni, J. and Walker, M. 2006. *Learning Database Content for Spoken Dialogue System Design*. Intern. Conf. on Language Resources and Evaluation (LREC), pp. 143-148.
- [Polifroni et al, 2003] Polifroni, J., Chung, G., Seneff, S. 2003. *Towards the Automatic Generation of Mixed-Initiative Dialogue Systems from Web Content*. European Conference on Speech Communication and Technology (Eurospeech), pp. 193–196.
- [Polifroni et al, 2000] Polifroni, J., Seneff, S. 2000. *Galaxy-II as an architecture for Spoken Dialogue Evaluation*. Intern. Conf. on Language Resources and Evaluation (LREC), pp. 725–730.
- [Prillwitz et al, 1989] Prillwitz, S., R. Leven, H. Zienert, T. Hanke, J. Henning, et al. 1989. *Hamburg Notation System for Sign Languages – An introductory Guide*. Intern. Studies on Sign Language and the Communication of the Deaf, Volume 5. Institute of German Sign Language and Communication of the Deaf, University of Hamburg.
- [Pyers, 2006] Pyers J.E., 2006. *Indicating the body: Expression of body part terminology in American Sign Language*. *Language Sciences*, Vol. 28, no 2-3, pp. 280-303. ISSN 0388-0001.
- [Ramasubramaniam et al, 2003] Ramasubramaniam, V., Sai Jayram, A. K. V., and Sreenivas, T. V. 2003. *Language Identification using Parallel Phone Recognition*. Workshop on Spoken Language Processing, pp. 109-116.
- [Reyes, 2005] Reyes, I. 2005. *Comunicar a través del silencio: las posibilidades de la lengua de signos española*. Universidad de Sevilla, 310 p.
- [Rodríguez, 1991] Rodríguez, M.A. 1991. *Lenguaje de signos*. Phd Dissertation. Confederación Nacional de Sordos Españoles (CNSE) and Fundación ONCE. Madrid. Spain.
- [Rosenfeld, 2000] Rosenfeld, R. 2000. *Two Decades of Statistical Language Modelling: Where Do We Go from Here?*. Proceedings of the IEEE, Vol. 88(8), pp. 1270-1278.
- [Rosenfeld, 1996] Rosenfeld, R. 1996. *A maximum entropy approach to adaptive statistical language modelling*. *Computer Speech and Language*, Vol. 10, pp. 187–228.
- [Rudnicky and Xu, 1999] Rudnicky, A., and Xu W. 1999. *An agenda based dialogue management architecture for spoken language systems*. IEEE Automatic Speech Recognition and Understanding Workshop, pp. 337-340.
- [Sai-Jayram et al, 2003] Sai Jayram, K. V., Ramasubramanian, V., and Sreenivas, T. V. 2003. *Language Identification Using Parallel Sub-Word Recognition*. Intern. Conf. on Acoustics, Speech, and Signal Processing (ICASSP), pp. I-32-35.
- [San-Segundo et al, 2001a] San Segundo, R., Montero, J. M., Colás, J., Gutiérrez, J. M., Ramos, J. M., and Pardo, J. M. 2001. *Methodology for Dialogue Design in Telephone-Based Spoken Dialogue Systems: A Spanish Train Information System*. European Conference on Speech Communication and Technology (Eurospeech), pp 2165-2168.
- [San-Segundo et al, 2001b] San-Segundo, R. Pellom, B., Hacıoglu, K., Ward, W., and Pardo, J. M. 2001. *Confidence measures for spoken dialogue systems*. Intern. Conf. on Acoustics, Speech, Signal Processing (ICASSP), pp 393-396.

- [Sarikaya et al, 2005] Sarikaya, R., Gravano, A., and Gao, Y. 2005. *Rapid Language Model Development Using External Resources for New Spoken Dialogue Domains*. Intern. Conf. on Acoustics, Speech, Signal Processing (ICASSP). Vol. 1, pp. 573- 576.
- [Sato and Nagao, 1990] Sato, S., and Nagao, M. 1990. *Towards memory-based translation*. 13th conference on Computational linguistics, Vol. 3, pp. 247-252.
- [Scholz, 2006] Scholz, K. W. 2006. *Speech Service Creation: An overview of Speech Services Creation Tools*. NY/NJ Chapter Meeting, Avios Co. December 12, 2006.
- [Schubert et al, 2005] Schubert, V., and Hamerich, S. W. 2005. *The Dialogue Application Metalanguage GDialogXML*. European Conference on Speech Communication and Technology (Eurospeech), pp. 789-792.
- [Schwenk, 2007] Schwenk, H. 2007. *Continuous space language models*. Computer Speech and Language, Vol. 21, Issue 3. pp 492-518.
- [Seneff and Polifroni, 2000] Seneff, S., and Polifroni, J. 2000. *Dialogue management in the mercury flight reservation system*. ANLP-NAACL Satellite Workshop, pp. 1-6.
- [Sommers, 1999] Somers, H.L. 1999. *Example-based machine translation*. Machine Translation 14 (2): 113-158.
- [Stein et al, 2007] Stein, D., Dreuw, P., Ney, H., Morrisey, S., and Way, A. 2007. *Hand in hand: Automatic Sign Language to English Translation*. TMI 2007, pp.214-220.
- [Stein et al, 2006] Stein, D., Bungeroth, J., and Ney H. 2006. *Morpho-Syntax Based Statistical Methods for Sign Language Translation*. 11th Annual conference of the European Association for Machine Translation, pp. 169–177.
- [Stolcke, 2002] Stolcke, A. 2002. *SRILM – An Extensible Language Modelling Toolkit*. Intern. Conf. on Spoken Language Processing (ICSLP), 2, pp. 901–904.
- [Stokoe, 1960] Stokoe, W., 1960. *Sign Language structure: an outline of the visual communication systems of the American deaf*. *Studies in Linguistics*. Buffalo, Univ. Paper 8.
- [Strik et al, 1997] Strik, H., Russel, A., van den Heuvel, H., Cucchiarini, C., and Boves, L. 1997. *A spoken dialogue system for the Dutch public transport information service*. Intern. Journal of Speech Technology, Vol. 2 (2), pp. 121-131.
- [Sumita, 2001] Sumita, E. 2001. *Example-based machine translation using DP-matching between word sequences*. Data-Driven Machine translation Workshop, 39th Annual Meeting of the Assoc. for Computational Linguistics, pp. 1-8.
- [Tam and Schultz, 2006] Tam, Y-C, and Schultz, T. 2006. *Unsupervised Language Model Adaptation Using Latent Semantic Marginals*. Interspeech 2006. pp. 2206-2209
- [Tam and Schultz, 2005] Tam, Y-C, and Schultz, T. 2005. *Dynamic Language Model Adaptation using Variational Bayes Inference*. Interspeech 2005. pp. 5-8.
- [Tillmann et al, 1997] Tillmann, C., Vogel, S., Ney, H., Zubiaga, A., and Sawaf, H. 1997. *Accelerated DP based search for statistical translation*. European Conference on Speech Communication and Technology (Eurospeech), pp. 2667–2670.
- [Timmermans, 2005] Timmermans, N. 2005. *The status of the sign language in Europe*. Council Of Europe Publishing, 164 pp. ISBN: 978-92-871-5720-1.

- [Torres-Carrasquillo et al, 2002a] Torres-Carrasquillo, P. A., Reynolds, D. A., and Deller Jr., J. R. 2002. *Language identification using Gaussian mixture model tokenization*. Intern. Conf. on Acoustics, Speech, Signal Processing (ICASSP), pp. I-757-760.
- [Torres-Carrasquillo et al, 2002b] Torres-Carrasquillo, P. A., Singer, E., Kohler, M. A., Green R. J., et al. 2002. *Approaches to Language Identification using Gaussian Mixture Models and Shifted Delta Cepstral Features*. Intern. Conf. on Acoustics, Speech, Signal Processing (ICASSP), pp.89-92.
- [Toth et al, 2002] Toth, A. R., Harris, T. K., Sanders, J., Shriver, S., and Rosenfeld, R. 2002. *Towards every-citizen's speech interfaces: an application generator for speech interfaces to databases*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 1497–1500.
- [Tsai, 2006] Tsai, M. J. 2006. *VoiceXML dialogue system of the multimodal IP-Telephony – The application for voice ordering service*. Experts Systems with Applications 31, pp. 684-696.
- [Turunen et al, 2004] Turunen, M., et al. 2004. *AthosMail - A Multilingual Adaptive Spoken Dialogue System for E-mail Domain*. Workshop on Robust and Adaptive Information Processing for Mobile Speech Interfaces. pp. 77-86.
- [Uebler, 2001] Uebler, U. 2001. *Multilingual speech recognition in seven languages*. Speech Communication, Vol. 35 (1), pp. 53-69.
- [Vogel et al, 1996] Vogel, S., Ney, H., and Tillmann, C. 1996. *HMM-based word alignment in statistical translation*. 16th Intern. Conf. on Computational Linguistics, pp. 836-841.
- [Wahlster (Ed.), 2006] Wahlster, W. 2006. *SmartKom: Foundations of Multimodal Dialogue Systems*. 644 p. ISBN: 978-3-540-23732-7.
- [Wang, 2002] Wang, K. 2002. *Salt: A Spoken Language Interface for Web-Based Multimodal Dialogue Systems*. Intern. Conf. on Spoken Language Processing (ICSLP), pp. 2241-2244.
- [Wang and Stolcke, 2007] Wang, W., and Stolcke, A. 2007. *Integrating MAP, Marginals, and Unsupervised Language Model Adaptation*. Interspeech 2007, pp. 618-621.
- [Wang and Acero, 2006] Wang, Y., and Acero, A. 2006. *Rapid development of spoken language understanding grammars*. Speech Communication, Vol. 48(3-4), pp 390-416.
- [Wang et al, 2003] Wang, Y.-F. H., Hamerich, S. W., and Schless, V. 2003. *Multi-Modal and Modality Specific Error Handling in the GEMINI Project*. Workshop on Error Handling in Spoken Dialogue Systems, pp. 139-144.
- [Witten and Bell, 1991] Witten, I., and Bell, T. 1991. *The zero-frequency problem: Estimating the probability of novel events in adaptive text compression*. IEEE Transactions on Information Theory. 37(4), pp. 1085-1094.
- [Yamada and Knight, 2001] Yamada, K., and Knight, K. 2001. *A syntax-based statistical translation model*. 39th Annual Meeting of the Assoc. for Computational Linguistics (ACL), pp. 523-530.
- [Yin et al, 2006] Yin, B., Ambikairajah, E., and Chen, F. 2006. *Combining cepstral and Prosodic Features in Language Identification*. 18th Intern. Conf. on Pattern Recognition. Vol 4, 254-257.

- [Yu et al, 2005] Yu, D., Mahajan, M., Mau, P., and Acero, A. 2005. *Maximum Entropy Based Generic Filter for Language Model Adaptation*. Intern. Conf. on Acoustics, Speech, and Signal Processing (ICASSP), Vol.1, pp. 597- 600.
- [Zissman and Berkling, 2001] Zissman, M. A., and Berkling, K. M. 2001. *Automatic Language Identification*. Speech Communication 35, Issues 1-2, pp. 115-124.
- [Zissman, 1996] Zissman, M.A. 1996. *Comparison of four approaches to automatic language identification of telephone speech*. IEEE Trans. Speech and Audio Processing, Vol. 4(1), pp. 31-44.
- [Zhao et al, 2004] Zhao, B., Eck, M., and Vogel, S. 2004. *Language model adaptation for statistical machine translation with structured query models*. 20th Intern. Conf. on Computational Linguistics (COLING), pp 411-417.
- [Zhao et al, 2000] Zhao, L., Kipper, K., Schuler, W., Vogler, C., Badler, N. and Palmer, M. 2000. *A machine translation system from English to American Sign Language*. AMTA pp. 54-67.
- [Zhu and Rosenfeld, 2001] Zhu, X., and Rosenfeld, R. 2001. *Improving trigram language modelling with the World Wide Web*. Intern. Conf. on Acoustics Speech and Signal Processing (ICASSP), Vol.1, pp 533-536.
- [Zue et al, 2000] Zue, V., Seneff, S., Glass, J., Polifroni, J., Pao, C., Hazen, T. J., and Hetherington, L. 2000. *JUPITER: A telephone-based conversational interface for weather information*. IEEE Transactions on Speech and Audio Processing. Vol. 8(1), pp. 85–96.

APPENDIX A. LIST OF ABBREVIATIONS

ADA	Application Description Assistant
AGP	Application Generation Platform
ANN	Artificial Neural Network
API	Application Programming Interface
ASL	American Sign Language
ASR	Automatic Speech Recognizer
BLEU	BiLingual Evaluation Understudy
BNF	Backus-Naur Form
BOS	Bag-Of-Sounds models
BSL	British Sign Language
CCXML	Call Control Extensible Markup Language
CFG	Context Free Grammars
CGI	Common Gateway Interface
CTI	Computer Telephony Integration
DB	DataBase
DCMA	Data Connector Model Assistant
DM	Dialogue Manager
DMA	Data Model Assistant
DML	Data Model Linker
DNI	National Identity Document
DTMF	Dual-Tone Multi-Frequency
EBMT	Example Based Machine Translation
FIA	Form Interpretation Algorithm
GDialogXML	Gemini Dialogue eXtensible Markup Language
GEMINI	Generic Environment for Multilingual Interactive Natural Interfaces
GMM	Gaussian Mixture Model
GSL	Nuance Grammar Specification Language
GUI	Graphical User Interface
HMM	Hidden Markov Models
IDE	Integrated Development Environment
IR	Information Retrieval
IVR	Interactive Voice Response
J2EE	Java 2 Platform, Enterprise Edition
JDBC	Java DataBase Connectivity
JSGF	Java Speech Grammar Format
LDA	Latent Dirichlet Allocation
LID	Language IDentification
LM	Language Model
LMT	Language Modelling Toolkit
LSA	Latent Semantic Analysis
LSE	Lengua de Signos Española
MAP	Maximum-A-Posteriori
MEA	Modality and Language Extension Assistant
MERA-Speech	Modality Extension Retrieval Assistant for Speech

MI	Mixed-Initiative
MRCP	Media Resource Control Protocol
MT	Machine Translation
NLG	Natural Language Generator
NLU	Natural Language Understanding
ODBC	Open DataBase Connectivity
OOV	Out-Of-Vocabulary
OV	Over-Answering
PER	Position independent word Error Rate
PLP	Perceptual Linear Predicative
POS	Part-Of-Speech
PPL	Perplexity
PPRLM	Parallel Phone Recognition followed by Language Modelling
PSTN	Public Switched Telephone Network
RMA	Retrieval Model Assistant
SALT	Speech Application Language Tags
SAMPA	Speech Assessment Methods Phonetic Alphabet
SFMA	State Flow Model Assistant
SIP	Session Initiation Protocol
SISR	Semantic Interpretation for Speech Recognition
SLM	Statistical Language Model
SMT	Statistical Machine Translation
SQL	Structured Query Language
SRGS	Speech Recognition Grammar Specification
SSML	Speech Synthesis Markup Language
TTS	Text-To-Speech
UMA	User Model Assistant
VB	Vocabulary Builder
WER	Word Error Rate
XSLT	eXtensible Stylesheet Language Transformations

APPENDIX B. ADDITIONAL INFORMATION ABOUT CURRENT COMMERCIAL AND WEB-BASED PLATFORMS

In this appendix, we provide an overview of the main features and accelerations included in several commercial and Web-based development platforms that we studied during the development of this thesis. This appendix complements the studied we have presented in section 2.1.1 (page 8).

B.1 Commercial Platforms

Audium Studio⁷⁰: It is an open Integrated Development Environment (IDE) that allows developers the creation of interactive and dynamic server-based services. The Audium platform consists of two main components: Audium Server and Audium Builder.

- Audium Server provides the runtime platform, backend connectivity, application management, and manages the dynamic generation of the application.
- The Audium Builder interface allows the complete design of the dialogue flow using a graphical tree view representation (see Figure B.1), which is created using drag and drop, zoom and right-click commands, and through the customization of the different VoiceXML elements incorporated into the design. Besides, Audium Builder also allows local and remote application deployment, access to backend databases, performing transactions from any Web service or XML-capable system, and it can be easily integrated with all major VoiceXML gateway vendors and speech solution providers. Finally, Audium Builder includes an extensible library of reusable components, known as module inventory, which can be used to create the call flow. The platform also includes components for menus, recognizing basic inputs, playing back audio prompts, running external applications, etc. Each component can be either configured to meet the particular needs of the service, or dynamically configured to act differently according to pre-defined business rules.

One of the most interesting features included in Audium is the possibility of creating dynamic applications using dynamic module configurations, rules, worklets, or new modules. All of them can be created/used through java classes or Web based HTTP/XML exchange. In detail, rules are used to change the call-flow sequence (i.e. similar to if-then-else conditions). Dynamic configurations are used to change the VoiceXML content (for instance, to provide dynamic content for prompts in the application). Worklets are used to perform background processing without changing the call-flow (i.e. similar to the functionality provided by ECMA-scripts). Finally, new modules allow the integration of functionalities that the platform does not provide (for instance, speech verification) or to create new reusable modules.

⁷⁰ http://www.audiumcorp.com/Audium_Studio/

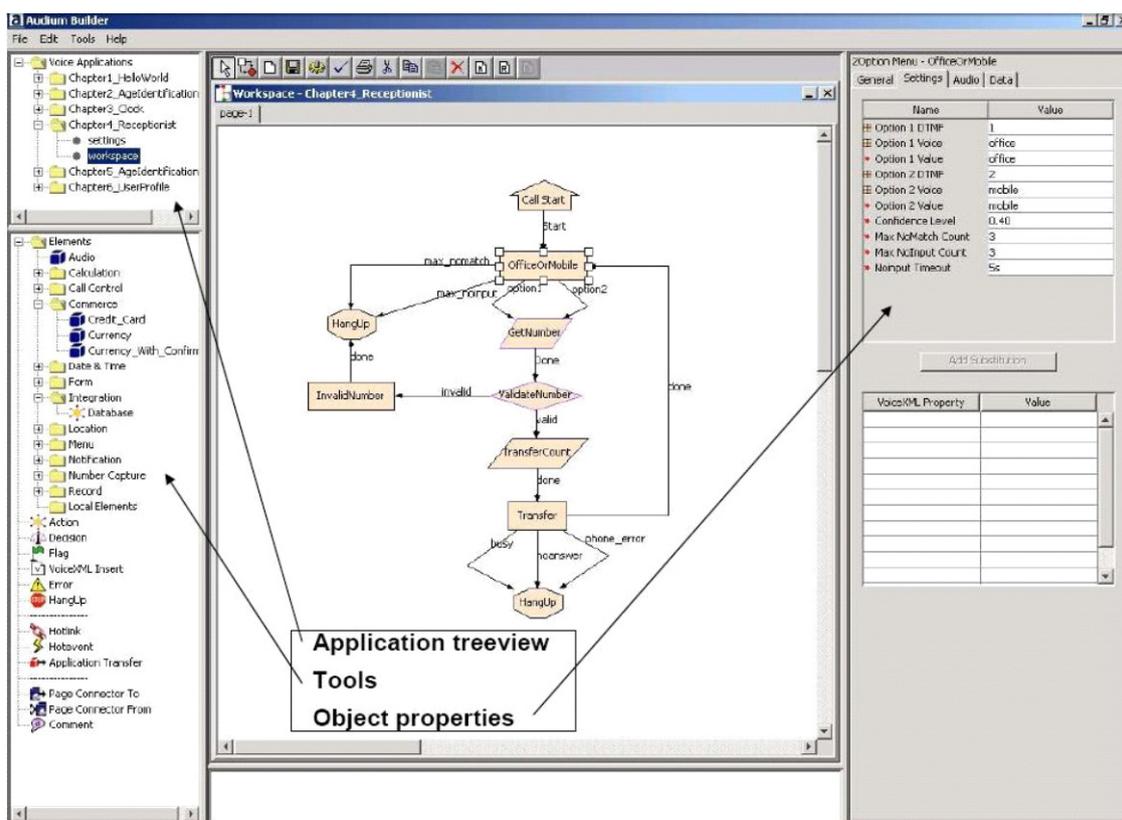


Figure B.1. Audium Builder main window. (Source: [Scholz, 2006])

Dialogue Designer by Avaya⁷¹: It is another IDE that includes several tools for designing and simulating speech-enabled services. The platform includes pre-built java classes and servlets for the dynamic generation of VoiceXML code and speech grammars, for the integration with backend databases, as well as support for the Java Telephone API (JTAPI) to allow the simulation and integration of telephony services.

In addition, Dialogue Designer allows the recording of high-quality speech messages, access to local and remote databases via the JDBC interface and SQL commands, and support for internet service integration using the Web Services Description Language (WDSL), Simple Object Access Protocol (SOAP), CCXML, and Remote Procedure Call (RPC) protocols. The platform also includes a speech recognition engine, and an embedded VoiceXML and CCXML browsers to allow the simulation and debugging of the final service. Figure B.2 shows the main components of the Dialogue Designer GUI, which allows full access to dialogue libraries and workflow, platform wizards, and component properties.

Finally, the platform includes an extensive library of predefined prompts in more than 20 languages, and allows the creation of TTS prompts supporting the SSML specification to dynamically modify the synthesis.

⁷¹ http://www.avaya.com/gcm/master-usa/en-us/products/offers/dialog_designer.htm#

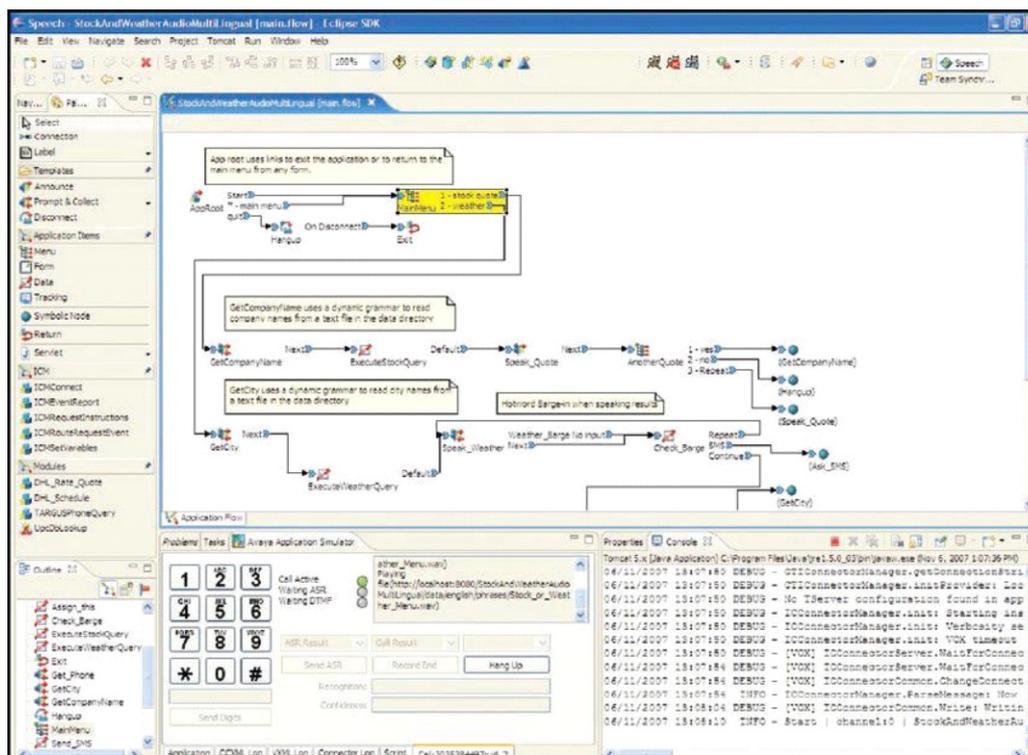


Figure B.2. Avaya Dialogue Designer. (Source: Avaya product brochure available at the corporate website)

Genesys Voice Platform⁷²: It is a stand-alone development platform for designing and testing VoiceXML-based applications. The platform includes all the necessary modules such as speech recognition, text-to-speech, and tools for backend, mainframes integration, and access to Web-based information. Genesys allows developers to deploy the service using the proprietary GenieHosting service or any Web server selected by the designer. The service can be tested using a SIP soft-phone or a Computer Telephony Integration (CTI) simulator that allows developers to transfer a call to an operator and to test the passing of arguments between dialogues and services (as we have also dealt in our platform successfully, see section 4.5.3, page 105).

The platform includes the following five software components: 1.) The Communications Server that acts as a media server interpreting and executing the VoiceXML commands, allowing the integration with the text-to-speech, ASR engines, and with the Genesys Voice Platform administration system. 2.) The Element Management Provisioning System that is a Web-based interface that allows the configuration and administration of the application and platform components. 3.) The Genesys Studio that allows the designer to develop the service using an intuitive Graphical User Interface. Then, the assistant automatically converts the graphical representation into the underlying code in VoiceXML without requiring the designer intervention as we do in our platform. 4.) The Voice Application Reporter that allows the designer to check user's interactions with the service, to save logging events, and to generate traffic and service reports using pre-defined templates. 5.) The Genesys Customer Interaction Management Platform Integration that allows, among

⁷² http://www.genesyslab.com/products/genesys_voice_platform.asp

others, the centralized management of the service, provides intelligent call queuing including personalized applications and music on hold, as well as different tools for making routing decisions.

One of the big advantages of the Genesys Voice Platform is that it supports the personalization of the applications through customized IVR voices and personalities, which can improve the quality of the service and reinforce the image of the company. For instance, the platform allows the possibility of automatically send recommendations, optional services, or other information of interest to the user while the system is running other time-consuming tasks.

Envox⁷³: Provides one of the most complete platforms for developing multimodal applications, supporting integration with IP, PSTN or mixed telephony environments. It includes four components: 1.) the Envoy Studio that provides the graphical interface for designing the service. 2.) The Communication Server that includes the VoiceXML gateway and browser, and acts as run-time platform. 3.) The Envoy Console that provides the graphical interface to administrate and control the service. 4.) And the Envoy Domain Server that ensures the continuous availability of the application. The platform may be integrated with different speech engines including Microsoft API and Nuance ASR, TTS, and speaker verification modules.

In addition, the platform supports several standard protocols that help to extend the VoiceXML specification, allowing new speech-based services and development functionalities. For instance, the platform allows the integration with conferencing, video messaging, email and fax services (such as receive, send, create or reply emails or faxes), extensive integration with backend systems, dynamic creation of Web pages, encryption, execution of external applications, and more. In relation with acceleration strategies, the platform includes advanced visual debugging tools (allowing variable simulation, breakpoints, step-by-step debugging, etc), hardware simulation, call logging, and a SQL wizard, which includes similar capabilities as the assistant described in section 4.3.2 (page 93), that allow the rapid definition of SQL statements suitable for designers with a reduced background on database languages and architectures.

Besides, the platform allows the rapid development of menus, forms, interaction with databases, and TTS messages. Like other development platforms, Envoy includes pre-built library components (such as dialogues, prompts and grammars) for requesting credit card numbers, currencies, dates, phone numbers, etc. Different assistants incorporated into the platform allow the designer to specify the properties of each component depending on the dialogue state; For instance, it is possible to specify the input mode (speech or DTMF), confidences, and the number of times the system confirms a slot, etc. Finally, another interesting feature is provided through the Nuance RealSpeak TTS system that support customs G2P (grapheme-to-phoneme) dictionaries, allowing the creation of new custom rules and entries in order to improve the quality of the pronunciation of special words (e.g. proper names).

OpenVXML Studio⁷⁴: It is an integrated design and management environment that simplifies the interaction of the platform with any VoiceXML compliant platform. The toolkit allows the development of DTMF and speech-enabled voice services. The graphical

⁷³ <http://www.envoy.com/>

⁷⁴ <http://www.eclipse.org/vtp/openvxml-announce/content/html/index.htm>

“drag-and-drop” interface allows the designer to create the dialogue flow as shown in Figure B.3. Key features included in this tool are built-in templates to support Web services and service oriented architectures, standard database templates for allowing an easy connection with the enterprise infrastructure, an editor for speech recognition grammars, support for common types of prompts (dates, currencies, ordinals, etc.), pre-recorded audio files, and support for user and language modelling through branding functionality.

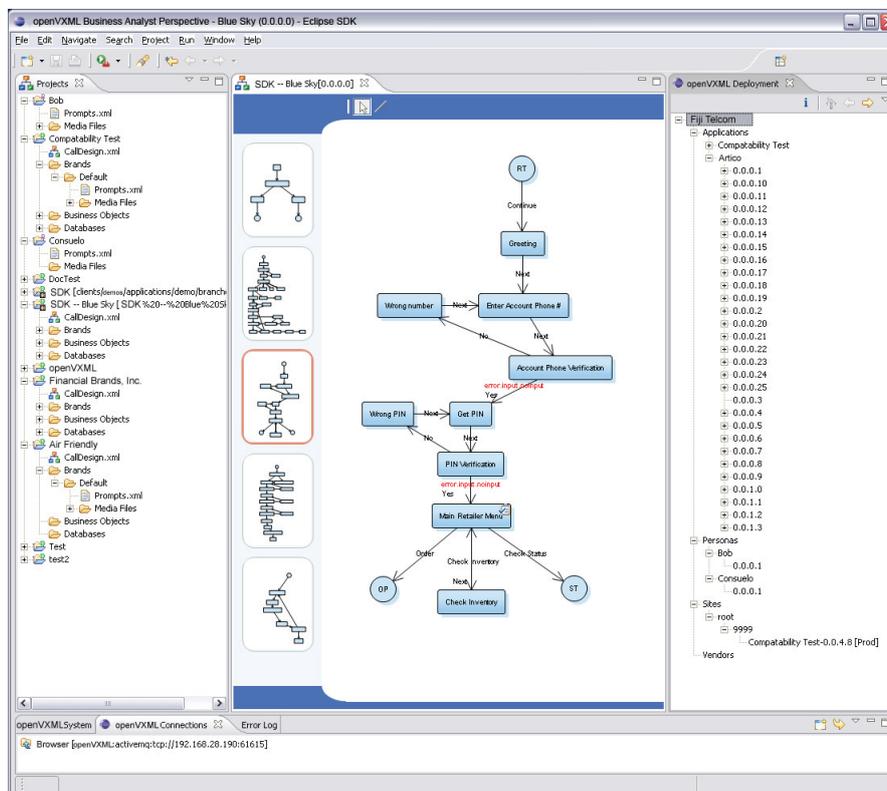


Figure B.3. Example of dialogue design using the OpenVXML toolkit. (Source: OpenVXML Web page)

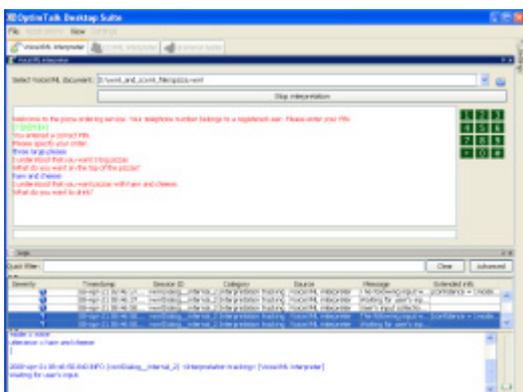
An interesting feature of this tool is that the design is made through configurable built-in modules, similar to some of the templates included in our development platform, which provide the main functionalities supported by the VoiceXML language. For instance, the platform includes the following modules: Fields, For Each, Play Prompts, Call Transfer, Comparisons, Menus, Database queries, Web services, etc. Each time a module is used in the application, the visual interface allows the designer to setup and configure it.

Unfortunately, the accelerations provided by the system are limited, since only default values are proposed when configuring the modules. Finally, during the design these modules are connected in order to define the dialogue flow. Although this approach allows a high level of fine-tuning, it could make difficult the creation of very complex services since the flow view could become confusing. The solution implemented by this platform is the creation of multiple parallel canvases and a special kind of connectors between canvases called wormholes. In our platform, we decided to use some kind of encapsulation of the actions defined in a dialogue in order to reduce the graphical representation, but allowing the designer to switch between a basic and a complex representation of the flow.

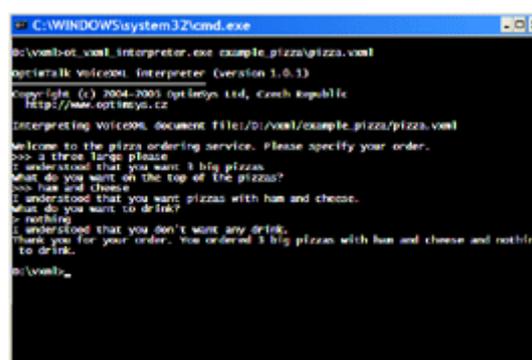
In addition, the platform incorporates the concept of Business Objects, which represent the fields of the database that the designer defines as necessary for the current service. This concept is similar to the classes and attributes we used in our platform (see section 3.2.2,

page 60). These Business Objects are later used in database queries allowing the designer to retrieve complex objects; then, each particular attribute of the object is matched to a local variable in the dialogue in order to provide the information retrieved from the database to the user. In our platform, we follow a similar approach but we went a step forward creating a semi-automatic procedure that proposes the matching and automatically creates the local variables (see section 4.5.3, page 105).

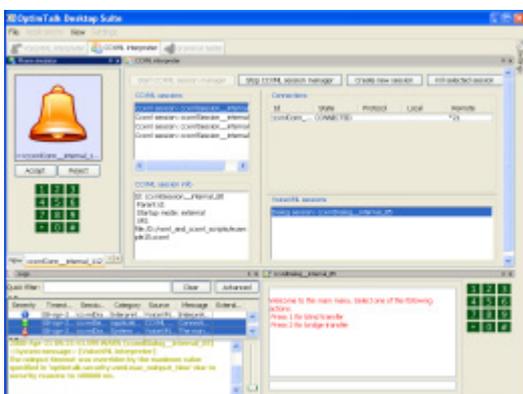
Finally, another interesting feature provided by this platform is the methodology used for the creation of system prompts. In this case, the assistant allows the designer to type in the words that make up the message, and complete it using dynamic values stored in local or global variables. The process is very similar to the one we propose in section 4.7.1.1 (page 116).



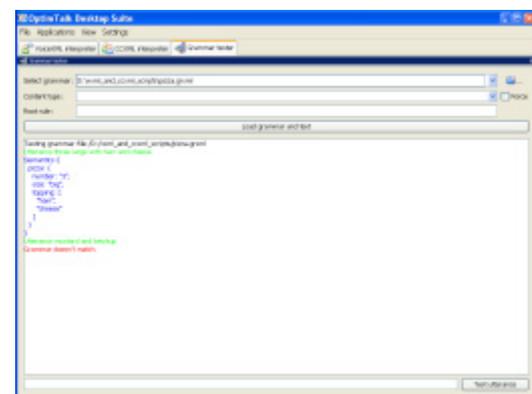
(a)



(b)



(c)



(d)

Figure B.4. Example of wizards included in the OptimTalk Professional Edition Toolkit. (Source: OptimTalk Desktop Suite Web page)

OptimTalk Desktop Suite⁷⁵: It is another development toolkit currently available in two versions: a free and a professional edition; both versions run on Linux and Windows, although some features may be disabled depending on the operating system. The professional edition includes a big number of features such as a GUI interpreter for VoiceXML and

⁷⁵ <http://www.optimtalk.cz/products/desktop-suite/introduction.php>

CCXML languages, support for TTS messages with SSML tags, speech recognition and speech synthesis using the Microsoft Speech API, simultaneous speech recognition and keyboard input, voice activity detector for recording, and barge-in. Besides, it includes a command line VoiceXML interpreter, logging of CCXML events, and a command line or GUI-based Grammar tester. The professional version offers support for Speech Recognition Grammar Specification (SRGS) and Semantic Interpretation for Speech Recognition (SISR), conferences, and more. Although the user interface is very simple, the platform can be interesting for novice programmers, companies with reduced budget, or for creating small business services.

Figure B.4. shows some examples of the wizards included in the professional edition. In (a) the designer can evaluate the speech recognition engine and the speech synthesizer for a given dialogue, as well as entering DTMF input using the telephone keyboard. In (b) the command line interface allows the designer to debug the application using speech from a microphone or typing in the utterance using the keyboard. The interface also provides information about recognised sentences and barge-in features, as well as useful feedback about the grammars and prompts used by the system. Using (c) the designer can set the CCXML session manager parameters, telephony hardware, and parameters for the VoiceXML interpreter used by the CCXML script, etc. Finally, the grammar tester, depicted in (d), allows the designer to type in utterances to be parsed using the current grammar in order to check if all the sentences can be correctly recognized or not. Besides, the assistant provides the semantic interpretation and other useful information about the utterances that do not match the grammar rules.

Vocalocity App Center⁷⁶: This platform includes a graphical user interface (see Figure B.5.) that simplifies the visualization and edition of VoiceXML applications, allows error checking, prompt recording and importing, as well as down-sampling of audio files for telephony compatibility. The GUI allows the creation of the dialogue flow through an object-model approach. In this approach, the flow is created using different objects, representing the different actions to perform the service, and connecting them through conditioned or direct transitions. Each object is defined by different configurable properties that control its behaviour. Besides, each object may have one or several outputs depending on its configuration for error handling or if there are different result outputs.

An interesting feature of the platform is that any basic design can be specified in four steps by only dragging and dropping three objects into the application canvas, connecting them sequentially, and setting some properties of the application. The first step, called Ask step, consists in the creation of an action for requesting information from the user. The second step, called Data step, represents the process of accessing and retrieving information from the backend database. The third step, called Tell step, corresponds to the action of providing the retrieved information to the user. Finally, during the fourth step, called Publish step, the designer creates the VoiceXML script and configures the platform in order to make the service available. Interestingly, in our platform we have developed a similar approach where most of the actions required to define a dialogue correspond to the first three steps described above (see section 4.5.2, page 103).

⁷⁶ <http://www.vocalocity.com/products/productdetail.cfm?productid=100007>

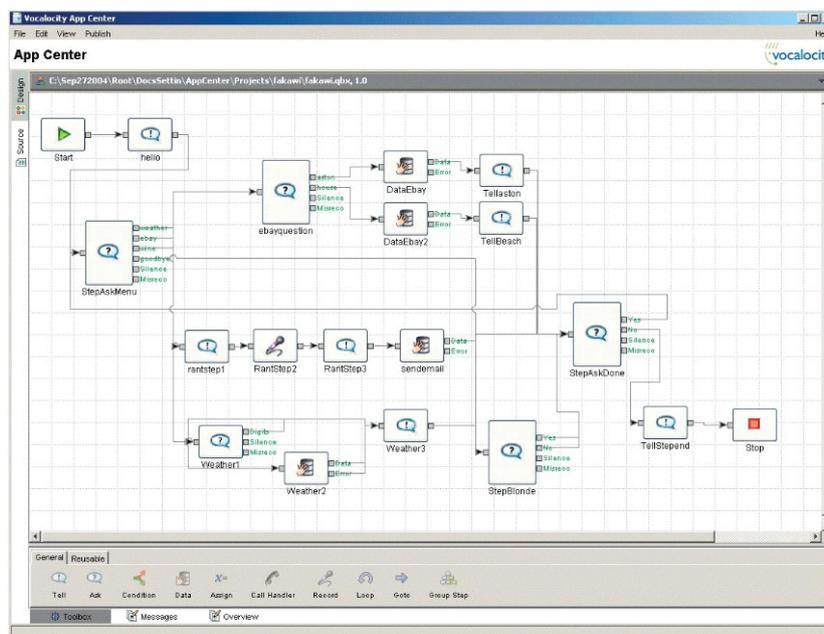


Figure B.5. Example of dialogue design using the Vocality App Center. (Source: Vocality Web page)

VoiceObjects Desktop⁷⁷: This is a free and downloadable platform that allows the design, development, debugging, deployment, and administration of multimodal applications (allowing among other modalities video, text, and speech) for voice-enabled and mobile Web-based services using VoiceXML. The platform is available in two editions: VoiceObjects Desktop that must be installed on the client machine, and VoiceObjects Desktop for Web that is a light Web-based version of the previous one. Both interfaces have the same core functionality and most of the project settings can be share between both editions.

A singular feature of this platform is that the dialogue flow is designed using a tree-based form-filling object modelling, in contrast to most of the dialogue platforms, including ours, that use state-based dialogue modelling. In this case, the dialogue flow is built using a hierarchical structure of objects and nested objects (see Figure B.6.). Objects are available through the GUI in a sidebar in the main window or using a quick-search window, which can be dragged-and-dropped into any place of the dialogue flow or even in some of the configuration windows of other objects. The workspace can be divided into different layers including system layers and user-built layers, contributing this way to simplify the visualization of the dialogue flow. The platform allows two operational modes: network and stand-alone. The former allows role-based team collaboration, and supports Concurrent Versions System (CVS) capabilities, and configurable audit trails. The latter is used for offline development and debugging.

In relation with accelerations and interesting features included in this platform, we can mention the Storyboard Manager, which is a special tool helps the designer to migrate from proprietary IVR systems to new VoiceXML-based platforms or to the VoiceObjects platform. Besides, the Storyboard Manager is also used to automatically export the list of the prompts

⁷⁷ <http://developers.voiceobjects.com/>

(audio, TTS, or video) used in the application, to create libraries of dialogues, and for importing VoiceXML code, among others. The platform also includes a phone simulator that can be used to test, debug, and check out VoiceXML files. Moreover, the phone simulator can be used to make demonstrations of voice and portable Web applications. Call tracing and logging/reporting are also supported, including information about recognition results, timestamps, and processing time. Other tools included in the platform allow the creation of prompt recordings and the automatic documentation of the project.

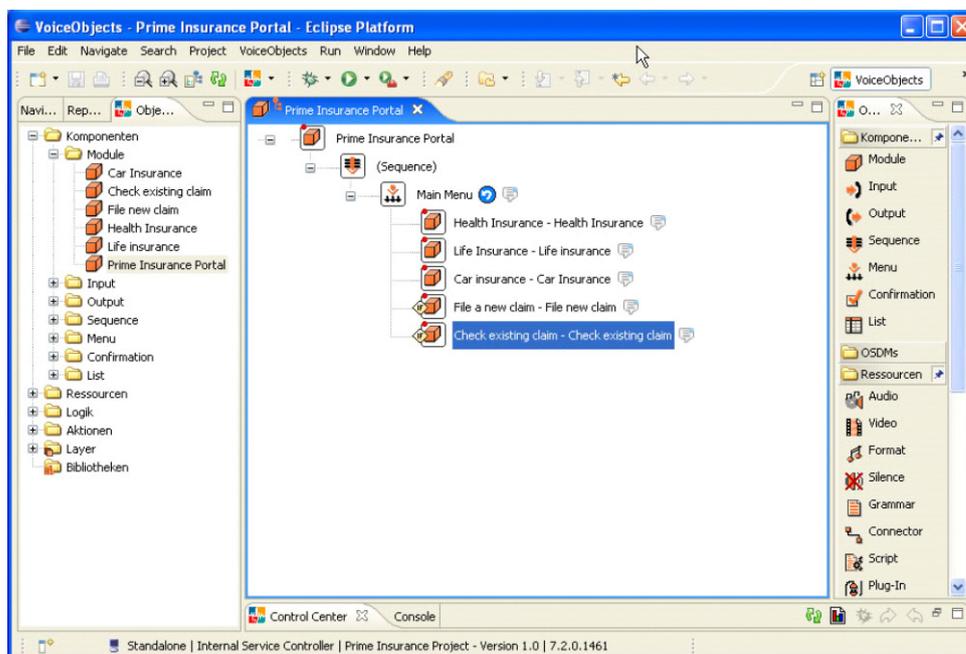


Figure B.6. Appearance of the main work area of the VoiceObjects Desktop. (Source: VoiceObjects Web page)

Finally, the platform incorporates a runtime version of the VoiceObjects Server that enables the deployment and management of the service, and includes a complete library of platform drivers to support different IVR and USSD (Unstructured Supplementary Service Data for GSM phones) platforms. The server includes a monitoring environment, provides a connection framework for integration with the backend system (allowing both server-side scripting and J2EE code execution), and offers support for rollbacks, application tracing, the creation and dynamic generation of video application for 3G phones, among others.

B.2 Web-Based Development Platforms

BeVocal Café⁷⁸: Recently acquired by Nuance, this site allows designers to use all the capabilities of Nuance modules and tools. The site offers a big number of VoiceXML sample applications that can be used as an initial point for the creation of basic and advanced services, including backend integration, dynamic VoiceXML applications using Apache, Perl

⁷⁸ <http://cafe.bevocal.com/>

or Visual Basic scripts, etc., and for learning about the main platform features. Designers can test the services using an international phone number or a VoIP number. Several Web-based tools are also available in order to develop, test, and publish the voice service. The most important ones are:

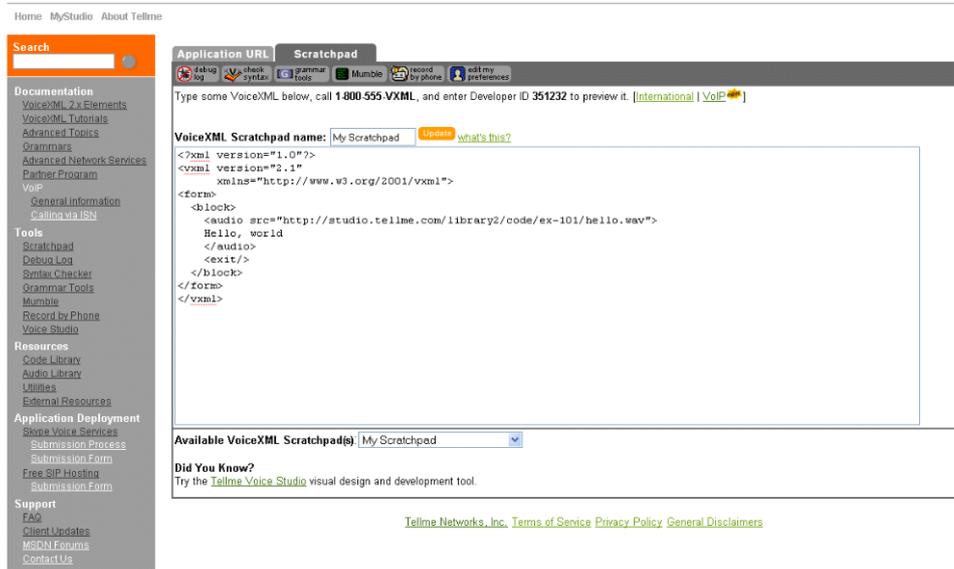
- **VoiceXML checker:** allows the designer to verify the syntactic correctness of the VoiceXML files used in the service.
- **Vocal Player and Log Browser:** The former allows replaying all calls made to the application and viewing their log entries; the latter displays information about errors, recognitions, Web access, variable traces, timestamps, call control, events, etc. These tools are useful for usability testing and for tuning speech recognition grammars since they provide feedback about confusing dialogues and in which dialogues the user spent more time.
- **Vocal Debugger and Vocal Scripter:** The former allows the designer to step through the VoiceXML code and view the state of VoiceXML variables during a call. The latter uses a text or "chat mode" channel to allow the designer to test the application flow in an interactive mode (i.e. the designer type in responses to VoiceXML text prompts in real time) or in batch mode (i.e. the designer uses a URL or text file containing inputs for running the VoiceXML service)
- **Grammar Compiler:** lets the designer to submit a grammar file and compile it offline in order to reduce overhead and significant delays during the execution of the service. The precompiled grammar can be referenced in the VoiceXML application using a key provided after the compilation.
- **Port Estimator:** is a statistical-based tool that provides an estimation of the number of telephony ports that the service will require in order to lose not users calls.

Tellme Studio⁷⁹: This site allows developing, maintaining, documentation, and testing speech-enabled services in two different ways: using the Web-based portal or an optional standalone application called Tellme Voice Studio.

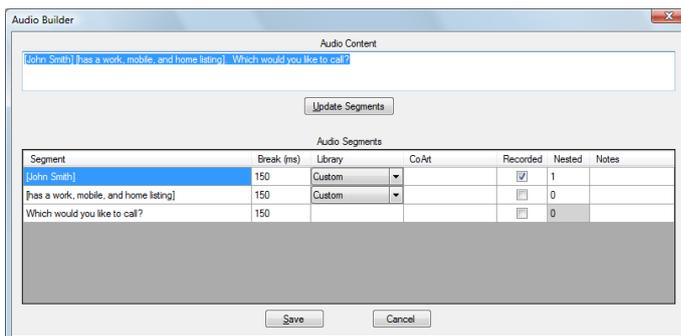
In relation to the Web site, it can be divided into two main sections: MyStudio and MyExtensions. MyStudio features different online tools including VoiceXML and Grammar scratchpads (see Figure B.7.a), which are used to write manually VoiceXML and grammar files respectively, and a syntax checker to validate the syntax of VoiceXML and grammar files. The VoiceXML terminal allows debugging the service using a text interface allowing the designer to interact with the service without making repeated calls, or using the speech interface. Moreover, the platform includes a grammar phrase checker and generator for testing speech grammars, as well as a DTMF generator for creating special grammars where each word is mapped to the letters of the English alphabet map that appears in the touch-tone keys on any standard telephone keypad. Besides, the designer can record prompts by phone and use the generated audio files in the VoiceXML application. In addition, several VoiceXML code and audio libraries examples, an audio conversion tool, and the possibility

⁷⁹ <http://studio.tellme.com/>

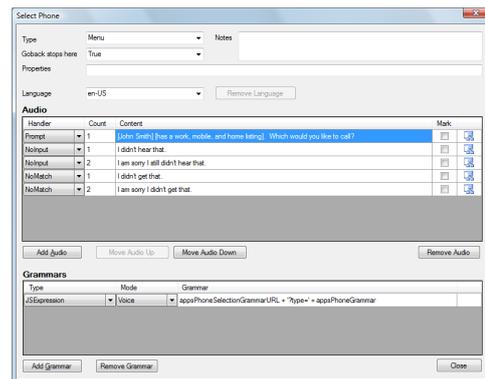
of testing the application using a toll-free number or free SIP calls through a VoIP telephone are also available.



(a)



(b)



(c)

Figure B.7. Appearance of different TellMe Studio tools. (Source: TellMe Studio Web site and TellMe Voice Studio user guide)

An interesting feature included in the platform, is the possibility of using an external Web application server to host some VoiceXML and grammars files of the service, instead of using the scratchpads to write, upload, and host all the files using the Tellme studio servers. This way some limitations of the Tellme platform for creating dynamic VoiceXML applications can be overcome. In any case, using the scratchpad or the Application URL, the system automatically validates the files every time the designer uploads the grammar/VoiceXML files or set the URL address.

On the other hand, MyExtensions allow designers to make demonstrations of the service to potential clients or to provide restricted access to groups of alpha/beta test users without providing them personal information as the Developer ID and PIN obtained after registering at the Tellme Web site. By default, both the ID and PIN are required to access the service through the phone, but they are also used to edit the service. This way it is possible to

analyze, document, and test the service under different conditions. Another available tool is Mumble; it lets the designer to test external procedures (API), generate documentation of the service through call flow diagrams, identify call flow coverage, obtain test metrics, and to specify a pool of different tests to check the behaviour of the service for different conditions such as nomatch, noinput, http errors, etc.

The Tellme Voice Studio can be downloaded from the Web site without any cost. The application is based on Microsoft's Visual Studio Domain Specific Language (DSL) toolkit, and, in a similar way as the Web-based environment, it can be used to design, maintain, develop, and document the voice application. The toolkit includes a graphical IDE with a toolbar of configurable modules (such as userinput, record, datafetch, transfer, subdialogs, presentation, decisions, etc) for creating the call flow specifying the states, transitions, and error handling procedures that make it up. The program includes tools to record audio prompts, to validate VoiceXML files, to create speech grammars, and for publishing the service.

Figure B.7.a shows the appearance of the Web-based interface of the scratchpad used to create and edit VoiceXML files. Figure B.7.b shows the assistant for creating prompts using static, concatenated or co-articulated audio files. Figure B.7.c shows the assistant for defining user input (menu or single slot), besides some error handling features (nomatch and noinput).

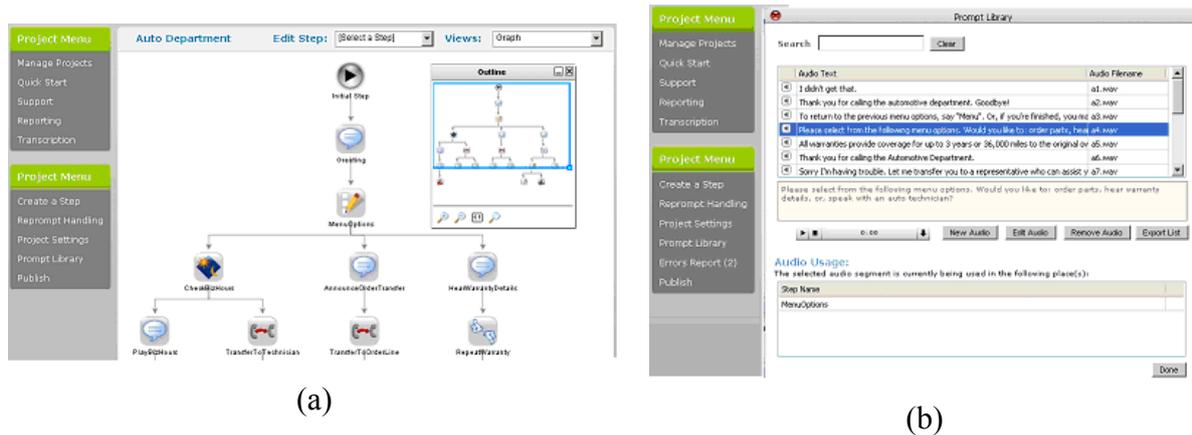
Voxeo Evolution⁸⁰: This Web portal provides several free development tools and advanced runtime modules to create a variety of speech and DTMF based services which can be called from or call out to any phone device. The site includes several free resources such as tutorials for VoiceXML, CCXML and CallXML languages, pre-recorded audio files in English for typical dialogues (e.g. names of airports, states, and airplane companies, months, numbers, weekdays, etc.), VoiceXML grammars, and sample applications. Besides, the site offers technical support 7 days/24 hours, a free direct-dial developer phone number avoiding advertisements or requesting awkward pin codes, this way accelerating the debugging of the service. Besides, the platform also offers the possibility of creating the service in several languages through the incorporation of different ASR and TTS engines from different vendors. The platform also features voice recognition, audio play and record, DTMF entry, and Voice over IP (VoIP) access.

In addition, the platform supports the CCXML standard protocol allowing call routing, call recording, visualization of logs (see Figure B.8.c), transfer, inbound and outbound calls, and conferencing capabilities (for 2 to 30 participants). Besides, it is possible to define different failover URLs and phone numbers allowing the Voxeo platform to ensure that any user's call will be processed or transferred in case the primary, secondary, or tertiary Web server are not responding or working properly.

The platform includes a Web-based application tool called Evolution Designer, also available as a PC-based standalone application, for developing the service. The Evolution Designer GUI (see Figure B.8.a) allows developers to create and edit the call flow through different wizards and configurable modules called "steps", which allow the creation of prompts, forms for multiple-choice questions, call recording, call transfers, time-based routing, data integration including support for Web-service access to external databases using PHP, Java/JSP, and SQLite, and to configure Voxeo speech recognition and synthesis engines, among others. The GUI allows the definition of custom rules to handle different user

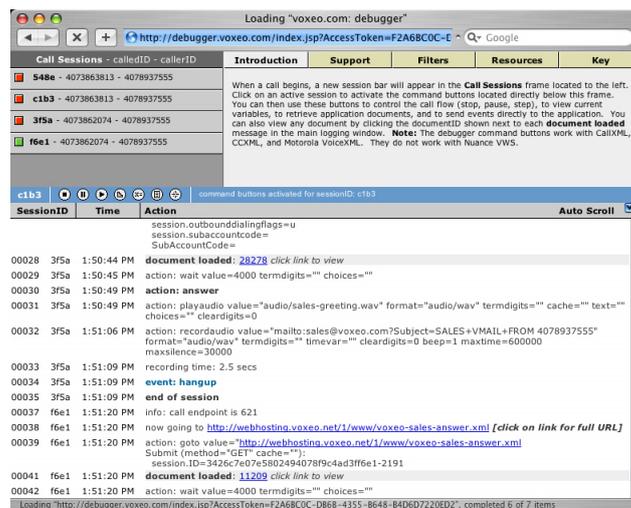
⁸⁰ <http://evolution.voxeo.com/>

and system errors such as re-prompting if the user does not say anything or the selected option is not available. The platform allows designers to track all the prompts used in the application (see Figure B.8.b) and the possibility of downloading the list of prompts as a spreadsheet file in order to record them later or to check them with the client. The Evolution Designer also supports the creation of query-based reports including information about calls and detailed steps used in the deployed application. These reports can be sent later in text, XML, HTML, or CSV (comma-separated values) format.



(a)

(b)



(c)

Figure B.8. Examples of Voxeo Evolution Web-based tools. (Source: Voxeo Studio Web page)

Finally, the site offers support for an open research meta-project called RocketSource⁸¹, which provide open source code to three different and common spoken dialogue applications: a speech-driven voicemail application, a phone and Web-based conference manager, and an auto attendant system to connect callers with any person or

⁸¹ <http://www.rocketsource.org/>

department in an enterprise. The main motivation of these applications is to provide a starting point for novice developers, as well as examples of use of advanced features included in the platform.

VoiceGenie⁸²: supported by Genesys, this site allows designers to host and access trial or demo versions of the final application using the GenieHosting service. The site offers, basically, the same services and features provided by the Voxeo Community. An important feature is that the site supports integration with several ASR and Text-To-Speech engines provided by different vendors such as Nuance, SpeechWorks, IBM, Speechify, AT&T, etc. The platform allows designers to select the language and gender of the TTS voice, as well as the ASR and TTS engines to be used when providing the service. Then, these settings are saved using different VoiceXML tags and attributes and applied when the service is loaded by the runtime platform. Besides, the service can be tested using two different phone numbers. Depending on the selected number, some properties of the platform will be available or not. The portal provides different tools for validating VoiceXML files, creation of grammar files, recording by phone of short prompts, call logs, and conversion of VoiceXML files among IVR platforms, etc.

Finally, the site also offers a free trial version of a PC-based application called GenieBuilder, which provides an intuitive graphical tool to create, test, and deliver the application. This software extends some of the functionalities offered by the Web-based site such as: a larger built-in library of reusable dialogue modules, support for collaborative and role-based development, built-in ASR and TTS engines, and support for JDBC, ODBC, Oracle, Microsoft SQL Server, DB2, and Informix database connections.

⁸² <http://developer.voicegenie.com/>

APPENDIX C. TEMPLATES FOR THE CREATION OF AUTOMATIC DIALOGUES IN THE MERA-SPEECH ASSISTANT

This appendix shows the templates used to generate, in the GDialogXML syntax, the dialogue flow for the confirmation handling and presentation of lists of results in the MERA-Speech assistant. In order to simplify the reading we use a pseudo code approximation. Although we have tried to make the pseudo code clear and independent of the GDialogXML syntax, the complete specification is available at the Gemini Web page⁸³ for further reading. The following terms appear throughout this appendix, and refer to specific terminology used in the templates described in this section.

- **StopFilling:** Tag used to stop the recognizer and finish the current dialogue.
- **DoFilling:** Tag used to repeat the query and load the current dialogue again.
- **isVarSet:** Tag to indicate a function that detects if a variable has been filled or not.
- **fConfidence:** Variable used to save the value of the confidence of a selected slot. The value is returned by the real-time function ConfidenceOfField.
- **Unset:** Tag used to indicate that the content of a variable is cleared, the variable is not destroyed.
- **sNextDialog:** Variable to save the name of the dialogue where to jump to when the system needs to return to a previous dialogue from inside of a non-returning dialogue.
- **sPreviousDialog:** Name of the last DGet dialogue called before the current one. We use this name in order to jump to this dialogue if there was a correction in an implicit confirmation procedure. The name is automatically detected.
- **xGenericFilling:** It makes reference to a template, created in the RMA (see sections 3.3.2 and 4.5.4, pages 63 and 106), that provides the behaviour when two or more slots need to be retrieved. Basically, the template attempts to fill in all the slots at the same time, but if this is not possible, or some of them have not been filled, the template attempts to fill in each slot one by one using automatic dialogues for each one.

C.1 Template for the Presentation of Lists of Objects

This is the template used for defining the internal actions and flow for the dialogues used for the presentation of lists of objects. The template considers four cases considering the number of items retrieved from the database: zero, one, in a range, and too many items. The process for filling this template also includes the creation of several automatic dialogues such

⁸³ <http://www-gth.die.upm.es/projects/gemini/>

as dialogues for providing complete or partial attributes for the object, for requesting the number of items to show, to ask the user the item he/she wants to obtain more info about, to inform the user there are no more items to show or that the database query does not return any result, etc.

```

//The dialogue returns, through the sNextDialog variable, the name of
the dialogue where to jump to
sNextDialog DSayOfList(Input_List)
{
//Variables' initialization
int iSizeList = sizeof(Input_List);

int iMax_Num_Items_DB = <xMaxDBResultsItems>;//The value is loaded in
the real-time system using this Call in a similar way as the
Confidence_Level (Remember that this value is User Level and Dialogue
dependent)

int iMax_Num_Items_Each_Time = <xMaxDBPresentableItems>;
int iMax_Num_Items = 0;//The number of items that the user wants to
listen to.
int InnerCounter = 0;//Counter for Case 3
int OuterCounter = 0;//Counter for Case 3
int iItem = 0;
String sMaxNumItems;
ObjRefr_Item_I; //It is an Object_Refr with the same type of the List

Unset(sNextDialog); //sNextDialog is a global variable defined in the
output file of the RMA
if(iSizeList == NULL || iSizeList == 0) //Case 1
{
    Call DSayNotificationNoItems; //It's an automatic dialogue, the
designer can change its name

    Unset(Slot1, Slot2, Slot3 ....);//Slots that must be unset in order
to repeat the flow

    sNextDialog = DFirstDialog or Another_Dialogue;//Name of the dialogue
to continue the flow or to repeat the query
}
else if (iSizeList == 1) //Case 2
{
    if(.TypeOf(Input_List) == Object)
    {
        Assign ObjRefr_Item_I = Input_List [0]; //We select the only item
of the list
        Call DSAY_ATTRS_FOR_LIST (ObjRefr_Item_I); //It's a configurable
DSay dialogue to show complete or partial attributes for the selected item
in the list
    }
    else
    {
        Call DSAY_LIST ( Input_List [0]);
    }
}
else if(1 < iSizeList <= iMax_Num_Items_DB ) //Case 3
{
    if(bSayNumberOfItems == true) //It's an option for the designer in
order to notify the user how many items exist in the List, and ask him/her
how many items does s/he wants to listen to. If the checkbox is selected
the following code is generated.
    {

```

```

    Call DSayNumberOfItemsOfList (iSizeList + 1); //Automatic DSay
dialogue to notify the number of items in the List
    sMaxNumItems = DGetObtainMaxNumItems(); //No. of items the user
wants to listen to
    iMax_Num_Items = GetIntFromRepr(sMaxNumItems) - 1;//Convert from
string to integer
    if(iSizeList < iMax_Num_Items)//If the value is greater than the
size of the list then change its value to the size of the list
        iMax_Num_Items = iSizeList;
    }
    else
    {
        iMax_Num_Items = iSizeList; //The user has to listen to all the
items in the list
    }

    while (OuterCounter < iMax_Num_Items )
    {
        for (InnerCounter = 0; InnerCounter < iMax_Num_Items_Each_Time
&& OuterCounter < iMax_Num_Items; InnerCounter++, OuterCounter ++)
        {
            if(.TypeOf(Input_List) == Object)
            {
                Call DSay_Basic_Info ( Input_List[OuterCounter], InnerCounter
+ 1);//It's an automatic dialogue to show info about the current item of
the list in the loop.
            }
            else
            {
                //It's an automatic dialogue to show information about the
current item of the list in the loop. We pass each attribute individually
                Call DSay_Basic_Info (Input_List[OuterCounter].Attr1,
List[OuterCounter].Attr2, ..., InnerCounter + 1);
            }
        }//for
        //Now we need to know if the user wants to continue listening
other in the list
        sAnswer = DGet_ItemIndex();

        if(sAnswer == Continue)
        {
            //Do one iteration more
        }
        //We need to change the counters and to perform a new iteration)
        else if(sAnswer == Repeat)
        {
            OuterCounter = OuterCounter - iMax_Num_Items_Each_Time;
        }
        else if(sAnswer == RepeatAll)
        {
            OuterCounter = 0; //Reset the counter and perform new
iterations
        }
        else if(sAnswer == Exit)
        {
            OuterCounter = iMax_Num_Items + 1;
        }
        //If the item of the list is not an object (it is only made up of
atomic types such as: Strings, Integer, Float, Boolean, etc.) we cannot
show more detailed info (previous info has been shown in DSayBasicInfo
dialogue) so we jump this step

```

```

if(.TypeOf(Input_List) != Object)
{
  //The user wants to obtain more information about one the
items.
  iItem = GetIntFromRepr(sAnswer) - 1;
  if(OuterCounter < iItem)
  {
    DSay_WrongItemIndex ();
    OuterCounter = OuterCounter - iMax_Num_Items_Each_Time;
  }
  else
  {
    if(bSayDetailedInfo == true)//It's a checkbox in the
assistant
    {
      Assign ObjRefr_Item_I = Input_List[iItem];
      Call DSay_FullInfo_OF_LIST(ObjRefr_Item_I);//It's the same
as for Case 2
      if(bCallDialog == true) //Optional step in the assistant
to jump to another dialogue
      {
        sNextDialog = DSelectedDialog;
      }
      OuterCounter = iMax_Num_Items + 1;//We exit the while and
jump to the next if condition (i.e., OuterCounter == iMax_Num_Items)
    }
  }
} //if
} //While

//If we arrive here is because the user has not selected any item,
so it is like case 1.The default values are the same for case 1, but the
designer can change them
if(OuterCounter == iMax_Num_Items)
{
  Call DSayNotificationNoItems;
  Unset (Slot1, Slot2, Slot3 ....);//The designer chooses the slots
that must be unset in order to repeat the flow
  If(NextDialog != "") //If the designer has selected a dialogue
where to jump..
  {
    sNextDialog = DFirstDialog or Another_Dialogue;
  }
}
}
else if(iSizeList > iMax_Num_Items) //Case 4
{
  Call DSayTooManyItems; //It's an automatic dialogue to notify the
user about the high number of returned items (The designer can change the
name of the dialogue)

  /*The designer chooses the slots that are relevant for the whole
query in order to check if they are filled or not.
  If all the relevant slots for the query are set, then we need unset
some of them in order to obtain new values to restrict the query */
  if(Slot_A is VarSet? && Slot_B is VarSet? && Slot_C is VarSet?,
....)
  {
    //If all slots are filled we unset some of them depending on
designer selection.
    unset(Slot_A, Slot_B, Slot_C,...)
  }
}

```

```

        //Jump to the dialogue that will fill them again (probably the
first one)
        sNextDialog = DSelectedDialogA; //It's a non-returning dialogue
    }
    else
    { // If all the relevant slots are not filled we can continue with
the dialogue flow to ask for the unfilled ones in order to generate a more
restrictive query.
        sNextDialog = DSelectedDialogB; //Dialogue to jump to in order to
fill the unfilled slots
    }
}
}
}

```

C.2 One Slot Confirmation

This is the template used to generate the flow for handling the confirmation of a single slot. In the code, Slot1 is the slot that the system has to confirm.

```

sNextDialog DGetConfirmationHandlingOneSlot (Slot1)
{
//Variable's declaration
float fConfidence = 0.0;
String sAnswerYesOrNo = "";
String sNextDialog = ""; //By default is empty
//In the xFilling section there must be the RecognizerCall
//Reaction
Unset(sNextDialog); //All the dialogues unset this global variable by
default
if(isVarSet (Slot1)) //The slot has been filled
{
    fConfidence = ConfidenceOfField(Slot1);
    //This confirmation is allowed only when the previous dialogue has
ImplicitConfirmation
    if(Slot1 EqualStrings "Correct") //Inspect if the recognition result
corresponds to the "Correct" command in case the system is using an
implicit confirmation prompt
    {
        if(fConfidence > implicit) //Go back to the previous dialogue to
ask the user again
        {
            Unset(Previous_Slot);
            Unset(Slot1);
            sNextDialog = Previous_Dialogue; //The Previous_Dialogue is
automatically set for the assistant.
            StopFilling;
        }
        Else //We are not sure about the "correct" command repeat again
the question
        {
            DSayNoMatch();
            Unset(Slot1);
            DoFilling;
        }
    }
}
if(0 <= fConfidence < explicit) //NoMatch
{
    DSayNoMatch(); //Repeat the question
    UnSet(Slot1);
}
}

```

```

        DoFilling; //Do one iteration more
    }
    else if(Explicit <= fConfidence < Implicit) //Explicit Confirmation
    {
        sAnswerYesOrNo = Call DGetYesOrNo(Slot1); //Ask for the current
slot in this dialogue
        if(sAnswerYesOrNo EqualStrings "No")
        {
            //The system asks again
            UnSet(Slot1);
            DoFilling;
        }
        else //The user answers "yes", so the recognition result is
correct
        {
            StopFilling;
        }
    }
    else if(Implicit <= fConfidence < none) // Implicit Confirmation
    {
        Call DSayImplicit_FOR_DGetConfirmationHandlingOneSlot(Slot1);
        StopFilling;
    }
    else if(none <= fConfidence <= 1.0) //Without confirmation
    {
        StopFilling;
    }
} //If IsVarSet
else ///The slot has not been filled, we need to fill it
{
    DoFilling;
}
}

```

C.3 Mixed-Initiative Confirmation

This is the template for the verification of two or more slots (Mixed Initiative). For simplicity, in the code the system only needs to confirm two slots, although the real template supports two or more slots. This dialogue does not have Implicit Confirmation.

```

Dialogue DGetConfirmationHandlingMixedInitiative (Slot1, Slot2)
{
    //Variable's declaration
    float fConfidence = 0.0;
    String sAnswerYesOrNo;
    String sNextDialog = ""; //By default is empty

    //In the xFilling section there must be the RecognizerCall

    //Reaction
    Unset(sNextDialog); //All the dialogues unset this global variable
by default
    if(isVarSet (Slot1) && isVarSet (Slot2)) //All the slots must be
filled . If not, we fill them one by one following the xGeneric template
    {

```

```

        fConfidence = ConfidenceOfField(Slot1); //We assume that the
recognizer returns one global confidence value for the recognition, not one
for each slot, so it is not important which slot we use here

        //This confirmation is allowed only when the previous DGet
dialogue has ImplicitConfirmation
        if(Slot1 EqualStrings "Correct") //Inspect if the recognition
result corresponds to the "Correct" command in case the system is using an
implicit confirmation prompt
        {
            if(fConfidence > implicit)
            {
                Unset(Previous_Slot);
                Unset(Slot1);
                Unset(Slot2);
                //The Previous_Dialogue is automatically set for the
assistant.
                sNextDialog = Previous_Dialogue;
                StopFilling;
            }
            else
            {
                DSayNoMatch();
                Unset(Slot1);
                Unset(Slot2);
                DoFilling;
            }
        }

        if(0 <= fConfidence < explicit) //NoMatch
        {
            DSayNoMatch();
            UnSet(Slot1);
            UnSet(Slot2);
            StopFilling; //Ask each slot individually using the xGeneric
Template
        } //If_NoMatch
        else if(Explicit <= fConfidence < none) //Explicit and Implicit:
We merge the Explicit and Implicit Confirmation
        {
            sAnswerYesOrNo = Call DGetYesOrNo_FOR_
DGetConfirmationHandlingMixedInitiative (Slot1, Slot2);
            if(sAnswerYesOrNo EqualStrings "No") //We try to confirm both
slots at the same time
            {
                //The system asks again
                Unset(Slot1);
                Unset(Slot2);
                //We stop the filling because in the xGenericFilling if the
slots are unset then we try to fill them one by one
                StopFilling;
            }
            else //The user answers "yes", so the recognition result is
correct
            {
                StopFilling;
            }
        }
        else if(none <= fConfidence <= 1.0) //Without confirmation
        {
            StopFilling;
        }

```

```

    }
    }//If IsVarSet
    else /Fill them one by one
    {
        Unset(Slot1);
        Unset(Slot2);
        //We stop the filling because in the xGenericFilling if the slots
        are unset then we try to fill them one by one
        StopFilling;
    }
}

```

C.4 One Slot Plus Over-Answering Confirmation

This is the template for the verification of one slot plus one slot for over-answering. This dialogue has implicit confirmation in only one case (Slot1 = Filled AND SlotOV = No_Filled).

Slot1	SlotOV	Action
Filled	Filled	Try to confirm both slots at the same time: We use NoMatch, Explicit and None
Filled	No_Filled	We try to confirm the compulsory and forget the OV: We use NoMatch, Explicit, Implicit and None
No_Filled	Don't Care	DoFilling

Table C.1. Proposed actions for the automatic filling of one slot and over-answering DGet dialogues in the MERA-Speech assistant

```

Dialogue DGetConfirmationHandlingOnePlusOV (Slot1, SlotOV)
{
    //Variable's declaration
    float fConfidence = 0.0;
    String sAnswerYesOrNo;
    String sNextDialog = ""; //By default is empty
    //In the xFilling section there must be the RecognizerCall
    //Reaction
    Unset(sNextDialog);
    if(isVarSet (Slot1) == true && isVarSet (SlotOV) == true)//All the
    slots must be confirmed at the same time
    {
        fConfidence = ConfidenceOfField(Slot1); //We assume that the
        recognizer returns one global confidence value for the recognition, not one
        for each slot, so it is not important which slot we use here

        //This confirmation is allowed only when the previous DGet dialogue
        has ImplicitConfirmation
        if(Slot1 EqualStrings "Correct")//Inspect if the recognition result
        corresponds to the "Correct" command in case the system is using an
        implicit confirmation prompt
        {
            if(fConfidence > implicit)

```

```

    {
        Unset(Previous_Slot);
        Unset(Slot1);
        Unset(SlotOV);
        sNextDialog = Previous_Dialogue;
        StopFilling;
    }
else
    {
        DSayNoMatch();
        Unset(Slot1);
        Unset(SlotOV);
        DoFilling;
    }
}
if(0 <= fConfidence < explicit) //NoMatch
{
    DSayNoMatch();
    UnSet(SlotOV);
    UnSet(Slot1);
    DoFilling;//Repeat the question, Do one iteration more

}
//If_NoMatch
else if(Explicit <= fConfidence < none)//Explicit and Implicit: We
merge the Explicit and Implicit Confirmation
{
    //Automatically generated dialogue in the MERA-Speech, the name
can be changed by the designer
    sAnswerYesOrNo = Call DGetYesOrNo_FOR_
DGetConfirmationHandlingOnePlusOV (Slot1, SlotOV);

    if(sAnswerYesOrNo EqualStrings "No")
    {
        //The system asks again
        UnSet(Slot1);
        UnSet(SlotOV);
        DoFilling;
    }
    else//The user answers "yes"
    {
        StopFilling;
    }
}
//If_ExplicitConfirmation
else if(none <= fConfidence <= 1.0) //No confirmation
{
    StopFilling;
}
//If_IsVarSet
else if(isVarSet(Slot1) == true && IsVarSet(SlotOV) == false)//We try
to confirm the compulsory slot only and forget the OV
{
    fConfidence = ConfidenceOfField(Slot1); //We assume that the
recognizer returns one global confidence value for the recognition, not one
for each slot, so it is not important which slot we use here

    //This confirmation is allowed only when the previous DGet dialogue
has ImplicitConfirmation
    if(Slot1 == "Correct")//Inspect if the recognition result
corresponds to the "Correct" command in case the system is using an
implicit confirmation prompt
    {

```

```

    if(fConfidence > implicit)
    {
        Unset(Previous_Slot);
        Unset(Slot1);
        sNextDialog = Previous_Dialogue;
        StopFilling;
    }
    else
    {
        DSayNoMatch();
        Unset(Slot1);
        DoFilling;
    }
}

if(0 <= fConfidence < explicit) //NoMatch
{
    DSayNoMatch(); //Pass the counter if there are different prompts
    UnSet(Slot1);
    DoFilling;//Repeat the question, Do one iteration more for the
loop
}
//If NoMatch
else if(Explicit <= fConfidence < Implicit)//Explicit Confirmation
{
    sAnswerYesOrNo = Call DGetYesOrNo_FOR_
DGetConfirmationHandlingOneSlot (Slot1);

    if(sAnswerYesOrNo EqualStrings "No")
    {
        //The system asks again
        UnSet(Slot1);
        DoFilling;
    }
    else//The user answers "yes"
    {
        StopFilling;
    }
}
else if(Implicit <= fConfidence < none) //Implicit Confirmation
{
    Call DSayImplicit_FOR_ DGetConfirmationHandlingOneSlot (Slot1);
    StopFilling;
}
else if(none <= fConfidence <= 1.0)//Without confirmation
{
    StopFilling;
}
}
else if(IsVarSet(Slot1) == false) //All the OV slots are unset
{
    UnSetVar(SlotOV);
    DoFilling;
}
else//Any other combination is unset
{
    UnSet(Slot1);
    UnSet(SlotOV);
    DoFilling;
}
}

```

C.5 Mixed-Initiative Plus Over-Answering Confirmation

This is the template for the verification of two or more slots (Mixed-initiative) plus one or more slots for OV. This dialogue does not allow implicit Confirmation in any case. For simplicity, we use only two slots for Mixed-initiative and one slot for Overanswering.

Slot1 And Slot2	SlotOV	Action
Filled	Filled	Try to confirm all at the same time: We use NoMatch, Explicit and None
Filled	No_Filled	We try to confirm the compulsory slots and forget the OV slot; We use NoMatch, Explicit and None
No_Filled	Don't Care	StopFilling, we use the xGenericFilling

Table C.2. Proposed actions for the automatic filling of mixed-initiative and over-answering DGet dialogues in the MERA-Speech assistant

```

Dialogue DGetConfirmationHandling_MI_OV(Slot1, Slot2, Slot_OV)
{
  //Variable's declaration
  float fConfidence = 0.0;
  String sAnswerYesOrNo;
  String sNextDialog = ""; //By default is empty
  //In the xFilling section there must be the RecognizerCall

  //Reaction

  Unset(sNextDialog);
  if(isVarSet (Slot1) == true && isVarSet (Slot2) == true && isVarSet
(SlotOV) == true)//All the slots must be confirmed at the same time
  {
    fConfidence = ConfidenceOfField(Slot1); //We assume that the
recognizer returns one global confidence value for the recognition, not one
for each slot, so it is not important which slot we use here

    //This confirmation is allowed only when the previous DGet dialogue
has ImplicitConfirmation
    if(Slot1 EqualStrings "Correct")//Inspect if the recognition result
corresponds to the "Correct" command in case the system is using an
implicit confirmation prompt
    {
      if(fConfidence > implicit)
      {
        Unset(Previous_Slot);
        Unset(Slot1);
        Unset(Slot2);
        Unset(Slot_OV);
        sNextDialog = Previous_Dialogue; //The Previous_Dialogue is
automatically set for the assistant.
        StopFilling;
      }
      else
      {
        DSayNoMatch();
        Unset(Slot1);
        Unset(Slot2);
      }
    }
  }
}

```

```

        Unset(Slot_OV);
        DoFilling;
    }
}

if(0 <= fConfidence < explicit) //NoMatch
{
    DSayNoMatch();
    UnSet(SlotOV);
    UnSet(Slot1);
    UnSet(Slot2);
    StopFilling;
} //If NoMatch
else if(Explicit <= fConfidence < none) //Explicit and Implicit: We
merge the Explicit and Implicit Confirmation
{
    sAnswerYesOrNo = Call
DGetYesOrNo_FOR_DGetConfirmationHandling_MI_OV (Slot1, Slot2, SlotOV);
    if(sAnswerYesOrNo EqualStrings "No")
    {
        //The system asks again using the xGeneric Template
        UnSet(Slot1);
        UnSet(Slot2);
        UnSet(SlotOV);
        StopFilling;
    }
    else //The user answers "yes"
    {
        StopFilling;
    }
} //If ExplicitConfirmation
else if(none <= fConfidence <= 1.0) //No confirmation
{
    StopFilling;
}
} //If IsVarSet
else if(isVarSet(Slot1) == true AND isVarSet(Slot1) == true AND
IsVarSet(SlotOV) == false) //We try to confirm the compulsory slots only
and forget the OV
{
    fConfidence = ConfidenceOfField(Slot1); //We assume that the
recognizer returns one global confidence value for the recognition, not one
for each slot, so it is not important which slot we use here

    //This confirmation is allowed only when the previous DGet dialogue
has ImplicitConfirmation
    if(Slot1 == "Correct") //Inspect if the recognition result
corresponds to the "Correct" command in case the system is using an
implicit confirmation prompt
    {
        if(fConfidence > implicit)
        {
            Unset(Previous_Slot);
            Unset(Slot1);
            Unset(Slot2);
            sNextDialog = Previous_Dialogue; //The Previous_Dialogue is
automatically set for the assistant.
            StopFilling;
        }
    }
    else

```

```

    {
        DSayNoMatch();
        UnSet(Slot1);
        UnSet(Slot2);
        DoFilling;
    }
}

if(0 <= fConfidence < explicit) //NoMatch
{
    DSayNoMatch(); //We pass it a counter to allow different prompts
    UnSet(Slot1);
    UnSet(Slot2);
    StopFilling;//Repeat the question, Do one iteration more for the
loop
} //If NoMatch
else if(Explicit <= fConfidence < None)//Explicit Confirmation
{
    sAnswerYesOrNo = Call DGetYesOrNo_FOR_
DGetConfirmationHandlingMixedInitiative (Slot1, Slot2);

    if(sAnswerYesOrNo EqualStrings "No")
    {
        //The system asks again
        UnSet(Slot1);
        UnSet(Slot2);
        StopFilling;
    }
    else //The user answers "yes"
    {
        StopFilling;
    }
}
else if(none <= fConfidence <= 1.0)//Without confirmation
{
    StopFilling;
}
}
else
{
    UnSet(Slot1);
    UnSet(Slot2);
    UnSet(SlotOV);
    StopFilling;
}
}

```

C.6 Simple Confirmation and Basic Dialogues

This is the template for the verification of basic dialogues e.g., DGetYesOrNo. This dialogue does not have implicit confirmation. The arguments of the dialogue are only useful for the prompt of the dialogue.

```

Dialogue DGetBasicOrSimple (Compulsory_Slot, Slots_OV)
{
//LocalVars
float fConfidence = 0.0;
String sReturningConcept = "";

```

Speed Up Strategies for the Creation of Multimodal and Multilingual Dialogue Systems

```
//If the dialogue does not exist in the output file of the RMA, we
need to specify the InputFieldVar, ReturningVars
//InputFieldVar
sReturningConcept;
//ReturningVar
sReturningConcept;

//Reaction
if(isVarSet(sReturningConcept) == true)
{
    Unset(sNextDialog);

    //If the dialogue is new in the output file of the MERA-Speech
    fConfidence = ConfidenceOfField(sReturningConcept);
    //Else, it exists in the output file of the RMA
    fConfidence = ConfidenceOfField(Compulsory_Slot);

    if (Implicit <= fConfidence <= 1.0)
        StopFilling;
    else
        DoFilling;
}
else
    DoFilling
}
```

APPENDIX D. QUESTIONNAIRE FOR EVALUATING THE APPLICATION GENERATION PLATFORM

Age: _____ yrs.

Experience on dialogue development: _____ yrs

Developer Status: Novice Intermediate Expert

Mother Tongue: German Greek Spanish

D.1 Specific Questions by Assistant

D.1.1 Questions regarding the *assistant*:

1. How quickly did you learn to use the *assistant*?

	Not fast at all									very fast
1	2	3	4	5	6	7	8	9	10	

Comment: _____

2. Is the *assistant* easy and intuitive to use? Do you know what to do at each step?

	Not easy at all									very easy
1	2	3	4	5	6	7	8	9	10	

Comment: _____

3. Is the functionality sufficient?

	Not at all									all of them
1	2	3	4	5	6	7	8	9	10	

Comment: _____

- How do you rate the appearance of the *assistant* (consistent, transparent, and intuitive)?

Very poor **very good**

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Comment: _____

D.2 General Questions about the AGP

D.2.1 Advantages of using the AGP

- The provision of data modelling and connecting to external data sources:

not useful at all **very useful**

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

- The provision of application state flow modelling:

not useful at all **very useful**

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

- Easy adaptability to other languages

not useful at all **very useful**

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

- Easy adaptability to other modalities

not useful at all **very useful**

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

- Ready-made error-handling (nomatch, noinput)

not useful at all **very useful**

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

6. Speed up of development time as compared to writing VoiceXML/XHTML code by hand

not useful at all

very useful

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

7. Provision of user modelling

not useful at all

very useful

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

8. Provision of mixed-initiative dialogue handling

not useful at all

very useful

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

9. Provision of list handling [if applicable]

not useful at all

very useful

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

10. Provision of over-answering [if applicable]

not useful at all

very useful

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

11. Provision of easy connection to run-time modules (i.e. speaker recognition, language recognition):

not useful at all

very useful

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

D.2.2 Do you learn quickly how to make applications with the AGP?

yes no

Comment: _____

D.2.3 How do you rate the overall appearance of the AGP (consistent, transparent, and intuitive)?

Very poor

very good

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Comment: _____

D.2.4 Do you find the various assistants of the AGP are well integrated?

yes no

Comment: _____

D.2.5 Do you think non-experts could use the AGP efficiently?

yes no

Comment: _____

D.2.6 Would you use this system in the future or recommend it to develop speech/Web applications?

For any decision, please give a few reasons or name conditions if there are any.

yes no

Comment: _____

D.2.7 If yes, how much would you be willing to pay for its use?

Comment: _____

Thank you very much for your participation!

APPENDIX E. DETAILED RESULTS OF THE OBJECTIVE EVALUATION OF THE PLATFORM

Table E.1. Quantitative measures obtained during the objective evaluation

Task	Av. Novices			Av. Intermediates			Av. Experts			All Average		
	AGP	Diagen	Impr.	AGP	Diagen	Impr.	AGP	Diagen	Impr.	AGP	Diagen	Impr.
DMA - Create DB-based Class												
Elapsed Time (in seconds)	55	122	54.9%	47	93	48.9%	22	85	74.0%	45	104	56.6%
No. of Clicks	12	20	37.2%	11	9	-21.4%	8	17	55.9%	11	16	30.0%
No. of Keystrokes	9	116	92.6%	7	117	93.8%	7	97	92.8%	8	112	93.1%
No. of Keystroke Errors	0	5	100%	0	2	100.0%	0	2	100.0%	0	3	100%
Average Improvement			69.8%			55.3%			80.7%			69.0%
DMA - Create Mixed Class												
Elapsed Time (in seconds)	67	157	57.5%	70	118	40.6%	52	125	58.2%	65	137	52.8%
No. of Clicks	13	27	50.5%	16	14	-14.3%	14	25	42.9%	14	22	34.8%
No. of Keystrokes	8	152	94.6%	17	147	88.7%	15	158	90.8%	12	152	91.8%
No. of Keystroke Errors	3	5	38.9%	1	3	77.8%	0	7	100.0%	1	5	68.3%
Average Improvement			60.4%			48.2%			73.0%			61.9%
DCMA - Create Database Function												
Elapsed Time (in seconds)	181	180	-0.8%	92	139	34.1%	65	137	52.7%	125	156	19.9%
No. of Clicks	23	26	13.3%	19	11	-70.6%	14	19	24.3%	20	20	0%
No. of Keystrokes	73	237	69.4%	51	211	76.0%	63	227	72.2%	63	226	72.1%
No. of Keystroke Errors	16	10	-71.1%	2	5	50.0%	2	4	62.5%	8	7	-25.0%
Average Improvement			2.7%			22.4%			52.9%			16.6%

Task	Av. Novices			Av. Intermediates			Av. Experts			All Average		
	AGP	Diagen	Impr.	AGP	Diagen	Impr.	AGP	Diagen	Impr.	AGP	Diagen	Impr.
SFMA – Create One Slot State												
Elapsed Time (in seconds)	36	92	61.4%	35	68	48.0%	24	56	56.8%	33	76	56.7%
No. of Clicks	12	13	9.4%	11	11	0.0%	6	12	50.0%	10	12	15.3%
No. of Keystrokes	25	80	69.5%	28	84	66.5%	26	71	64.1%	26	79	67.4%
No. of Keystroke Errors	1	2	50.0%	2	3	44.4%	1	1	50.0%	1	2	50 %
Average Improvement			47.6%			39.8%			55.2%			46.6%
SFMA - Create State With Mixed-Initiative Slots + Transition												
Elapsed Time (in seconds)	73	128	42.7%	54	91	40.9%	33	79	58.9%	58	105	44.9%
No. of Clicks	21	23	6.6%	16	16	4.1%	16	21	23.8%	18	20	9.9%
No. of Keystrokes	27	81	66.1%	26	88	70.9%	25	77	67.3%	26	82	68.1%
No. of Keystroke Errors	2	3	10.0%	1	1	33.3%	0	4	100.0%	1	2	50.0%
Average Improvement			31.4%			37.3%			62.5%			42%
SFMA - Create Connection Between States												
Elapsed Time (in seconds)	7	87	91.6%	15	50	70.9%	8	44	81.8%	10	65	84.8%
No. of Clicks	6	3	-83.3%	9	4	-136.4%	6	1	-400.0%	7	3	-132.0%
No. of Keystrokes	0	36	100.0%	0	21	100.0%	0	15	100.0%	0	26	100.0%
No. of Keystroke Errors	0	7	100.0%	0	1	100.0%	0	0	0.0%	0	3	100.0%
Average Improvement			52.1%			33.6%			-54.5%			38.2%
RMA - Create A Menu Dialogue												
Elapsed Time (in seconds)	100	759	86.8%	61	618	90.2%	55	470	88.3%	77	647	88.1%
No. of Clicks	20	44	55.1%	18	35	49.5%	18	33	44.6%	19	38	51.4%
No. of Keystrokes	47	344	86.4%	49	402	87.7%	48	454	89.4%	48	388	87.6%
No. Of Keystroke Errors	1	95	99.5%	1	51	97.4%	0	14	100.0%	1	62	98.9%
Average Improvement			82.0%			81.2%			80.6%			81.5%

Task	Av. Novices			Av. Intermediates			Av. Experts			All Average		
	AGP	Diagen	Impr.	AGP	Diagen	Impr.	AGP	Diagen	Impr.	AGP	Diagen	Impr.
RMA - Create a Dialogue With OVR + Condition												
Elapsed Time (in seconds)	187	1,763	89.4%	107	1,421	92.5%	99	1,060	90.7%	140	1,493	90.6%
No. of Clicks	35	92	62.0%	28	83	66.0%	31	80	61.9%	32	86	63.3%
No. of Keystrokes	19	918	97.9%	18	1,079	98.4%	15	1,040	98.6%	18	998	98.2%
No. of Keystroke Errors	0	164	99.8%	0	56	100.0%	0	35	100.0%	0	100	100 %
Average Improvement			87.3%			89.2%			87.8%			88.0%
RMA - Create a Dialogue With Mixed-Initiative + Local Variable												
Elapsed Time (in seconds)	126			69			68			94		
No. of Clicks	23			21			21			22		
No. of Keystrokes	8			6			8			7		
No. of Keystroke Errors	1			0			2			1		
MERA-Speech – Create a Dialogue To Present A List Of Objects												
Elapsed Time (in seconds)	109			89			52			89		
No. of Clicks	20			15			16			17		
No. of Keystrokes	0			0			0			0		
No. of Keystroke Errors	0			0			0			0		
MERA-Speech - Fill In All DGet Dialogues												
Elapsed Time (in seconds)	4			4			6			4		
No. of Clicks	3			2			3			3		
No. of Keystrokes	0			0			0			0		
No. of Keystroke Errors	0			0			0			0		

Table E.2. Results for general questions about the assistants in the AGP during the subjective evaluation

General Questions About Each Assistant	Average Novice	Average Intermediate	Average Expert	Average All
How quickly did you learn to use the DMA? (Very Slow – Very Fast)	7.8	8.0	9	8.1
How intuitive do you think is the DMA? (No Intuitive at all – Very Intuitive)	8.0	8.7	9.5	8.6
Do you think that the functionality of the DMA is sufficient? (No at all – Totally)	8.8	7.7	8.5	8.3
How do you rate the appearance of the DMA (consistent, transparent)? (Very Poor – Very Good)	8.0	8.7	8	8.2
Average Score DMA	8.1	8.3	8.8	8.3
How quickly did you learn to use the DCMA? (Very Slow – Very Fast)	7.8	8.3	9	8.2
How intuitive do you think is the DCMA? (No Intuitive at all – Very Intuitive)	7.3	8.7	9.5	8.2
Do you think that the functionality of the DCMA is sufficient? (No at all – Totally)	7.8	8.7	9.5	8.4
How do you rate the appearance of the DCMA (consistent, transparent)? (Very Poor – Very Good)	7.3	8.7	9	8.1
Average Score DCMA	7.5	8.6	9.3	8.3
How quickly did you learn to use the SFMA? (Very Slow – Very Fast)	8.8	8.7	9	8.8
How intuitive do you think is the SFMA? (No Intuitive at all – Very Intuitive)	9.3	8.7	9	9.0
Do you think that the functionality of the SFMA is sufficient? (No at all – Totally)	9.5	8.3	10	9.2
How do you rate the appearance of the SFMA (consistent, transparent)? (Very Poor – Very Good)	9.3	8.7	9	9.0
Average Score SFMA	9.2	8.6	9.3	9.0

General Questions About Each Assistant	Average Novice	Average Intermediate	Average Expert	Average All
How quickly did you learn to use the RMA? (Very Slow – Very Fast)	8.3	7.3	8.5	8.0
How intuitive do you think is the RMA? (No Intuitive at all – Very Intuitive)	9.3	8.0	9	8.8
Do you think that the functionality of the RMA is sufficient? (No at all – Totally)	9.5	8.0	9.5	9.0
How do you rate the appearance of the RMA (consistent, transparent)? (Very Poor – Very Good)	9.0	8.3	9	8.8
Average Score RMA	9.0	7.9	9.0	8.6
How quickly did you learn to use the MERA-Speech? (Very Slow – Very Fast)	9.3	9.3	10	9.4
How intuitive do you think is the MERA-Speech? (No Intuitive at all – Very Intuitive)	8.5	9.0	10	9.0
Do you think that the functionality of the MERA-Speech is sufficient? (No at all – Totally)	8.8	9.0	9.5	9.0
How do you rate the appearance of the MERA-Speech (consistent, transparent)? (Very Poor – Very Good)	8.8	8.3	8.5	8.6
Average Score MERA-Speech	8.8	8.9	9.5	9.0
How quickly did you learn to use the Diagen? (Very Slow – Very Fast)	5.5	4.3	7.5	5.6
How intuitive do you think is the Diagen? (No Intuitive at all – Very Intuitive)	4.0	4.7	5.5	4.6
Do you think that the functionality of the Diagen is sufficient? (No at all – Totally)	3.3	5.3	3.5	4.0
How do you rate the appearance of the Diagen (consistent, transparent)? (Very Poor – Very Good)	3.0	5.3	4	4.0
Average Score Diagen	3.9	4.9	5.1	4.5