Project ref. no.	IST-2000-25426
Project acronym	*VICO*
Project full title	**V**irtual **I**ntelligent **Co**-Driver

Security (distribution level)	Project internal
Contractual date of delivery	June 2002 (M 15)
Actual date of delivery	8.11.2002
Deliverable number	D11
Deliverable name	Progress Report on the Natural Language Understanding, Dialogue Management, Response Generation, and Speech Synthesis Components
Type	Report
Status & version	Final
Number of pages	76
WP contributing to the deliverable	WPs 5.1, 5.2, 5.3
WP / Task responsible	NISLab, University of Southern Denmark
Other contributors	DaimlerChrysler AG (DCAG)
Editor	Niels Ole Bernsen (NISLab)
Author(s)	Niels Ole Bernsen (NISLab), André Berton (DCAG), Marcela Charfuelán (NISLab), Laila Dybkjær (NISLab), Mykola Kolodnytsky (NISLab), Dmytro Kupkin (NISLab), Mohamed Zakaria Kurdi (NISLab)
EC Project Officer	Domenico Perrotta, Philippe Gelin
Keywords	in-car information systems, navigation, hotel reservation, natural language understanding, dialogue management, response generation, text-to-speech synthesis

Formateret

Abstract (for dissemination)	This report from the IST/HLT VICO (Virtual Intelligent Co-driver) project describes progress made in developing the natural language understanding, dialogue management, and response generation components of the VICO system for three languages, i.e. UK English, German, and Italian, for five tasks, i.e. route navigation, point of interest navigation. VICO information, hotel selection and reservation, and restaurant reservation. In addition, the report describes the work done on observation-based user modelling and the consortium's evaluation of existing text-to-speech systems with the purpose of selecting the best current TTS system for the 1st VICO prototype.

Table of Contents

VICO 1
Table of Contents 3
Chapter contributions 6
1. Introduction 7
1.1 Scope of this report 7
1.2 List of abbreviations 7
1.3 State of progress overview 8
1.3.1 Tasks 8
1.3.2 Task/language pairs 8
1.3.3 Domain 9
1.3.4 Style of interaction 9
1.3.5 Portability 9
1.3.6 Module interfaces 10
1.3.7 Data structures 10
1.4 Innovation 10
1.4.1 NLU 10
1.4.2 DM 11
1.4.2.1 Modularity 11
1.4.2.2 Error correction 11
1.4.2.3 Observation-based user modelling 12
1.4.2.4 Task grammar 12
1.4.3 RG 12
1.5 Work process 12
1.5.1 Complexity 12
1.5.2 Rapid prototyping 12
1.5.3 Wizard of Oz simulation and experimentation 13
1.5.4 Generalisations 13
1.6 Module and system testing 14
1.7 Plan of this report 14
2. General system description 15
3. The natural language understanding component 18
3.1 Introduction 18
3.2 Architecture and data flow 18
3.3 Description of NLU enhancements 21
3.3.1 Main understanding manager 21
3.3.2 Specific Language Module Manager (SLMM) 21
3.3.3 Parsing 21
3.3.4 Semantic analysis 22
3.3.5 Arbitration module 23
3.3.6 NLU pre- and post-processor 23
3.4 Test results 25

3.5 Issues 25
4. The dialogue manager 26
4.1 Introduction 26
4.2 Architecture and data flow 26
4.3 NLU input frame example 27
4.4 DM expectations 27
4.4.1 Expectations from the DM to the SRs 28
4.4.2 Expectations from the DM to the NLU 28
4.5 The task manager 28
4.5.1 Task manager architecture and data flow 28
4.5.2 TM data flow 31
4.6 Dialogue structure example 33
4.7 Dialogue history example 36
4.7.1 Topic history 36
4.7.2 Task history 36
4.8 Domain handler scheduler 38
4.8.1 Domain handler scheduler architecture and data flow 38
4.8.2 Domain handler scheduler rule example 39
4.9 Domain agent example 39
4.10 Point of interest task 40
4.10.1 POI types and user requests for particular POIs 41
4.10.2 POI input combinatorics and DB queries 42
4.10.3 POI DB returns analysis 43
4.10.4 POI dialogue structure 43
4.10.5 POI use case 44
4.11 Hotel reservation and restaurant reservation tasks 45
4.11.1 User input for the hotel task 45
4.11.2 User input for the restaurant reservation task 47
4.11.3 Hotel and restaurant input combinatorics and DB queries 47
4.11.4 Hotel and restaurant DB returns analysis 48
4.11.5 Hotel and restaurant reservation dialogue structure 48
4.12 The user modelling module 49
4.12.1 Deciding what to model for the 1st VICO prototype 49
4.12.2 VICO UM specification 50
4.12.3 VICO UM implementation 52
4.13 CWW query XML wrapper and unwrapper example 54
4.14 Test results and objectives 55
4.15 Issues 55
4.15.1 Dialogue structure design 55
4.15.2 Recogniser behaviour 55
4.15.3 Database behaviour 56
4.15.4 Meta-issues 56
5. The response generation component 57
5.1 Introduction 57

5.2 Architecture and data flow 57
5.3 Test results 59
5.4 Issues 59
6. The Speech Synthesis Component 61
6.1 Evaluation of commercially available TTS systems 61
6.1.1 Evaluation criteria 61
6.1.2 Set of evaluation sentences 61
6.1.3 Set of TTS synthesis systems to be evaluated 62
6.1.4 Evaluation results 62
6.1.4.1 Evaluation results for German TTS systems 63
6.1.4.2 Evaluation results for British-English TTS systems 64
6.1.4.3 Evaluation results for Italian TTS systems 65
6.2 API definition for the TTS component 66
6.3 Summary of the TTs assessment 66
7. Conclusions and future work 67
8. References 68
Speech synthesis websites 68
9. Acknowledgements 69
Appendix A – Collection of synthesis test sentences 70
A1 Evaluation sentences for German TTS systems 70
A1.1 Set A. Sentences from the VICO Scenario 70
A1.2 Set B: Sentences and phrases to check the pronunciation 71
A1.3 Set C: Sentences to check word and sentence prosody 71
A1.4 Set D: Sentences and Phrases to check language-specific transcriptions 72
A.2 Evaluation sentences for English TTS systems 72
A2.1 Set A: Sentences from the VICO Scenario 72
A2.2 Set B: Sentences and Phrases to Evaluate Pronunciation 73
A2.3 Set C: Sentences and Phrases to Evaluate Prosody 73
A2.4 Set D: Sentences and Phrases to Evaluate Transcription 74
A.3 Evaluation sentences for Italian TTS systems 74
A3.1 Set A: Sentences from the VICO Scenario 74
A3.2 Set B: Sentences and Phrases to Evaluate Pronunciation 75
A3.3 Set C: Sentences and Phrases to Evaluate Prosody 76
A3.4 Set D: Sentences and Phrases to Evaluate Transcription 76

Chapter contributions

Chapter	Title	Author(s)
1	Introduction	Niels Ole Bernsen (NISLab)
2	General system description	Niels Ole Bernsen, Marcela Charfuelán, Mykola Kolodnytsky (all NISLab)
3	The natural language understanding component	Mohamed Zakaria Kurdi, Dmytro Kupkin (both NISLab)
4	The dialogue manager	Niels Ole Bernsen, Marcela Charfuelán, Laila Dybkjær (all NISLab)
5	The response generation component	Niels Ole Bernsen, Marcela Charfuelán (both NISLab)
6	The speech synthesis component	André Berton (DCAG)
7	Conclusions and future work	Niels Ole Bernsen (NISLab)

Progress Report on the Natural Language Understanding, Dialogue Management, Response Generation, and Speech Synthesis Components

1. Introduction

1.1 Scope of this report

This report describes the progress made at NISLab on the natural language understanding (NLU), dialogue management (DM), and response generation (RG) components of the VICO system in preparation of the 1st in two system prototypes due for review in November 2002. Other deliverables describing NISLab's work in this first phase of the VICO project are [Bernsen et al., VICO deliverable D9, March 2002] and [Bernsen, VICO deliverable D10, August 2002] both of which cover particular aspects of our work on the dialogue management component. The contents of these deliverables are not repeated here except to provide brief background information. In addition, the present report describes the consortium's evaluation process for selecting text-to-speech (TTS) synthesisers for the VICO system.

1.2 List of abbreviations

It may be useful for the reader to have ready-at-hand a list of the abbreviations to be encountered below:

CS = confidence score
CWW = car wide web
DA = domain agent
DA_POI = POI domain agent
DA_HOTEL = hotel domain agent
DA_ROUTE = route domain agent
DB = database
DHa = domain handler
DHI = dialogue history
DHS = domain handler scheduler
DM = dialogue manager
DS = dialogue structure
DvM = device manager
FRT = filled response template
HDA = hotel domain agent
HRUM = hotel reservation UM
LR = linguistic realiser
NLU = natural language understanding
POI = point of interest
RG = response generator
RTF = response template filler

SLMM = specific language module manager
SM = system manager
SS = speech synthesis
ST = sentence template
TA = technical annex
TaHI = task history
TM = task manager
TMS = task manager scheduler
ToHI = topic history
TTS = text-to-speech
UM = user modelling module
UM_HOTEL = user model for the hotel task
UMDA = user modelling domain agent
UMS = user modelling module scheduler
URT = unfilled response template
WHG = word hypothesis graph

1.3 State of progress overview

The three main modules to be reported on in what follows have been developed according to the specification in the Technical Annex (TA) to the VICO contract as well as to the TA amendments decided by the consortium as the project has developed. The main changes have been that, instead of developing a general restaurant reservation task, we have (a) developed a restaurant reservation task for the hotel selected and booked by the driver in the hotel reservation task, and (b) developed a VICO information task for English in which the driver can obtain information about the VICO system itself. This change was made because it seemed obvious that a system as complex as VICO will need to be able to explain its functionality and manner of operation to the drivers. In this way, we are able to demonstrate both restaurant reservation and VICO information.

The following is an overview of the work done.

1.3.1 Tasks

The 1st VICO prototype helps the driver solve the following tasks:

route task: negotiating and receiving navigation guidance to specified addresses;

POI task: negotiating and receiving navigation guidance to specified points of interest (POIs);

hotel reservation task: negotiating hotel selection, getting additional hotel information, booking room(s) at the hotel, each driver being supported in the hotel selection sub-task by an observation-based user-modelling module;

restaurant reservation task: booking a table at the restaurant in the booked hotel (cf. 3);

VICO information task: getting information about what VICO can do, how to operate VICO, and how the VICO technology works.

1.3.2 Task/language pairs

Each VICO task has been developed for a number of languages, as shown in Table 1.1. For each of the 13 task/language pairs shown in the table, we have developed (a) an input grammar and a lexicon for the NLU and (b) a set of output utterances for the RG.

Task -> Module/Language	Route	Point Of Interest	Hotel reservation	Restaurant reservation	Information
NLU/English	x	x	x	x	x
NLU/German	x	x	x	x	
NLU/Italian	x	x	x	x	
RG/English	x	x	x	x	x
RG/German	x	x	x	x	
RG/Italian	x	x	x	x	

Table 1.1. Task/language pairs for the 1st VICO prototype.

1.3.3 Domain

The domain of the 1st VICO prototype is the beautiful Trento province in North-Eastern Italy. Thus, task addresses, points of interests, and the hotels and restaurants which can be booked or reserved are all located in the Trento province.

1.3.4 Style of interaction

The driver-VICO dialogues are spontaneous dialogues initiated by the driver. Thus, dialogue with VICO does not require any learning and subsequent memory of command words or phrases. Also, the dialogues are mixed-initiative dialogues in which the user has the initial initiative and always has the option of taking the initiative, changing the topic or the task at will, etc. Some would call such dialogues conversational dialogues. However, the dialogues are still task-oriented rather than being conversational in the full sense of abandoning the task-orientation.

1.3.5 Portability

A main goal throughout is to enable easy portability to new tasks of the technology developed.

At component level, we continue to push towards identifying the right modularity-for-portability for the NLU, DM, and RG, always seeking to maintain clear separations between task and language-dependent functionality, on the hand, and task and language-independent functionality on the other. The ideal of complete task-independence of the NLU-DM-RG complex probably is just that. However, approximation to the ideal is clearly possible. We have found that two factors, at least, militate against achieving the portability ideal in VICO. One factor is the fact that it may take as much as two months to specify the program for implementing a new task, and VICO is planned to incorporate in the order of 7 tasks. This means that it is not possible to specify all tasks before starting implementation, and this again means that new tasks may include surprises that may challenge the architecture and data flow of the system already in existence. The second factor is the high speed of implementation in VICO, which makes elaborate implement-and-revise experimentation at main component level rather difficult. This having been said, we now appear to have built high-modularity main components which appear likely to be able to cope with the new tasks to come in the second phase of the project.

Easy and rapid portability is not just an issue of modularity, it is also an issue of following well-defined procedures and being trained to do so. At procedural level, and after several phases of experimentation and sometimes also training of people, we are now in the position of following well-defined procedures for specifying new tasks, representing their specification in a format which is immediately intelligible to the programmers and which has a good fit with the architectures and data flow of the main components, carrying out rapid parser testing, doing rapid generation for a new language/task pair, doing CORBA integration, etc.

Following the upcoming testing of the 1st VICO prototype as a whole, we expect to add to our stock of procedures in several ways. For instance, we plan to develop a more detailed set of guidelines for dialogue structure (DS) design and implementation.

The above hypotheses will be tested in the second phase of the project, the key prediction being that the addition of new tasks to the system will be done at an increasing pace.

1.3.6 Module interfaces

We have developed pre- and post-processing interfaces to the NLU in order for the NLU, which has been developed in Prolog, to communicate with the C++ speech recognisers for English, German, and Italian, as well as for the NLU to communicate with the C++ dialogue manager. The RG, on the other hand, having been developed in C++, does not require pre- and post-processing in order to communicate with the DM.

To enable the DM to communicate with the Car-Wide-Web (CWW) –based databases (DBs), we have developed wrapper/unwrappers which express queries to the CWW in XML and decode DB responses from XML.

For general inter-module communication, we have developed CORBA interfaces between the three main modules (NLU, DM, RG) and the system manager (SM).

1.3.7 Data structures

The *lingua franca* data structure used by all main modules (NLU, DM, RG) is the frame. Thus, the NLU produces a frame-based semantic representation of the user's input and the DM outputs a frame-based semantic representation to the RG. Internally in the DM, the frame is being widely used for data storage and exchange. Other data structures are being used as well, such as such as topic history data type, task history data type, strings, and XML file format is used for exchanges with the CWW.

1.4 Innovation

1.4.1 NLU

Significantly generalising an existing in-house NLU component, we have developed a highly modular VICO NLU component which includes modular parsing units per task, receives DM expectations as to the next user input, produces parsing confidence scores, and combines speech recogniser confidence scores and parsing confidence scores into a joint confidence score which constitutes an important factor in the DM's decision-making on which output to produce in context. Some actual and expected advantages of the new modular architecture are:

1. Speed of processing: the modular parsing architecture allows to reduce the size of the search space by looking only at the sub-set of the rules that are relevant for the input being parsed.
2. Ambiguity resolution or reduction: by reducing the search space we also reduce the ambiguities. For example, at the full grammar level, numbers can trigger different slots in different frames (number of street, number of persons, room numbers, etc.). When an input is marked as being a route input, it is parsed by the route sub-grammar and there it can only trigger a street number slot.
3. Grammar writing organisation: distinguishing between task-dependent and task-independent rules makes it possible to organise the grammar writing procedure. For example, each part of the grammar can be put in a separate file which makes its maintenance easier than if the entire grammar, which, at the end of the project, may contain more than one thousand rules, was put into a single file. Furthermore, this approach makes it possible to have more than one linguist work on the same grammar

at the same time, i.e. by asking each to write a grammar for a sub-task. This will help speed up grammar writing in the second phase of VICO.

4. Portability: the fact that we divide the grammar into task-dependent and task-independent parts helps limiting the amount of work necessary to write a new grammar for a new task, because we only need to write the new rules that are relevant to this task. Again, this will help speed up grammar writing in the second phase of VICO.

1.4.2 DM

The VICO dialogue manager has been developed from scratch.

1.4.2.1 Modularity

The DM maintains a clear separation between the processing of task-independent and task-dependent information, respectively. The latter is done by task-specific domain agents (DAs), one per task, which communicate with the CWW for DB and other information retrieval. Task-independent processing is done by the task manager (TM) which follows task-independent procedures in consulting, internally to the DM, with task-specific dialogue histories (DHIs), task-specific dialogue structures (DSs), and the already mentioned task-specific DAs. Similarly, the TM follows task-independent procedures in its communication with the following external main modules: SR, NLU, CWW, RG, and SS. This architecture makes it possible to add a new task without modifying the general procedures of the task manager, thereby approximating the easy portability to new tasks which is one of the key objectives of the VICO project.

1.4.2.2 Error correction

Given the ambitious nature of VICO in the context of the state of the art, in particular with respect to very large vocabulary speaker-independent speech recognition in noisy conditions and the aim of handling multiple tasks in spontaneous mixed-initiative dialogue, it is clear that VICO's dialogue management needs powerful error-correction functionality to succeed. We have found it necessary to address the issue of error correction from at least three angles:

1. The first one is the classical one of specifying a dialogue which enables *graceful degradation* of the system's responses when there are difficulties in recognising and understanding the user's input by the speech recogniser and the NLU. This angle or aspect has been addressed at design-time by specifying slightly different experimental variations of the dialogue structures for each VICO task.
2. The second one is the more recent approach of taking SR and NLU *confidence scores* into account on-line in order for the DM to dynamically modify its output in the light of the confidence scores. Thus, the management of all VICO tasks include the taking into account of three different levels of confidence in the system's recognition and understanding of the user's input. Both of the above error-correction strategies have been implemented at task level.
3. However, given the rather radical multiple-task nature of the VICO system, we need a third approach to error handling as well. The NLU-DM-RG complex needs to be able to handle *meta-task input issues and problems* of various kinds, such as when the NLU cannot tell from the user's input which task the user intends to address. This meta-task error-correction aspect is still very much ongoing work because we need to further investigate the scope of the issues involved before being able to design and implement a general meta-task error-correction strategy.

1.4.2.3 Observation-based user modelling

The DM's hotel reservation DA includes a user modelling module (UM) which incorporates sub-module functionality (UM_HOTEL) for observing the individual driver's hotel preferences, building and updating in a database a model of that driver's hotel preferences, and making this model available to the DA_HOTEL when it is engaged in hotel selection conversation with the driver. The support offered to the driver during hotel selection has two aspects: when the UM_HOTEL has no information about the driver's whereabouts (i.e. about the location of the car), UM_HOTEL offers generic support to hotel selection, such as knowledge about the number of stars of the hotels preferred by the driver. When UM_HOTEL does have information about the driver's location, it offers, in addition, location-based hotel selection support based on knowledge of the hotels which the driver has previously visited in the area in question.

1.4.2.4 Task grammar

As a fourth innovation, the DM includes a growing meta-dialogue management structure for handling task interdependencies. Thus, for instance, the driver should be able to interrupt a task and return to it later without having to repeat what has already been agreed with VICO, and VICO should be able to interrupt the driver's current task-oriented dialogue by generating a driving instruction, such as "turn next left". For lack of a better term, we call this meta-dialogue management structure for a *task grammar*. Again, as for the meta-task input issues and problems described in Section 1.4.2.2 above, this is ongoing work. We need to investigate what the task interdependencies are as we keep adding tasks to VICO, before we can develop a final, robust and general solution to them.

1.4.3 RG

The template-based response generation component of VICO has been developed from scratch. The RG maintains a clear separation between language-independent processing which is done by its first module, the response template filler (RTF), and language-dependent processing which is being done by its second module, the linguistic realiser (LR). The linguistic realiser uses an output database segmented into sets of language/task pair responses for storing the output which can be sent to the text-to-speech (TTS) module for output synthesis.

1.5 Work process

1.5.1 Complexity

Somehow interlinked, VICO is both extremely rewarding as a research endeavour and represents a complex interdisciplinary challenge as regards NLU, DM, and RG analysis, design, implementation, and testing. Thus, at NISLab, some 16 people are involved in VICO system development in many different roles. Throughout, the sheer number of system modules to be developed, tested, and integrated is in competition with the need for experimentation with problems and solutions which are new in the context of the state of the art. Although the Lab as such is not new to CORBA integration, the people actually doing it were new to CORBA. Similarly, we were all new to complex geographical information system (GIS) databases, such as those of TeleAtlas.

1.5.2 Rapid prototyping

The approach adopted to system development was that of rapid prototyping. The first NISLab prototype was ready in December 2001. It addressed the English version of the route task and included a simple NLU, a first prototype DM, and a first prototype RG. It worked with a fixed grammar, command-based US English DaimlerChrysler recogniser rather than with the

DCAG recogniser used for the 1st VICO prototype. It used Festival for English speech synthesis and had access to an in-house-made geographical database containing Danish addresses. The choice of Danish addresses was made to make the route task familiar to English speaking Danish test subjects. Three levels of confidence scores were used to support dialogue management on-line.

1.5.3 Wizard of Oz simulation and experimentation

The rapid prototype was user tested during December 2001 through February 2002, which yielded valuable information on user's linguistic and non-linguistic behaviour during communication with VICO. We used a variant of the Wizard of Oz (WoZ) methodology for the user testing. The users would use non-priming scenario representations to tell them what to talk to VICO about in spontaneous (English) speech. During conversation with VICO, the users would drive a car in a car game in front of a 42" screen. Next to the screen, the user would have VICO's text output available on a portable PC screen. The wizard would fill an NLU output frame with the user's input and send a text string to Festival for output. Of particular interest was the experiment we made on the nature and length of the on-screen text output presented to users during the dialogue. The conclusion was that users may benefit from brief and to-the-point text output on the current state of achievement of their task objective, cf. [Bernsen and Dybkjær 2001].

1.5.4 Generalisations

Based on the WoZ simulations, we analysed which generalisations were needed of the NLU, DM, and RG in order for these modules to meet the requirements of a multilingual system able to handle multiple, sometimes rather different, and at least partially interdependent tasks. The conclusions arrived at were as follows:

For NLU development, the WoZ simulations made it clear which modular NLU architecture to aim for, so that work could start on generalising the NLU which was used for rapid prototyping.

For RG development, the analysis made it clear that we needed to partially re-implement the RG in order to (a) ensure clean separation between the DM and the RG, and (b) ensure clean separation between language-independent processing and language-dependent processing.

As regards DM development, whilst the DM seemed to work quite well for the route task, we faced several major uncertainties. The first uncertainty was that it is difficult to anticipate the issues involved in handling new tasks which have not yet been analysed and specified. Whilst architecture and data flow analyses of the then-existing DM seemed to suggest that the DM was sufficiently modular for handling new tasks whilst preserving clear separations between task-specific and task-independent knowledge, the proof of the pudding would lie in the eating. We continued to specify and implement new tasks, including the POI task, the VICO information task, the hotel reservation task, and the restaurant reservation task. However, it is only at the time of writing that we begin to be able to claim that the DM actually is proving capable of enabling the addition of new tasks in a modular fashion. A second uncertainty which we are only now able to address, i.e. with the existence of an integrated 1st VICO prototype, is how the NLU-DM-RG complex will behave when integrated with all other VICO modules. For instance, peculiarities in the behaviour of the SR or the DBs may demand modifications of the NLU and/or the RG.

The analysis also made it clear that we needed the task grammar referred to in Section 1.4.2.4 above. So, a first task grammar was specified and implemented.

Finally, the analysis uncovered an increasing number of meta-task issues, such as NLU task ambiguity in which the NLU cannot tell from the user's input, nor from the input predictions it receives from the DM, which task the user is addressing, "empty" task input which does not

address any particular task at all, user task cancellations, users' asking VICO to repeat what it just said, users who continuously produce noisy input or very low confidence scores, etc. (cf. Section 1.4.2.2). The decision taken was to postpone a single general solution to these issues until we have a better grasp of their extent, i.e. when we (a) have studied the behaviour of the integrated 1st prototype and (b) have tested with real users our implementation of a goodly number of tasks. Meanwhile, the mainly task-specific dialogue structures used in the 1st VICO prototype take care of some of the issues listed above.

1.6 Module and system testing

The module and system testing done so far includes the rapid prototype user testing described in Section 1.5.3 above, blackbox testing of the rapid prototype DM and its response generation, extensive software debugging, integration testing in connection with the delivery of modules for the 1st VICO prototype, and encouraging but limited parser testing. The 1st prototype provides the occasion for systematic module and system testing with and without users in the loop, followed by the necessary revisions to the NLU, DM, and RG modules, the laying down of general dialogue strategies across tasks, identifying a general solution to the meta-task issues, etc.

1.7 Plan of this report

The route task and first VICO information task specifications are documented in [Bernsen et al., VICO deliverable D9, March 2002]. The user modelling analysis preceding the specification and implementation of the observation-based user modelling module for drivers' hotel preferences is documented in [Bernsen, VICO deliverable D10, August 2002]. The present report documents the remainder of the analysis, specification, implementation, and testing work on NISLab's parts of the 1st VICO prototype. Below, Chapter 2 presents a general view of the VICO system, focusing on the NLU, DM, and RG components. Chapter 3 describes the NLU component. Chapter 4 describes the DM component. Chapter 5 Describes the RG component. Chapter 6 describes the consortium's selection of speech synthesis (SS) component. Finally, Chapter 7 draws some main conclusions and briefly describes the second phase of our work on VICO.

2. General system description

The main modules of the VICO system are shown in Figure 2.1. The modules are glued together using CORBA. Different programming languages are used for the different modules. For instance, C++ is used for the DM and RG, Prolog is used for the NLU and Java is used for the CWW. All modules communicate via the system manager (SM). The four modules addressed in this report are the NLU, the DM, the RG, and, with respect to comparative evaluation results only, the speech synthesiser (SS).

Figure 2.1 presents an overall view of the entire VICO system, highlighting the roles of the three main NISLab modules. The NLU receives a word hypothesis graph (WHG) including confidence scores from the SR and expectations as to the next user input turn from the DM. Having processed the WHG, the NLU sends a filled frame to the DM. The DM sends expectations to the SR on which SR units should be active for the next user input turn and receives a frame from the NLU. If and when needed for dialogue progress, the DM queries the CWW. The query is wrapped in XML. The return to the DM from the CWW is also wrapped in XML. When the DM knows what to communicate to the user in the next output turn, it sends the semantics of what has to be said to the RG. At this point, the DM also sends expectations regarding the next user input to the SR and the NLU. Based on the semantic input from the DM, the RG produces the actual text to be synthesised and sends it to the SS. In addition, the RG produces the corresponding text to be displayed on the small in-car screen and sends it to the DvM. All communication between the modules is via the SM.

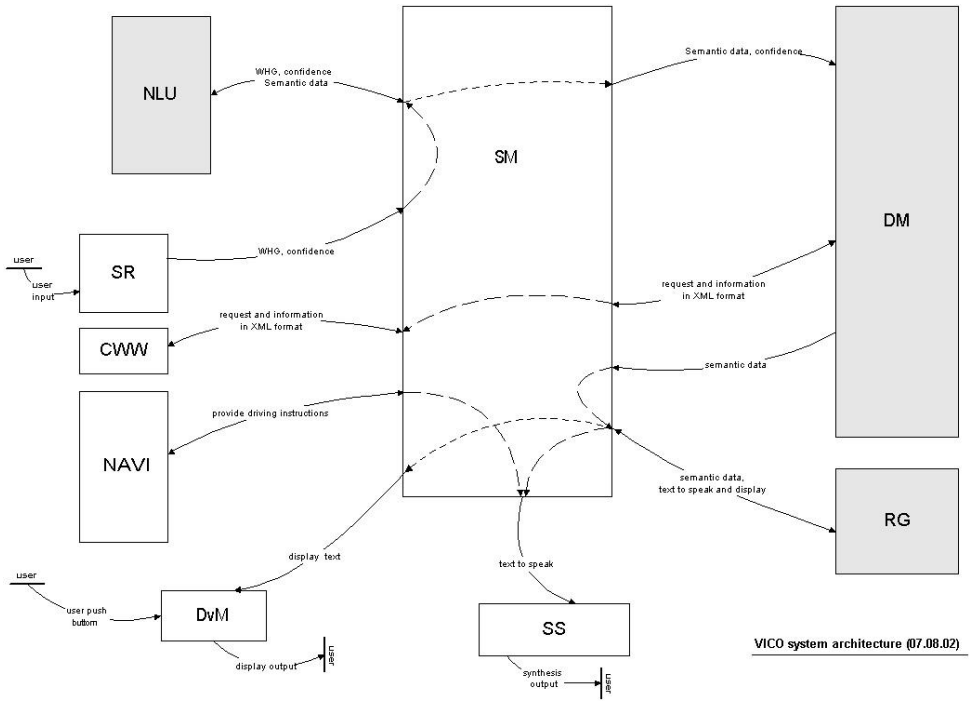


Figure 2.1. High-level system architecture and data flow, highlighting NISLab's main modules (in grey).

Figure 2.2 provides a more detailed view, again adopting the point of view of NISLab's main modules by numbering the data transfers involving those modules. The numbered data flow is explained in Table 2.1.

The following three chapters include description of the goals, functionality, and internal architecture of the NLU (Chapter 3), the DM (Chapter 4), and the RG (Chapter 5).

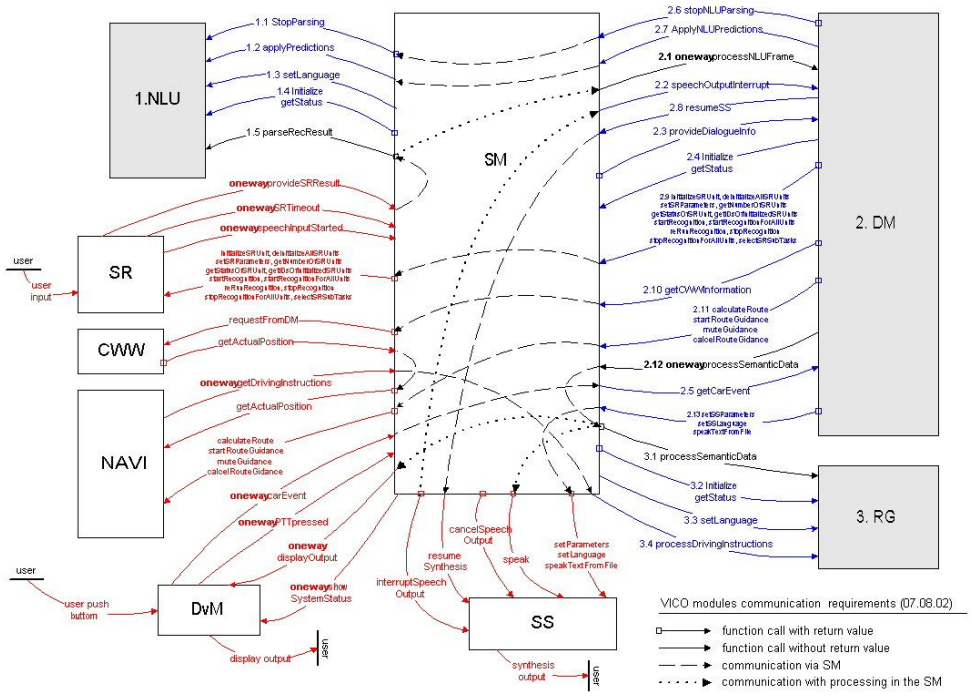


Figure 2.2. A more detailed view of the data flow in Figure 2, highlighting the roles of the NLU, DM, and RG.

Module	No	Message	Corresponding Function in the IDL file
NLU (server) Functions implemented in the NLU.	1.1	stopParsing	boolean stopParsing()
	1.2	applyPredictions	void applyPredictions(in NLUPredictionType pPredictions)
	1.3	setLanguage	void setLanguage(in languageID pLanguage)
	1.4	initialise, getStatus	boolean initialize(in string pConfigPathAndFilename) short getStatus()
	1.6	parseRecResult	NLUFrame parseRecResult(in WHGSequence pRecResult)
DM (server) Functions implemented in the DM.	2.1	oneway processNLUFrame	oneway void processNLUFrame(in NLUFrame pNLUFrame)
	2.2	speechOutputInterrupt	void speechOutputInterrupt(in speechPointer pSSAbortTag)
	2.3	provideDialogInfo	string provideDialogInfo()
	2.4	initialise getStatus	boolean initialize(in string pConfigPathAndFilename) short getStatus()

	2.5	getCarEvent	void getCarEvent(in carEventLabel pEventDescriptor)
DM (client) Interface used by DM, these are functions used by the DM but implemented in other modules.	2.6	stopNLUParsing	boolean stopNLUParsing()
	2.7	applyNLUPredictions	void applyNLUPredictions(in NLUPredictionType pPredictions)
	2.8	resumeSS	void resumeSS(in speechPointer pResumePoint)

	2.9	initializeSRUnit
		deInitializeAllSRUnits
		setSRParameters
		getNumberOfSRUnits
		getStatusOfSRUnit
		getIDsOfInitializedSRUnits
		startRecognition
		startRecognitionForAllUnits
		reRunRecognition
		stopRecognition
		stopRecognitionForAllUnits
		selectSRSubTasks
			boolean initializeSRUnit(in string pConfigPathAndFilename, in short pUnit);
			void deInitializeAllSRUnits();
			void setSRParameters(in SRParameterSelector pParamID, in long pParam);
			short getNumberOfSRUnits();
			short getStatusOfSRUnit (in short pUnit);
			unitSequence getIDsOfInitializedSRUnits();
			boolean startRecognition(in unitSequence pUnits, in string pWavePathAndFilename);
			boolean startRecognitionForAllUnits(in string pWavePathAndFilename);
			WHGSequence reRunRecognition(in unitSequence pUnits, in string pSoundfile);
			boolean stopRecognition(in unitSequence pUnits);
			boolean stopRecognitionForAllUnits();
			boolean selectSRSubTasks(in unitSequence pUnits, in stringSequence pSubTaskNames);

	2.10	getCWWInformation	octetSequence getCWWInformation(in octetSequence pRequest);

	2.11	calculateRoute
		startRouteGuidance
		muteGuidance
		cancelRouteGuidance
			boolean calculateRoute(in location pLocationRequest);
			boolean startRouteGuidance(in guidanceSelector pDemoOrSimu);
			boolean muteGuidance();
			boolean cancelRouteGuidance();

	2.12	oneway processSemanticData	oneway void processSemanticData(in DMOutput pSemanticFrames, in string TMtemplate, in short pPriority);

	2.13	setSSParameters
		setSSLanguage
		speakTextFromFile
			boolean setSSParameters(in SSParameterSelector pParamID, in short pParam);
			void setSSLanguage(in languageID pLanguage);
			void speakTextFromFile(in string pTextFile);

RG (server) Functions implemented in the NLU.	3.1	processSemanticData	RGResult processSemanticData(in DMOutput pSemanticFrames, in string TMtemplate, in short pPriority)
	3.2	initialise getStatus	boolean initialize(in string pConfigPathAndFilename) short getStatus();
	3.3	setLanguage	void setLanguage(in languageID pLanguage)
	3.4	processDrivingInstruction	void processDrivingInstruction(in string pDrivingInstruction)

Table 2.1. Detailed description of functionality of the NLU, DM and RG modules. Numbers refer to the data flow shown in Figure 2.2.

3. The natural language understanding component

3.1 Introduction

The goal of the NLU is to accept sequences of word hypothesis graphs (WHGs) from the SR and analyse them linguistically using robust ("island") parsing. The parser result is inserted into a semantic frame and passed on to the DM for further processing. The NLU handles three different languages, English, German, and Italian. Language-setting is being done during initialisation by the SM.

The VICO NLU is based on our previous system Oasis (Kurdi 2001). Oasis is serial system in which input processing follows three main steps:

1. pre-processing: the pre-processing module is intended to detect and correct spoken language extragrammaticalities. It is based on a combination of pattern matching techniques and a shallow syntactic chunker (Kurdi 2002).
2. parsing: this step is done by combining a partial parsing approach to a semantics-based grammatical formalism: the Semantic Tree Association Grammar (Sm-TAG).
3. semantic analysis: the semantic analysis module in Oasis takes a parsing tree as input and produces a semantic representation in the IF (Interchange Format) formalism. The IF formalism was defined as an Interlingua formalism within the framework of the C-STAR and Nespole projects, both of which are in the field of speech-to-speech translation.

In order to adapt the NLU to the heavy demands to natural language processing in VICO, the NLU has been significantly enhanced, as follows. The NLU has been:

1. adapted to receive WHGs from the VICO SRs;
2. equipped with a modular architecture that includes several parsing and semantic analysis units that can be activated following the contents of the current input as well as the dialogue context in which this input has been produced;
3. augmented with an arbitration module that includes a parsing scorer, a module for calculating the global speech recognition score, and a ranking module that combines the later scores and selects the best parse;
4. In addition, implemented in Prolog, the NLU uses a C++-to-Prolog pre-processor which converts the SR WHGs so that these can be processed by the NLU, and a Prolog-to-C++ post-processor which converts the NLU-produced frame information string into a C++ frame which can be processed by the DM.

The designed and implemented NLU enhancements are described in more detail in Section 3.3.

3.2 Architecture and data flow

The NLU architecture is based on division of labour among modules according to language and task. This allows a better sharing of common resources across tasks and languages. The architecture of our NLU is presented in Figures 3.1 and 3.2.

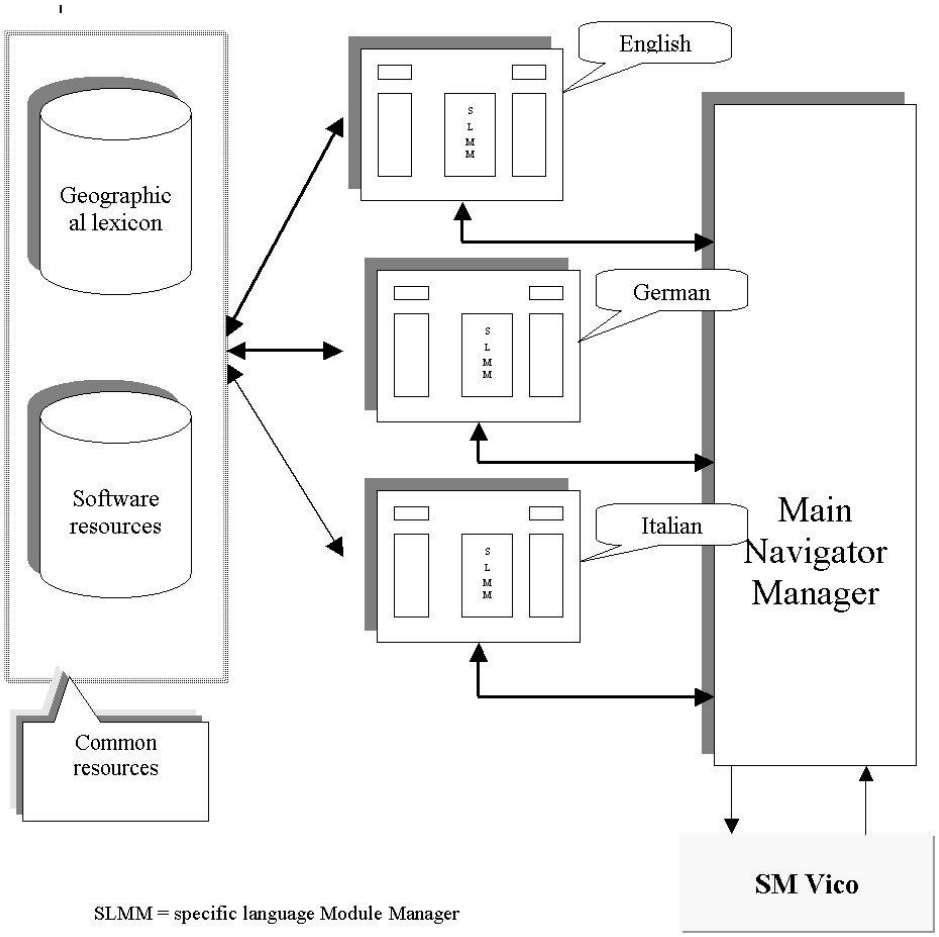


Figure 3.1. High-level architecture for the VICO natural language understanding module.

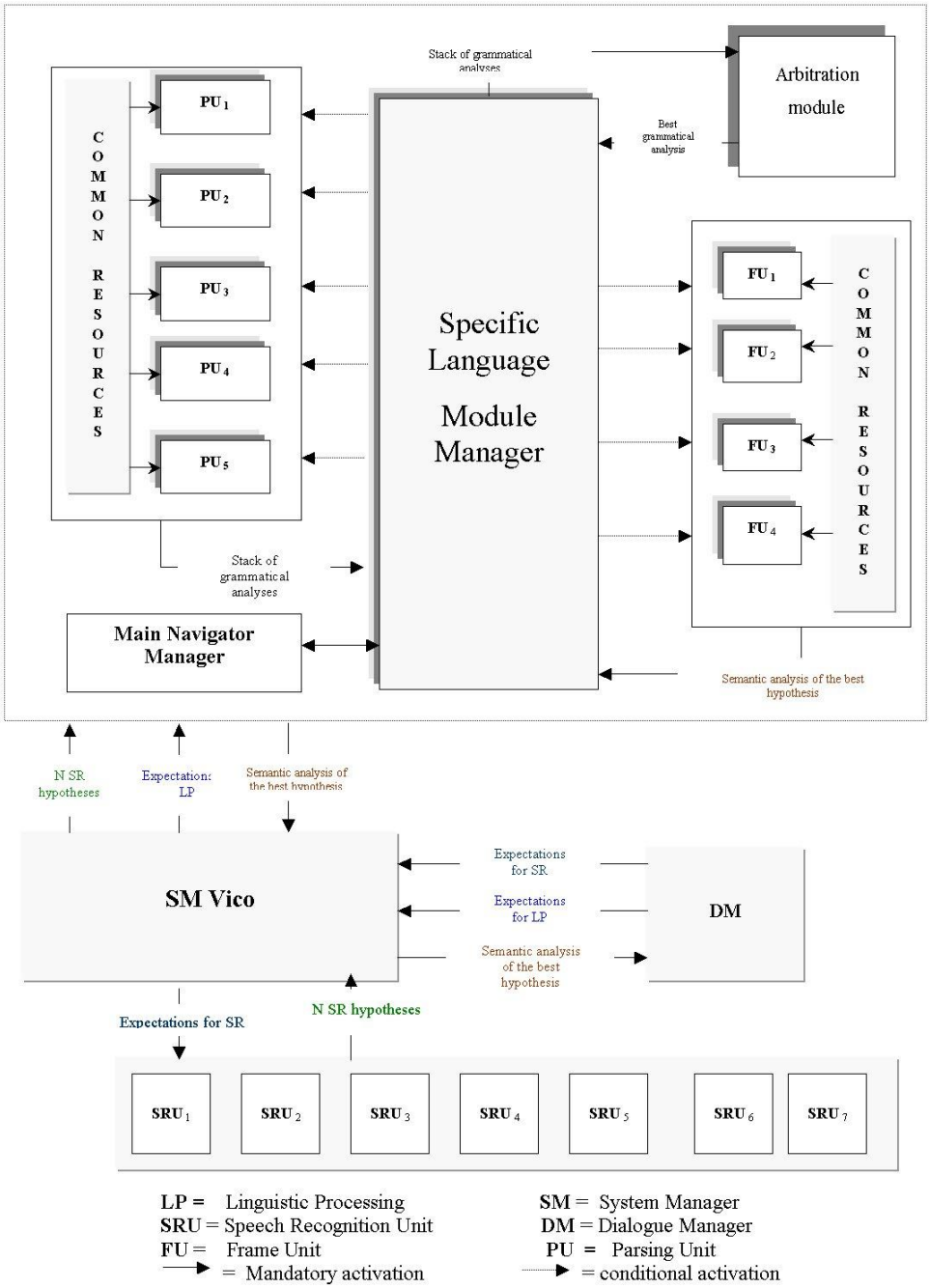


Figure 3.2. Architecture and general data flow for a language-specific VICO NLU module.

3.3 Description of NLU enhancements

3.3.1 Main understanding manager

The main function of the main understanding manager is to serve as a common interface between, on the one hand, the SM and, by consequence, the rest of VICO's modules, and, on the other, the different language-specific understanding components. Another function of this module is to activate the relevant language for the current dialogue following the information provided by the system manager.

3.3.2 Specific Language Module Manager (SLMM)

The SLMM module is responsible for the interaction between, on the one hand, the language-specific understanding modules, such as the English-specific modules, and the main understanding manager, and, on the other, between the language-specific understanding modules themselves. As for the internal interaction inside a language-specific module, the SLMM distributes the information according to its source, such as the speech recognition unit from which the WHG sequences come, its nature (parse tree, semantic frame), the current dialogue task, etc.

3.3.3 Parsing

Parsing is done by using an approach similar to the one used in Oasis, i.e. a combination of a partial parsing algorithm and the Semantic Tree Association Grammar (Sm-TAG). The work achieved in the parsing module can be divided into two parts:

1. Grammar writing: the first step in grammar writing consists in defining a common format for the grammars in the three languages English, German, and Italian. The general schema of the rules is the following:

   ```
   (Left_hand_non_terminal,
   [The_list_of_the_right_hand_non-terminals],
   Comment1,   /*   segment example in German/italian and English */
   Comment2,   /*   utterance in German/italian   */
   Comment3,   /*   utterance in English   */
   Comment4),  /*   Personal comment   */
   ```

   1.1. This format allows expression of some additional constraints on the elements, such as unbounded repetition (if an element can repeat more than one time), single repetition, alternatives (the different items that can occupy the same place in a rule), optional (if an element is not mandatory to trigger the left-hand side non-terminal), etc. The grammar writing for the three languages was done by three different linguists. We used simulated corpora for grammar writing in the three languages.

   1.2. During the grammar writing procedure, particular effort was put into increasing the coverage of the grammars by adding new lexical entries taken from the thesaurus, or adding new combinations of rules that were not observed in the data.

2. Grammar implementation: the main task in grammar implementation consists in the conversion of the grammar into recursive transition networks (RTRs). This is done by a compiler that we developed specially for this purpose. An example of Sm-TAG rules and the corresponding RTRs produced by the compiler is shown in Table 3.1.

3. Grammar integration: this step consists in defining manually the sources-sharing between the different compiled grammars (that are each relevant for a specific domain). The general schema of the grammar writing and implementation procedure is presented in Figure 3.3.

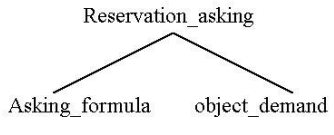
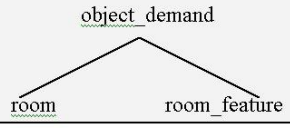
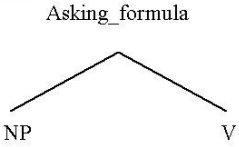
Sm-TAG basic rules	Equivalent RTRs generated by the compiler
	initial(0, reservation_asking). final(2, reservation_asking). arc(0,1, formule_demande, reservation_asking). arc(1, 2, objet_demande, reservation_asking).
	initial(0, object_demand). final(2, object_demand). arc(0,1, room, object_demand). arc(1, 2, room_feature, object_demand).
	initial(0, asking_formula). final(2, asking_formula). arc(0,1, np, asking_formula). arc(1, 2, v, asking_formula).

Table 3.1. Example of Sm-TAG rules and the corresponding RTRs.

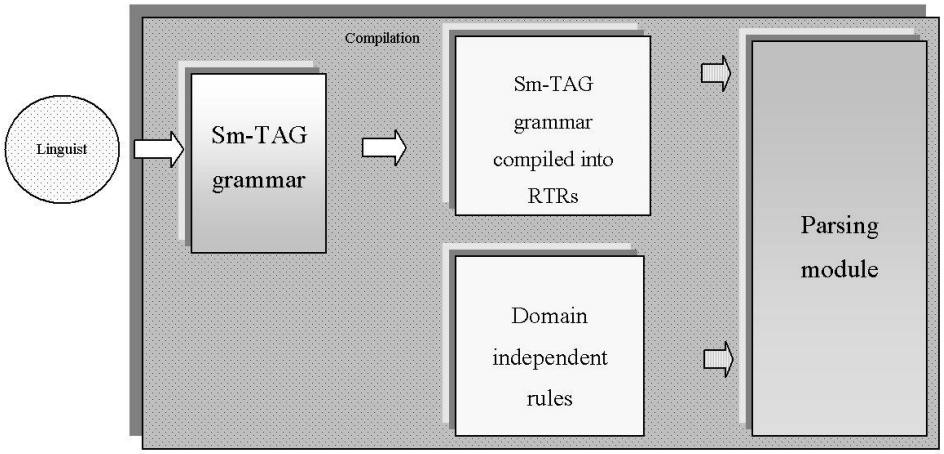


Figure 3.3. Grammar writing and implementation in VICO.

3.3.4 Semantic analysis

The semantic analysis module builds semantic frames from the parsing trees received from the parser. Each frame contains a set of slots necessary for its activation. Modular implementation of the frame allows us to take into consideration cross-frame slots, such as asking for repetition or asking for a particular possibility.

3.3.5 Arbitration module

The arbitration module can be viewed as a multi-criterion classifier whose main function is to take the Nbest parses (corresponding to the Nbest WHGs provided by the SR) and select the best one (Kurdi 1999). Two main sources of information are used in the classification procedure: parsing score and global speech recognition score.

1. Parser confidence score computation: the main idea behind this score that we can associate to the parses provided by the parser a number corresponding to the completeness of the generated parse tree.
   1.1. Classification of the chunks: in this step we convert the chunks provided by the parser into two types of categories: noisy chunks and relevant chunks. Each of the chunks is associated with a Chunk Coverage Score CCS. The CCS of a noisy chunk is the number of unparsed words and the one of a relevant chunk is the number of the rules (sub-trees) contained in this chunk.
   1.2. Local parse scoring: in this step we associate a special weight to each of the chunks. The weight of a noisy chunk is negative while the one of the relevant chunks is positive. The result of the weighting step is the association of a local score to each of the chunks. This score is computed as the following: Local Score = Chunk_Weight * CCS.
   1.3. Global Parse Scoring: the global score of a parse tree depends on two factors: the local scores of the existing chunks and the length of the input. So the global parse score is computed as the following: sum of local scores / N * length of the input. Where *N* is a weight found empirically.
2. Global speech recognition score calculation: the global speech recognition score of the entire hypothesis is computed following four steps:
   2.1. Semantic weighting of the single words scores: this step consists in associating a special weight to the speech recognition score of each word. Three categories are distinguished: words that can fill a slot in the frame (geographical names, cardinals, etc.), words that play a key role in the meaning construction of the utterance (e.g. adverbs, verbs, names), rest (e.g. determinants, adjectives). The weights associated to each category are found empirically.
   2.2. Calculation of the global score from weighted individual scores: this is the cumulated score of the weighted scores / sum of the semantic weights.
3. Combination of parsing score and global speech recognition score and selection of the best parse: the average of the parsing score and the global recognition score is used to select the best parse.
4. Normalisation of the combined score of the best parse: the combined score is converted into one of three values (one, two or three) that can be used for the DM to select the best dialogue strategy.

3.3.6 NLU pre- and post-processor

The NLU module consists of pre- and post-processor sub-units created in C++ as well as the NLU itself which has been created in Prolog.

The goal of the NLU pre- and post-processor sub-units is to enable communication between the core NLU Prolog subunit and other main modules of the VICO system. In particular, the NLU pre-processor enables communication from the speech recognisers (SRs) to the NLU, and the NLU post-processor enables communication from the NLU to the dialogue manager (DM). As indicated in Figure 3.1, all communication with other main VICO modules is brokered by the CORBA system framework and goes via the system manager (SM).

The module communication shown at a high level of abstraction in Figure 3.4, is as follows. The pre-processor accepts a sequence of WHG structures (one per active recogniser unit) from the SM. First, the pre-processor creates a string which includes all the information retrieved from the WHG structure sequence. This string is created in a format appropriate for input to the NLU Prolog module.

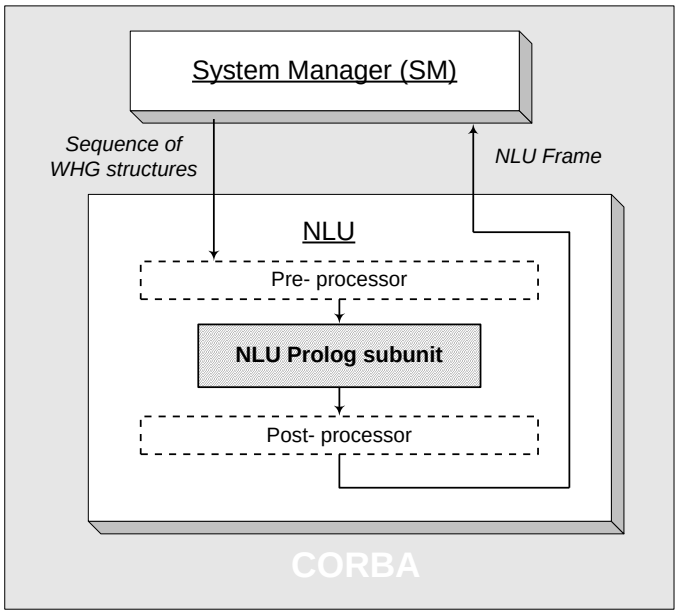


Figure 3.4. The VICO natural language understanding module wrapped in (a) its C++ pre- and post-processor modules and (b) in CORBA.

To illustrate, suppose the user said: "Take me to Via Roma twenty in Bolzano". The pre-processor output string may look as follows: "english empty SRU0 take 0.53 me 0.44 to 0.45 Via_Roma 0.9 twenty 0.16 in 0.50 Bolzano 0.77". The word "english" means that the recogniser works with the English language. The values following each word in the string are the word confidence scores determined by the SR.

The pre-processor then launches the NLU Prolog module and passes the string to it.

Following the parsing of the string, the NLU Prolog module sends an output string for processing by the post-processor. In the present example, the output string is: "[[route_frame, [[[[street_name, Via_Roma], [number_street, 20]]], [city_name, Bolzano]]]], [confidence_score, 3]]".

Type	**route**
City	Bolzano
Street	Via Roma
Number	20
ConfScore	3

Table 3.2. Frame filled by the NLU post-processor.

The task of the post-processor is to fill the frame slots in accordance with the NLU's output string. The filled frame is returned to the SM which forwards the frame to the DM. In the present example, the frame slots filed by the NLU are shown in Table 3.2.

3.4 Test results

This section describes the results of a first test of the NLU parser. The test involved a single processing unit consisting of one parsing unit and one semantic unit for the route task. The test corpus consisted of 199 pairs of input utterances, each pair consisting of the transcribed input utterance and the corresponding SR output. The corpus was created by VICO partners Bosch and DCAG based on early Wizard of Oz experiments in a car simulation environment (see [Manstetten et al., VICO deliverable D7, 2002]). The corpus included examples of input sentences addressing many different VICO tasks as well as out-of-(VICO)-task input utterances. This provided a relatively severe challenge for the NLU route task parser, because we had occasion to not only test how the parser handles route task input proper but also how the parser handles out-of-(route)-task input. As a base-line, the route parser should not parse input pertaining to other VICO tasks nor should it parse input pertaining to out-of-VICO-tasks. Rather, those kinds of non-route task input should not be parsed by the route task parser at all but should be either delegated to other parsing units, such as the point of interest parser, or should not be parsed by any VICO NLU parser at all. To score the parsing results, we counted both (a) parsing results from the parsing of route task input and (b) false positives arising from the parsing of non-route task input. The results are shown in Table 3.3.

Transcribed utterances		**SR output**	
Recall	**Precision**	**Recall**	**Precision**
97,95	96,96	72,44	58,19

Table 3.3. Route task parsing test results.

Table 3.3 shows, we believe, encouraging NLU parser results at the present stage of VICO system development. In particular, the parsing of the transcribed utterances was performed rather well. Analysis of the parsing of SR output showed that the major reason for parser failure was that the SR did not output correct proper names, i.e. of address items, points of interests, hotels, etc.

3.5 Issues

Most of the weights used by the arbitration module have been found empirically using a small corpus of utterances which have been tested independently of the dialogue context. In the next steps in our work, we intend to do further investigation, using the fully integrated VICO system, in order to optimise these weights.

4. The dialogue manager

4.1 Introduction

The goal of the DM is to produce, based on a semantic frame received from the NLU, a high-level semantic specification of appropriate output to the user in all circumstances, independently of: what the user has said, which problems the recogniser and/or the parser have had in recognising resp. understanding the user's input, which problems the DM has had in processing the semantic representation of the user's input, which problems the DB has had in responding to the DM's queries, which task is being addressed by the user, if any, etc.

4.2 Architecture and data flow

The general DM architecture is shown in Figure 4.1.

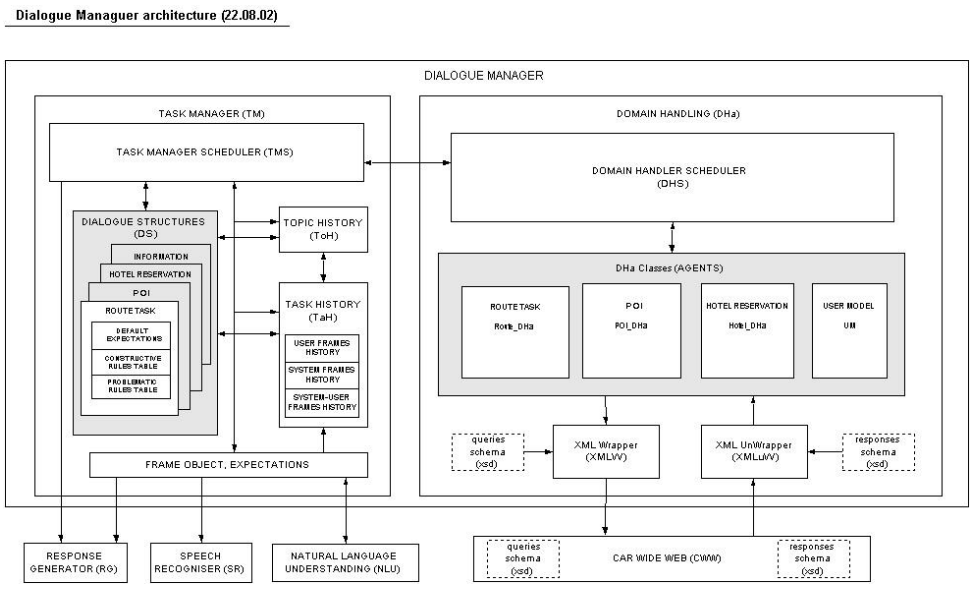


Figure 4.1. VICO dialogue manager architecture.

The task manager scheduler (TMS) manages communication with the other main system modules as well as within the DM.

Communication with the other main system modules involves receiving semantic (user) input frames from the NLU, sending semantic (system) output frames to the RG, sending predictions (or expectations) on the next user input to the speech recogniser (SR) and the NLU, and querying the CWW, all communication being mediated by the SM.

Within the DM, the TMS manages communication with the main DM modules, i.e. the dialogue structures (DSs, called task objects in Figure 4.1), the dialogue histories (DHIs), i.e. the ToH and the DaH, and the domain handler scheduler (DHS). To enable its communication management tasks, the TMS uses a task-independent communication protocol. The DHS manages communication with the domain agents (DAs), one per task which needs communication with the CWW, as well as with the user modelling module (UM). To enable its communication management tasks, the DHS uses a communication protocol.

The DM is separated into two main parts, the task management (TM) part and the domain handling (DHa) part. The basic difference between these two parts is that the TM part communicates, via the TMS, with the SR, NLU, and RG, whereas the DHa part only communicates with the TMS and, when required, with the CWW in order to access task-relevant databases (DBs).

The TM part of the DM includes, in addition to the TMS:

1. a repository of task objects, i.e. the dialogue structures (DSs) for each task supported by VICO; and
2. two different dialogue histories (DHIs), i.e. the topic history (ToH) and the task history (TaH).

The topic history (ToH) keeps information related to each user-system turn, such as number of turns, current active task, response to the current input turn, predictions for the next input turn, status of the dialogue, action to follow if any, etc. The task history (TaH) accumulates information related to the task which is currently being solved. That is, for each user-system turn, the TMS adds to the TaH information provided by the user as well as information retrieved from the CWW. When the DM has decided which output to produce relative to the user's input, that output is being stored in the task history as well.

The TMS communicates with the DHIs several times in each user-system turn cycle. When the DM receives a new user input frame, that frame is being stored in the DHI. When the DHa consults the CWW and new information is added to the frame, that information is stored in the DHI. When the TMS consults the DS, that information is stored in the DHI, in particular in the topic history.

The TMS communicates with the DS once in each user-system turn cycle. This happens when the user's input has been evaluated by the domain handler scheduler (DHS) and, when necessary, by the relevant domain agent (DA). Following the result of the evaluation, the TMS retrieves the appropriate system response-in-context from the DS.

In addition to being stored in the DHI, each new user input semantic frame is sent to the domain handler (DHa). The DHa is controlled by the domain handler scheduler (DHS). The DHS decides if the input needs further processing by a particular DA or if appropriate output can be produced by the DS without any further DHa assistance. In the latter case, the DHS instructs the TMS to retrieve the appropriate input from the DS. In the former case, the DHS forwards the user's input semantic frame to the relevant DA. The DA decides on the task-relevant query to be sent to the DBs via the CWW, wraps the query in XML, unwraps the DB's XML response, processes the response, inserts the processed response into the semantic frame, and returns the frame to the TMS via the DHS. Based on the frame received, and after consultation with the DS, the TMS sends a semantic output frame to the RG and stores the output frame in the DHI.

It should be noted that the user modelling module (UM) in the DHa does *not* communicate with the CWW. This is because it has been decided by the VICO consortium that, for the purpose of the 1st VICO prototype, at least, the UM will maintain its database of models of the individual drivers internally, rather than having the driver models DB located on the CWW. This may be changed in the 2nd VICO prototype to the effect that the driver user models will be stored on the CWW.

4.3 NLU input frame example

See Section 4.10.5 below.

4.4 DM expectations

Dialogue manager expectations (or predictions) concern the contents of the next user input.

From the point of view of the DM, the expectations at a particular point of the dialogue are the set of frames shared by the NLU and the DM for which a particular response or action has being defined in the TMS. These expectations have been defined into three general sets called Global system, Expected, and Non-expected, respectively (see also Figure 4.2).

- Global System frames: for each task defined in the system, we define a frame or a set of frames which, when received from the NLU, trigger (start) the task. These frames are called the default frames for each task.
- Expected frames: these frames are defined for each particular task or task case and for each turn. These are the expectations defined in the DS for each task.
- Non-expected frames: for each particular task case and each turn, this set is dynamically defined as constituting the complement of the previous sets.

For instance, suppose the user starts a route task saying: "I would like to go to Odense". The topic history record (illustrated in Table 4.5) then includes the following active task values for the next user turn in its NumExpected slot: *yes*, *yesroute*, *no*. This means that, taking into account the system's response to the user's input (Response -> "Is there a particular part of city or street you want to go to in Odense Fyn?"), the DM expects that users replies with a yes frame, a yes and/or route frame (yes + extra information), or a no frame.

4.4.1 Expectations from the DM to the SRs

Now, from the point of view of the SR the set of current DM expectations are mapped by the DM into the corresponding speech recogniser units. In the example used in this section, the DM should indicate to the SR that the following units must be activated for the next turn: the SR1 navigation unit and the SR3 unit for handling confirmation and refusal. (cf. [Berton et al., VICO deliverable D12, October 2002]).

4.4.2 Expectations from the DM to the NLU

The current implementation of the NLU basically uses the DM-provided expectations for disambiguation among tasks. So, from the point of view of the NLU, the set of current expectations constitutes the NLU's the knowledge about which particular task, if any, the DM is currently solving. Continuing with the example above, the DM indicates to the NLU that the current active task is a route task. In case of ambiguity among tasks, the NLU uses this information as a factor in deciding to which particular type of frame the next user input may correspond.

4.5 The task manager

4.5.1 Task manager architecture and data flow

The TM data flow is shown in Figure 4.2 and the procedures involved are described in Table 4.1. Procedures 4.1 through 4.6 take into account the TM's expectations concerning the user input received, including the cases in which the user's input is not expected in context, the user's request for repetition, and the cases in which the system returns to a previously interrupted task. Between the states 4.1 through 4.6 and state 5 in Figure 4.2 happens everything to do with the second part of the DM, i.e. the DHa. So, from state 5 in Figure 4.2, the TM procedures concern the user's input frame which has been evaluated by the DHa, starting, in state 5, with the TM's consultation of the dialogue structure (DS).

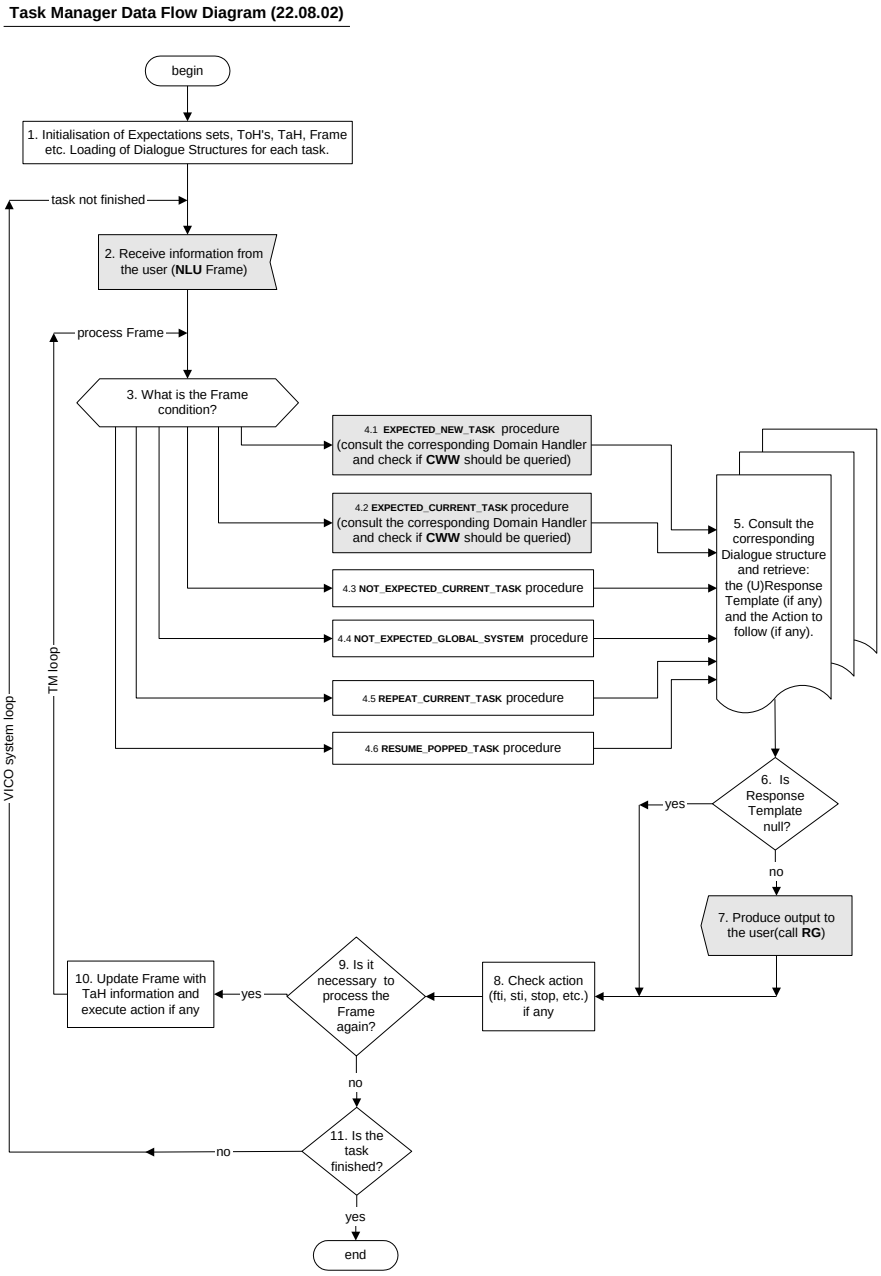


Figure 4.2. Task Manager data flow diagram. The procedures in grey have some external relation with other modules in the VICO system.

No.	Procedure or step in the data flow	Brief description
1	Initialisation of Expectations sets, ToH's, TaH, Frame etc. Loading of	This procedure is done when an instantiation of the DM class is created.

	Dialogue Structures for each task.	
2	Receive information from the user (NLU Frame)	The DM waits until the NLU produces a new frame.
3	Check Frame condition	The frame condition basically depends on the type of frame, if a new task is started it is pushed in the stack.
4.1	EXPECTED_NEW_TASK procedure (consult the corresponding Domain Handler and check if CWW should be queried)	Last frame was one of the expected in the global system so a new task is started.
4.2	EXPECTED_CURRENT_TASK procedure (consult the corresponding Domain Handler and check if CWW should be queried)	Task Manager is currently solving a particular task and the frame is one of the expected in this task.
4.3	NOT_EXPECTED_CURRENT_TASK procedure	Task Manager is currently solving a particular task and last frame is not among the expectations of this task.
4.4	NOT_EXPECTED_GLOBAL_SYSTEM procedure	Last frame is not one of the expected in the global system and Task Manager is not solving any task currently.
4.5	REPEAT_CURRENT_TASK procedure	user ask for repetition in current task
4.6	RESUME_POPPED_TASK procedure	user and system have finished the negotiation and the stack still has a task pending.
5.	Consult the corresponding Dialogue structure and retrieve: the (U)Response Template (if any) and the Action to follow (if any).	The dialogue structure corresponds to the constructive table and the problematic table.
6	Is Response Template null?	in particular points of the dialogue the dialogue structure just contains actions but not responses.
7.	Produce output to the user (call RG)	after retrieving the (U) Response Template from the DS it is sent to the RG.
8	Check action (fti, sti, stop, etc.) if any	in some cases actions and responses are defined in the dialogue structure, so actions are always checked.
9	Is it necessary to process the Frame again?	This is a procedure applied under certain conditions and basically implies that Task Manager does not get any extra NLU frame but process again the current frame using the information in the Task History.
10	Update Frame with TaH information and execute action if any	This procedure is applied if the frame is processed again.
11	Is the task finished?	Task manager finishes when it finishes successfully a task-case, when users says goodbye or when Task Manager could not complete successfully a task-case after four or five problematic turns.

Table 4.1. Brief description of main procedures in the Task Manager flowchart (Figure 4.2).

4.5.2 TM data flow

A finite-state automaton diagram of the TM data flow is shown in Figure 4.3. The diagram is explained in the following Table 4.2.

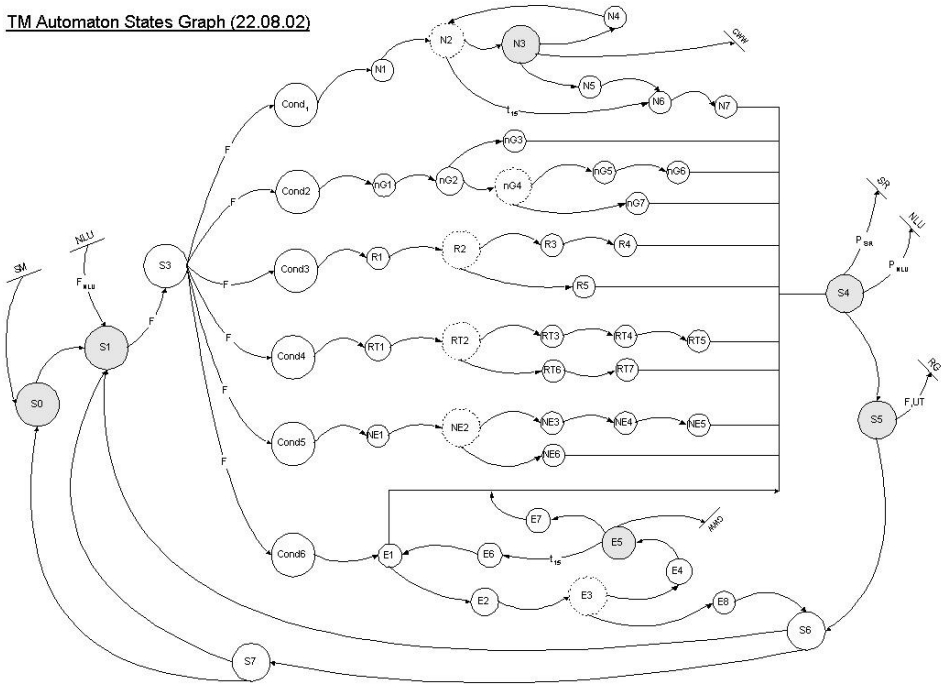


Figure 4.3. Task Manager finite-state automaton diagram.

No	State Name	State Description
1	**S0**	Initialisation
2	**S1**	Ready to receive a new frame
3	**S3**	Frame condition identified
4	**Cond1**	New task started
5	**Cond2**	Frame not expected in the global system
6	**Cond3**	Repetition
7	**Cond4**	Resuming task
8	**Cond5**	Frame not expected current task
9	**Cond6**	Frame expected current task
10	**S4**	Sending predictions to the SR, and NLU
11	**S5**	Sending output response to the RG (Frame + Unfilled template)
12	**S6**	Executing action

13	**S7**	Checking task status and tasks stack	
14	**Cond1**	**N1**	Creating a new TaH and a new ToH
		N2	Update TaH and ToH with information in the Frame
		N3	Consult Domain Handler Scheduler
		N4	CWW was consulted
		N5	CWW was not consulted
		N6	Consult Dialogue Structure corresponding to the Active Task
		N7	Update ToH slots with information from the Dialogue Structure
15	**Cond2**	**nG1**	Update ToH slots with task status and information in the frame
		nG2	Checking if there is an Active Task
		nG3	No Active Task, so set output "how may I help you"
		nG4	There is an Active Task, so consult corresponding DS
		nG5	Check if Action is stop
		nG6	Set output "good bye"
		nG7	Update ToH slots with information from the Dialogue Structure
16	**Cond3**	**R1**	Update ToH slots with task status and information in the frame
		R2	Consult corresponding Dialogue Structure
		R3	Check if Action is continue
		R4	Get last output from current task TaH
		R5	Update ToH slots with information from the Dialogue Structure
17	**Cond4**	**RT1**	Update ToH slots with task status and information in the frame
		RT2	Consult corresponding Dialogue Structure
		RT3	Check if Action is continue
		RT4	Update frame with the information in current task TaH
		RT5	Get last output from current task TaH
		RT6	Check if Action is stop
		RT7	Set output "good bye"
18	**Cond5**	**NE1**	Update ToH slots with task status and information in the frame
		NE2	Consult corresponding Dialogue Structure
		NE3	Check if number of problematic turns is 3 or status equal end
		NE4	Delete ToH of the active task and TaH content
		NE5	Set output "good bye"
		NE6	Update ToH slots with information from the Dialogue Structure
19	**Cond6**	**E1**	Update current task TaH and ToH with information in the Frame
		E2	Update current frame with information in the SUTaH
		E3	Consult Dialogue Structure corresponding to the Active Task
		E4	Response from the DS is not null
		E5	Consult Domain Handler Scheduler
		E6	Frame updated with information from the CWW
		E7	Update ToH slots with information from the Dialogue Structure

		E8	Action from the DS is not null

Table 4.2. Brief description of main procedures in the Task Manager finite-state automaton diagram (Figure 4.3).

4.6 Dialogue structure example

We illustrate the DS by an example from the route task. Structurally, the DS is a tree or a directed acyclic graph.

Formally, the route task involves the following possible route *cases:*

F1: Input route frame empty

F2: Input frame has a lexical ambiguity

F3: Input frame has a type mismatch

R1: Input items unique and complete according to DB

R2: Input items unique and incomplete according to DB

R3: Input items non-unique (2 or 3 alternatives)

R4: Too many locations (> 3)

R5: Input items inconsistent

We use the term *cases* to denote formally distinct, higher-order types of user input. The set of cases pertaining to a particular task are identified during specification and design analysis. The distinction between F-type cases and R-type cases is that F-type cases are identified already by the NLU whereas the R-type cases only become identified following DB evaluation of the user's input and subsequent DA processing. The DA processing issues in the insertion into the frame of case values based on the DB's response (see below and Tables 4.4 and 4.5).

For each route case there is a rules table which is organised per confidence score (1,2,3 – 3 being the highest score), per DM expectations set (i.e. the expectations to the next user input which the DM sends to the SR and the NLU), and per turn. The table for the R1 case is shown in Table 4.3. For simplicity, we include only two user turns and two system turns. As shown in Table 4.3, for each system turn and confidence score, at least one *action* is defined in terms of an unfilled response template, or *URTemplate* which is the response the VICO system should output, and a set of *expectations* concerning the next user input. In Table 4.3, URTs are referred to by numbers, possibly including sets of attribute-values called [items]. Additional actions may be defined for the user's input, as appropriate (cf. Step 8 in Figure 4.2). Actions enable the TMS to execute sophisticated procedures when processing the user's input in context, such as making the system's output at a particular node in the DS tree conditional on the result of checking a particular attribute-value in the input frame.

U(1)→ S	**S(1)→U**	**U(2)→S**	**S(2)→U**	**U(3)**	**S(3)**
R1: route	Action: null CS=3 URTemplate: <12: [items], 13> Expectations:	confirmation frame and/or end frame CS=3	Action: stops further negotiation, this is a second-time route input URTemplate: null Expectations:null		

	confirmation, end, negation + new items	confirmation frame and/or end frame CS=2,1	Action: null URTemplate: <37> Expectations: confirmation, end, negation + new items		
		negation frame+new items CS = 3,2,1?	Action: as first-time input (nfti) URTemplate: null Expectations:null		
	Action: null CS=2 URTemplate: <2: [items]> Expectations: confirmation, negation + new items	confirmation frame CS=3,2,1	Action:as first-time input (fti) URTemplate: null Expectations:null		
		negation frame + new items CS=3,2,1	Action:as first-time input (nfti) URTemplate: null Expectations:null		
	Action: null CS=1 URTemplate: <6, 5> Expectations: route items	R1-R6 CS=3,2,1	Action:as first-time input (fti) URTemplate: null Expectations:null		

Table 4.3. Route Task Case R1: input items unique and complete according to the DB.

If the user says, for instance: "I would like to go to Cikorievej twenty.", and supposing that the confidence score = 3, then the NLU sends to the DM the following NLU frame (Table 4.4):

Type	**route**
Topic	
PartOfCountry	
City	
PartOfCity	
Street	Cikorievej
Number	20
ConfScore	3
TypeMismatch	0
LexAmbiguity	0
Unique	-1
Complete	-1
NonUniqueMax3	-1
NonUniqueMore3	-1

NonExistent	-1
Inconsistent	-1
StreetNumbersDB	-1

Table 4.4. Route task input frame example.

The TM receives the frame in Table 4.4 and, since it is a route frame and it is the first input from the user, the TM starts a ROUTE_TASK, and pushes it onto the stack:

TASKS STACK: [ROUTE_TASK]

As the NLU did not report an F-type case, the TM checks with the Domain Handler, which should consult the CWW, if the input is consistent, complete, etc. The DHa returns the frame shown in Table 4.5.

Type	**route**
Topic	
PartOfCountry	Fyn
City	Odense
PartOfCity	SE
Street	Cikorievej
Number	20
ConfScore	3
TypeMismatch	0
LexAmbiguity	0
Unique	1
Complete	1
NonUniqueMax3	-1
NonUniqueMore3	-1
NonExistent	-1
Inconsistent	-1
StreetNumbersDB	1

Table 4.5. Domain handler return wrt. the frame in Table 4.4.

Table 4.5 shows that the DHa, after querying the CWW, has completed the user's information about city, part of city, etc., and has also evaluated whether the information is consistent, unique and complete. The TM checks this information and concludes that the case is R1 (the information is unique and complete). Now the TM consults the corresponding R1 case in the ROUTE_TASK table and, since this is first user turn and the confidence score is 3, the TM retrieves from the table's R1 case the corresponding *Action, URTemplate* (unfilled response template, URT) and the set of *Expectations*. This information is kept for each turn in the topic history (ToHI). The first record of the topic history is shown in Table 4.6.

Frame slots	**Values**
ActiveTask	ROUTE_TASK
Turn	1
FrameType	route

CS	3
Case	R1
Status	ini
URTemplate	12:[non-empty-items],13
Action	NULL
NumExpected	2 -> yesend, no+route

Table 4.6. Topic history record.

At this point, the TM knows which URT should be sent to the RG in order to produce the output. In the present example the URT is "**12:[non-empty-items],13**". The URTs, specified in the DS tables for each task, are defined in terms of sentence templates. An URT consists of one or several sentence templates. The sentence templates in the URT are coded using unique identification numbers and labels between square brackets for the slots (attributes) that should be filled (be assigned values).

In the present example, the UR consists of the following sentence templates:

12:[non-empty-items] which corresponds to "You want to go to **[non-empty-items]**."

13 which corresponds to "The route is being planned".

The present example is continued in Section 5.2.

4.7 Dialogue history example

4.7.1 Topic history

An example of a topic history record is shown in Table 4.6. The task manager creates a new topic history record each time it receives a new user turn or when the action retrieved from the DS indicates that this should be done. The slots are filled by the TM during the solving of a task, using, among others, the information obtained after consulting the DS of the corresponding task. Since the ToH data type is a list of linked records, one per user turn, the ToH is used by the TM to consult the dialogue structures as well. That is, in order to retrieve from the DS the corresponding response, expectations, and/or action to follow in the next turn, it is necessary to consult a record of what was said by the user and by the system in the previous turns. In order to do that, the TM uses the ToH list of records as a guide to walk through the different nodes of the DS and retrieve the correct information (response, action, and/or expectations).

4.7.2 Task history

The slots and default values defined for the task history are shown in Table 4.7.

Slot	Brief description
Turn	current user turn
ActiveTask	task being solved
Case	task-case being solved (if any)
Uframe	user information, for example addresses, poi names, etc
Sframe	system information, for example addresses, poi names, etc. and information completed after consulting the domain handler
SUframe	combined user and system information
LastOutput	last output produced by the system in the current task

Table 4.7. Task history slots and default values.

Uframe, Sframe and SUframe are data structures of type frame. Their contents are explained in Section 4.6. The TM creates a new TaH history record each time a new task is started. Each time the TM receives a new frame from the user, the information is filled into the Uframe data structure as well as into the SUframe. Each time the Task Manager consults the DHa, the information retrieved from the CWW is filled into the Sframe and into SUframe as well, so that the TM always works with the information in the SUframe unless it has to backtrack to the last information provided by the user. The TaH is deleted after three turns of communication problems, that is, if the TM is solving a particular task and it does not receive an expected frame after three turns, the TM cleans all the slots of the current task and tries to start the dialogue from scratch. If the communication problems persist, eventually the task is popped form the stack and the task is completely deleted. The following is an example of how frame slots are filled during the solving of a particular task:

INPUT User(1): "I want to go to Campusvej number twenty." Recogniser error: the user actually said Cikorievej.

The TaH contents are shown in Tables 4.8 and 4.9.

Turn	1
ActiveTask	ROUTE_TASK
Case	R5
Uframe	(see next table)
Sframe	(see next table)
Suframe	(see next table)
LastOutput	14:[empty-subtypes][non-empty-sub-items]

Table 4.8. Task history contents. See also Table 4.9.

	Uframe	**Sframe**	**SU frame**
Type	route	route	route
PartOfCountry		Fyn	Fyn
City		Odense	Odense
PartOfCity		SE	SE
Street	Campusvej		Campusvej
Number	20		20
ConfScore	2	2	2
TypeMismatch	0	0	0
LexAmbiguity	0	0	0
Unique	-1	-1	1
Complete	-1	-1	1
NonUniqueMax3	-1	-1	-1
NonUniqueMore3	-1	-1	-1
NonExistent	-1	1	1
Inconsistent	-1	-1	-1
StreetNumbersDB	-1	-1	-1

Table 4.9. Task history contents example. See also Table 4.8.

OUTPUT System(1): "There is no street number 20 in Campusvej in Odense SE Fyn."
INPUT User(2): "No, I said Cikorievej."
The TaH contents are shown in Tables 4.10 and 4.11.

Turn	2
ActiveTask	ROUTE_TASK
Case	R1
Uframe	(see next table)
Sframe	(see next table)
SUframe	(see next table)
LastOutput	12:[non-empty-items],13

Table 4.10. Task history. See also Table 4.11.

	Uframe	**Sframe**	**SU frame**
Type	route	route	route
PartOfCountry		Fyn	Fyn
City		Odense	Odense
PartOfCity		SE	SE
Street	Cikorievej		Cikorievej
Number	20		20
ConfScore	3	3	3
TypeMismatch	0	0	0
LexAmbiguity	0	0	0
Unique	-1	1	1
Complete	-1	1	1
NonUniqueMax3	-1	-1	-1
NonUniqueMore3	-1	-1	-1
NonExistent	-1	-1	-1
Inconsistent	-1	-1	-1
StreetNumbersDB	-1	1	1

Table 4.11. Task history, cf. Table 4.10.

OUTPUT System(2): "You want to go to Cikorievej 20 in Odense SE Fyn." The route is being planned.

4.8 Domain handler scheduler

4.8.1 Domain handler scheduler architecture and data flow

The main tasks of the DHS is to (i) re-direct the frame to the corresponding domain agent and to (ii) sometimes simply return the frame because its contents do not require DB verification or the frame does not fulfil the rule conditions of a particular DA. As illustrated in Figure 4.3, the DHS is consulted in two particular states of the TM automaton diagram: N3 and E5,

corresponding to the conditions new task started and frame expected in the current task, respectively. None of the other conditions, such as repeat or resume, require any consultation with the DHS.

4.8.2 Domain handler scheduler rule example

In order to make a query in the route task, the following rule conditions must be checked in the DHS:

- the type of frame is route;
- the confidence score is 3;
- the task case is different from F1 (input route frame empty), F2 (input frame has a lexical ambiguity), or F3 (input frame has a type mismatch and confidence score is less than 3);
- check if in the destination provided by the user there is missing a city name;
- check if in the destination provided by the user there is missing a street name.

4.9 Domain agent example

Figure 4.4 shows the main modules and the data flow among these in the hotel reservation domain agent, or HDA (cf. Figure 4.1). The HDA is controlled by the central unit module which interacts with all other modules in the domain agent. When the HDA receives a frame as input from the TM, it first checks the contents and status of information in the frame. If the frame includes information which needs further checking, then the central unit consults one or several of the modules Stars, Price, Time unit, Number of Persons, and Room type to check if the information provided by the user is correct.

If, for example, the user says "I want a 3-star hotel in the city centre of Trento", and if this sentence is correctly recognised and parsed, then the hotel domain agent will receive a frame which contains filled slots for stars (3), location (city centre) and city (Trento). The number of stars must be checked to ensure that the number is between one and five. For this purpose, the central unit calls the stars module. In the present case everything is all right. However, if, e.g., the user had asked for a 6-star hotel, the stars module would have returned an error. Whenever an error occurs, further HDA processing is stopped by the central unit which then returns a frame to the TM. The returned frame included slots which report on all possible errors.

Returning to our example, processing continues by the central unit since no errors were discovered by the stars module. So far, no additional correctness checking is needed given the contents of the user's input. So, the next step is to send a query to the CWW to find out if the user's input is sufficient to uniquely identify a hotel. To do this, the central unit contacts the XML wrapper by sending it the frame with the current information plus information about the kind of query to be made to the CWW. The XML wrapper wraps the relevant information in XML and returns the query to the central unit which sends it to the CWW via the SM.

The CWW's return is wrapped in XML. The response is sent to the central unit which sends it to the XML unwrapper. The XML unwrapper unwraps the information and inserts it into the frame which is then returned to the central unit. There are basically four possibilities wrt. the nature of the returned information:

1. no hotel exists which satisfies the constraints;
2. exactly one hotel satisfies the constraints;
3. two or three hotels satisfy the constraints;
4. more than three hotels satisfy the constraints.

The central unit decides what to do depending on which of the four cases obtain in the returned frame. For all cases, the case number is included in the frame.

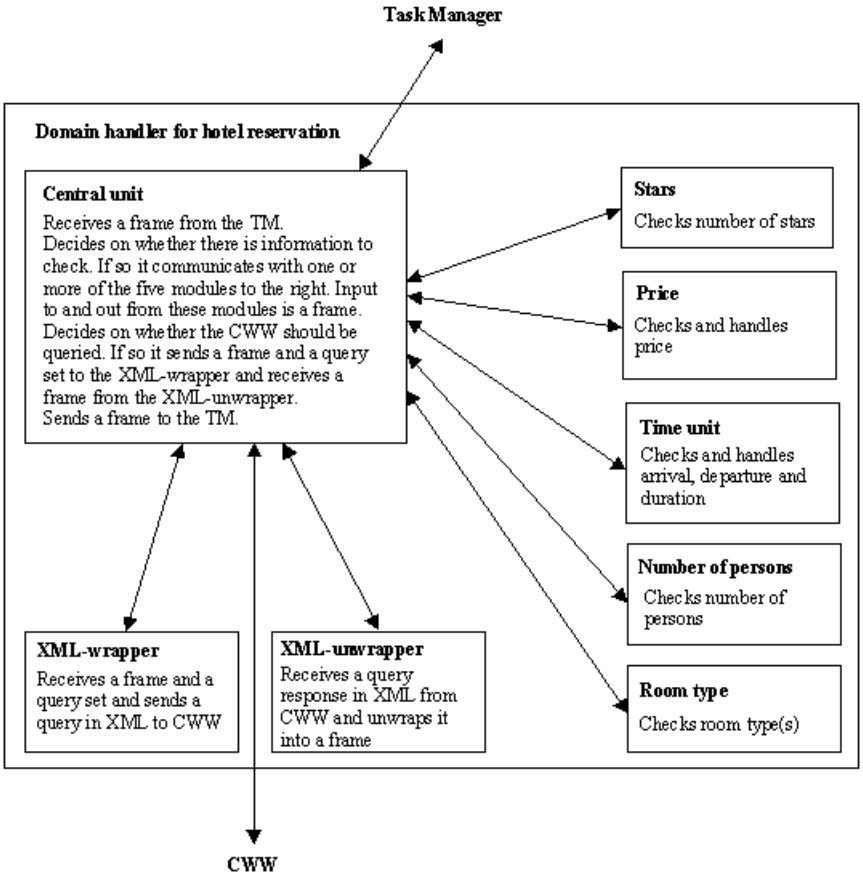


Figure 4.4. The hotel reservation domain agent architecture and data flow.

In case 1, the central unit returns the frame to the TM. The frame now includes the information that no hotel was found which satisfies the constraints provided by the user.

In case 2, the central unit checks if there is already sufficient information available in the frame to also make a reservation query. If yes, then a new query is wrapped and sent to the CWW. If no, the frame is returned to the TM.

In case 3, the frame is just returned to the TM.

In case 4, the central unit checks if there is useful information available from the user modelling module, cf. Section 4.12. If no, then the frame is just returned to the TM. If yes, the user model information is added to the already available information and a new query is sent to the CWW in the hope that the additional constraints will bring the matching number of hotels down to at most three. Unless the new CWW return leads to case 2 or 3, the first result is used and the frame reporting more than three hotels is returned to the TM.

The XML wrapper and XML unwrapper are described in more detail in Section 4.12.

4.10 Point of interest task

The POI task is specified in the two NISLab internal documents [Bernsen 2002a] and [Bernsen 2002b].

4.10.1 POI types and user requests for particular POIs

The left-hand column of Table 4.12 shows the 25 POI types which drivers can be helped by VICO to navigate to. The following five columns of Table 4.12 show, for each POI type, the information which the driver can provide in order to indicate a particular POI goal or which VICO can use to uniquely identify a particular POI goal in its response to the user. Based on an early, preliminary consortium decision on which POI types to include in the 1st VICO prototype, the final selection of POI types and their properties have been negotiated with ITC-Irst based on their knowledge of the APT Trentino tourist database and the TeleAtlas database for the Trento province.

POI TYPE	NAME	SUPER NAME	NEAREST [TYPE]	ADDRESS ITEMS	NEAREST [SUPER NAME]
airport	x	-	x	x	
camping ground	x	-	x	x	
car repair	x	-	x	x	
cash dispenser/bank	-	x	x	x	x
castle	x	-	x	x	
church	x	-	x	x	
cinema	x	-	x	x	
college/university	x	x	x	x	x
disco	x	-	x	x	
doctor	x	-	x	x	
hospital/polyclinic	x	-	x	x	
hotel/motel	x	x	x	x	x
ice-skating rink	x	-	x	x	
mountain pass	x	-	x	x	
museum	x	-	x	x	
parking garage	x	-	x	x	
petrol station	x	-	x	x	
pharmacy	x	-	x	x	
post office	x	-	x	x	
railway station	x	-	x	x	
restaurant	x	-	x	x	
stadium	x	-	x	x	
swimming pool	x	-	x	x	
theatre	x	-	x	x	
winery	x	-	x	x	

Table 4.12. What users can input wrt. points of interest.

It appears from Table 4.12 that virtually all POI entities have a name, such as doctor Zampolli's car repair shop or a disco named Casanova. The exception is cash dispensers and the banks which have them. These do not have names in the APT and TeleAtlas DBs but only

have what we call super names, such as Banco di Trentino. Hotels have names, such as Buonconsiglio, and sometimes also super names, such as Hilton. Also, the university of Trento has several names belonging to its different faculties together with the super name of the university administration building itself. For all POI types, the driver can ask to go to the nearest one, such as the nearest restaurant. Moreover, drivers can ask to go to the nearest super named POI, such as the nearest Banco di Trentino. Finally, drivers may at any time include address items in their request for a particular POI. These address items are the same as those used in the route task, i.e. part of country, city, part of city, street, and street number. In trying to uniquely identify the POI entity which the driver asks for, VICO will add what may be the missing information on that POI from querying the DBs.

In the NLU lexicon, all named and super named POI entities are tagged with their POI type, as are, of course, all identified linguistic equivalents for the POI types. This strongly facilitates the reasoning task of the dialogue manager. In addition, the tagging makes it easy for the NLU to decide when a driver starts addressing the POI task.

4.10.2 POI input combinatorics and DB queries

As can be seen from Table 4.12, the possible combinatorics of user input POI information input is somewhat complex. Before describing the POI standard input, let us look at the error cases which may occur.

In addition to handling the POI standard input cases, the DM must be able to handle at least four error cases up front, i.e. (a) the reception of an empty POI frame; (b) the reception of a lexically ambiguous POI frame in which, for instance, the NLU has tagged the POI name Casanova as referring to both a disco and a hotel; and (c) the reception of a type mismatched POI frame in which, for instance, the user claims that the POI named Casanova is a pharmacy whereas the NLU has the name Casanova tagged as a disco and a hotel but not as a disco. Lexically ambiguous and type mismatched POI frames may also include errors arising from the address items proposed by the driver. These errors are handled in the same way as for the route task, cf. [Bernsen et al., VICO deliverable D9, March 2002]. A fourth error case (d) is also found in the route task as well [ibid.]. In this case, the DM receives partly useless address items, such as a named part of country followed by a street number. As it hardly makes sense to search for, e.g., street number 5 in the entire Trento province, such useless address items are removed by an address items pre-processor before querying the DBs. In the error cases of an empty POI frame (a) and lexical ambiguity (b), the DM similarly sorts out the error with the driver without DB query. The type mismatch error case (c), on the other hand, does require a DB query.

For the non-error, standard POI input cases, the driver may provide any combination of values for the following attributes, always including the POI type which is what makes the driver's input POI input:

1. POI **type** (e.g. petrol station)
2. POI **name** (e.g. Buonconsiglio), *not for cash dispensers/banks*
3. POI **super name** (e.g. Hilton)
4. **nearest** [POI type, e.g. hotel]
5. POI **address items** (e.g. Via Roma 10, Trento)
6. **nearest** [POI **super name]**

Even if the input case combinatorics is a bit large, design analysis has shown that, through pre-processing, the cases can be narrowed down to 8 different DB queries, i.e.:

1. Search nearest (to the car's position) on: nearest + type
2. Search locally (relative to the car's position) on: type + name
3. Do local or global search on: type + address items

4. Do local or global search on: type + name + address items
5. Search nearest (to the car's position) on: nearest + type + super name
6. Search locally (relative to the car's position) on: type + name + super name
7. Do local or global search on: type + super name + address items
8. Do local or global search on: type + name + super name + address items

The POI domain agent (DA_POI) decides whether to do local or global search depending on whether or not the driver has provided a city and/or a part of country for the POI in question. In the absence of city and/or part of country, a local search is performed by limiting the DB search to the area around the car's current position. The nearest queries always return a single POI entity. In a global search, the entire DB is searched.

4.10.3 POI DB returns analysis

The DB query, serving the purpose of user input validation, may produce a series of new error cases. In particular, we need to distinguish between 9 different types of inconsistency flags which may be received from the DB, as follows:

1. no city in part of country
2. no part of city in city
3. no street in city
4. no street in part of city
5. no number in street
6. no type-item in address item(s)
7. no type-item of super name in address item(s)
8. no type-item of name and super name in address item(s)
9. no type-item of name in address item(s)

In addition, the DB returns may include information to the effect that the DBs do not have street numbers, or even street names, for a particular POI. It is important that the driver be told about such deficits.

Otherwise, DB returns can be systematised into the following non-error cases. The queried POI may be:

1. unique and complete according to the DBs
2. non-unique (2 or 3 alternatives found)
3. too many POI items (> 3)

The id of the query, i.e. one of the eight DB queries described in Section 4.10.3 above, is inserted into the frame by the XML unwrapper together with the information items (attribute values) retrieved from DBs. Adding the query id to the frame enables VICO to provide feedback to the driver which assures the driver that VICO has understood the driver's request correctly. The resulting frame is sent by the DA_POI to the TMS which then consults the dialogue structure (DS) in order to retrieve the unfilled template to be used in responding to the driver.

4.10.4 POI dialogue structure

The POI dialogue structure is a comprehensive table which determines the semantics of the system's output for every possible input POI frame as processed by the DHa, cf. the DS example in Table 4.3. The TMS uses a task-independent procedure for looking up the appropriate semantic output in the DS given the input confidence score, the frame's POI-relevant values, and the query id reported by the DHa. To make this possible, all task DSs are structured in the same way, i.e. according to input confidence score, input error/standard case id, and the evolving dialogue context. As will be explained in Chapter 5, in its output to the

RG, the TMS/DS only includes the semantics of the output in the form of strings of output sentence references (numbers) and output sentence frame attributes. The corresponding, actual output values are extracted from the corresponding semantic frame by the RG.

4.10.5 POI use case

This section presents a POI task use case. User input and CWW return is shown in Table 4.13.

User1: Can you find the next campsite?

	NLU output	**TM**	**CWW**
Frame type	poi	poi	poi
POI type	camping ground	camping ground	camping ground
POI name			Trento Municipal
POI super name			
nearest **POI type**	true	true	true
nearest **POI super name**			
Part of country			Trentino
City			Canazei
Part of city			
Street			Via Pordoi
Street number			
TypeMismatch	false	false	false
LexAmbiguity	false	false	false
Conf. Score	3	3	3
unique			true
NonUniqueMax3			
NonUniqueMore 3			
Inconsistent			
StreetNumbersDB			false
POI StreetsDB			true

Table 4.13. Point of interest task user input and CWW return (Use Case 1).

We have to do a Q1 query to the CWW. Query1:

<poi_query query_id="Q1" local_assumption="**yes**" nearest="**yes**">
<poi_items poi_type="**camping_ground**">
</poi_query>

Response:

<poi_query_response results="**1**"/>
<poi_items poi_type="**camping_ground**" poi_name="**Trento Municipal**" poi_supername=" " />
<position part_of_country="**Trentino**" city="**Canazei**" street="**Via Pordoi**" />

</poi_query_response>
The Topic History record is:
ActiveTask: POI
Turn(User): 1
FrameType: poi
CS: 3
Case : POI CASE 1, Q1
Status:
(U)Response Template: <28>The nearest [insert POI type] is [insert POI type] / optional: [insert POI name] / optional: [insert POI super name] / in [insert part of country], [insert city] / optional: [insert part of city] / optional: [insert street] / optional: [insert street number] /<12> The route is being planned.
Action:
NumExpected (Expectations): -> yes and /or end frame
RG: (F)Response Template -> The nearest camping ground is camping ground Trento Municipal in Trentino, Canazei Via Pordoi. The route is being planned.
TASKS STACK: [POI_TASK]

4.11 Hotel reservation and restaurant reservation tasks

The hotel reservation task and the restaurant reservation task are specified in a series of NISLab internal documents [Dybkjær 2002].

The hotel reservation task and the restaurant reservation task are rather different from the route task and the information task (see [Bernsen et al., VICO deliverable D9, March 2002]), and the POI task described in Section 4.10 above. In particular, the hotel reservation task is a quite comprehensive one; it has several phases, starting with the hotel selection phase and ending with the hotel reservation phase proper, but the driver can address any phase at any time; compared to the route, information, and POI tasks, the hotel reservation dialogue structure requires more complex dynamical updating in context in order to compute the output to the driver; and the hotel reservation task incorporates on-line observation-based user modelling of the driver's hotel preferences (see Section 4.12), increasing the functional complexity of the task even further.

The hotel reservation task and the restaurant reservation task have thus served as the first real tests of the generality of our dialogue manager. We were keen to see how straightforward it would be to specify and implement those two new tasks for handling by the would-be general DM architecture and data flow arrived at during the spring of 2002. In particular, the challenge posed by the new tasks concern the TM part of the DM (see Section 4.5) whereas the DHa part of the hotel reservation task is, of course, entirely ask dependent. As it has turned out, the hotel reservation task and the restaurant reservation task can be straightforwardly implemented in the DM architecture. A crucial role in making this possible is assumed by the action functionality embodied in the task-specific dialogue structures (cf. Figure 4.2). As remarked in Section 1.3, the consortium has decided to closely connect the restaurant reservation task to the hotel reservation task since it does not add much new to our knowledge of task handling to implement a full and independent version of restaurant reservation when we have already done hotel reservation.

4.11.1 User input for the hotel task

The reservation of a hotel room basically consists in first selecting a hotel and then booking one or several rooms in the selected hotel if rooms are available for the chosen period of time.

The information items available for hotel selection and room reservation have been negotiated with ITC-Irst based on their knowledge of the APT Trentino tourist database and the TeleAtlas database for the Trento province.

The list in Figure 4.5 shows the information items which the driver may combine almost at will in the hotel selection phase of the dialogue with VICO. As a minimum, either an overnight stay type, such as *hotel*, or an overnight stay name must be indicated when the driver wants to initiate a hotel reservation task. The driver may ask for any combination of services (restaurant, protected parking) and/or any combination of properties (chain, number of stars, price, location), and the user may add any address items as well. If no part of country and/or city address items are indicated, the system will assume that the user wants to book a local hotel relative to the car's current position. Note that the address items are the same as for the route and POI tasks.

Name of hotel
Type, this is by default "hotel"
Address items
part of country
city
part of city
street
street number
Properties
hotel chain, e.g. Hilton
stars, between 1 and 5
price, i.e. maximum price for a single and a double room
location, i.e. city centre, outskirts of city, and country side
Services
restaurant available
protected parking available
Additional information (only when a specific hotel is identified)
other services
telephone number

Figure 4.5. Items that may be specified in the selection phase of a hotel reservation dialogue.

A major difficulty in designing appropriate dialogue for the hotel selection task is that there are plenty of services which a user may be interested in. The term 'service' is used in a broad sense in this document, meaning something in, or nearby, the hotel and which the user might want to make use of when staying in the hotel. Some examples are: swimming pool, ski lift, river, sauna, non-smoking rooms, in-room safe deposit box, pets accepted, or mini-bar. The problem, however, is that there are so many services in which at least some users may be interested (probably more than 50) that we decided to just include two key services in the list of items which the user can explicitly ask for, i.e. restaurant and protected parking. Thus, it is not considered the task of the VICO project to demonstrate that we can enable drivers to specify any possible set of services desired when carrying out hotel selection. Rather, it is considered important to demonstrate that it is feasible to carry out meaningful and useful hotel reservation dialogues with VICO and make reservations over the Car Wide Web (CWW).

In addition to the two services mentioned, the driver may obtain from VICO a full list of the services registered for a particular hotel in the database. Finally, the user may ask for the phone number of a specific hotel. By phoning the hotel, the driver may clarify any outstanding issues before booking rooms there.

Once the driver has selected a particular hotel, room reservation can start. To book one or more rooms, the list of items in Figure 4.6 must be specified. In contrast with the information items in Figure 4.5, all of the first five information items in Figure 4.6 must be provided by the user. The only exception is that the system is itself able to calculate one of the three information items *arrival*, *departure*, and *duration* given that it already has two of them. We assume that the credit card number must be provided in order to confirm a reservation. Once the system has received the first five information items in Figure 4.6, it makes a CWW query to find out if the requested room(s) is/are available. If it turns out that the rooms asked for by the user are not free during the indicated period, the system is prepared to handle a new reservation dialogue which re-uses the relevant parts of the information which was already provided by the driver.

Type of room (single/double)
Number of persons
Arrival date
Departure date
Number of nights
Credit card number

Figure 4.6. Items that are needed in the room reservation phase of a hotel reservation dialogue.

4.11.2 User input for the restaurant reservation task

When the driver has booked one or more rooms in a hotel, the driver is allowed to book a table in the restaurant of the hotel, provided that there is a restaurant in the hotel of choice. Three information items are required from the user in order to book a table. These are listed in Figure 4.7. All three items must be provided in order for the DM to make a query to the CWW.

Number of persons
Date
Time

Figure 4.7. Items that are needed in the restaurant reservation dialogue.

4.11.3 Hotel and restaurant input combinatorics and DB queries

Given the number of hotel selection items in Figure 4.5 and the fact that these items may be combined almost arbitrarily, the input combinatorics is quite large. Moreover, even if we want to keep hotel selection and hotel reservation apart as a matter of system design, we cannot prevent the user from mixing, in the input to the system, hotel selection items and hotel reservation items, which increases the input combinatorics that must be handled even more.

As for the route and POI tasks, the DM must be able to handle an empty input frame as well as cases of ambiguity and type mismatch identified by the NLU. If none of these error cases are present in the user's input, the DM goes on to processing the input contents proper. For many information items, this means that these must be checked for existence, completeness,

and/or consistency. Examples of the non existence of user-indicated information items are “a 6-star hotel” and “30 February”. The issue of completeness relates to dates information. A date is incomplete if only the month and/or the year has been provided. Issues of consistency may arise as soon as at least two information items have been provided. For instance, the arrival date provided by the driver may be before today, or the day of the week and the date indicated may not match one another. The DM must check for all such potential errors in the user’s input and resolve them through dialogue with the driver before a query can be sent to the CWW.

Another situation that must be checked for before a query can be sent to the CWW is whether the DM has the necessary information for making a query. For the selection part of the dialogue, a query can be sent as soon as the DM has one of the information items *type* or *name*, cf. Figure 4.5. Since the type is “hotel” by default, this means that a selection query can be sent to the CWW as soon as the DM receives a hotel reservation frame from the NLU without erroneous input. For the reservation phase of the dialogue, the case is different, however. First of all, a hotel must already have been uniquely identified and selected by the user in order for reservation to begin. Secondly, the DM must have received all of the first five information items listed in Figure 4.6, and these must be free of error. Only then can a reservation query be sent to the CWW. In order for a restaurant query to be sent to the CWW, the user must have booked a room successfully all three information items in Figure 4.7 must have been provided, and the items must have been verified to be error-free.

4.11.4 Hotel and restaurant DB returns analysis

As described in Section 4.10, the queries and responses exchanged between the DAs and the CWW are wrapped in XML. Also, the overall flow in handling returns from the CWW is described in Section 4.10 and will not be repeated here since the restaurant and reservation tasks follow the same procedures.

An return from the CWW to a hotel selection query can be categorised as belonging to one of the following four cases:

1. no hotel satisfies the constraints;
2. exactly one hotel satisfies the constraints (unique);
3. two or three hotels satisfy the constraints (non-unique max 3);
4. more than three hotels satisfy the constraints (non-unique > 3).

In Case 1, the query may be over-constrained, such as when the driver asks for a 5-star hotel in a small town which only has a single hotel which is, say, a 3-star one. It may also be that, e.g., some of the address items provided are inconsistent, resulting in a CWW return to the effect that there is no hotel which satisfies the constraints provided.

If the CWW returns a Case 4 reply and if there is a user model of the hotel preferences of this particular driver, the DM will try to use this information to further constrain the DB search and make a new query to the CWW, see also Section 4.12 on user modelling.

For room reservation queries, the CWW may return that, either

1. the hotel is fully booked; or
2. the desired rooms are available.

When the rooms are available and the credit card number is provided, the CWW will return a reservation confirmation.

For restaurant reservation, the CWW will reply if the desired table is available or not.

4.11.5 Hotel and restaurant reservation dialogue structure

The hotel and restaurant reservation dialogue structure (DS) may be viewed as a comprehensive table which determines the system’s output relative to all possible input in

context. It is the state of the DM and the input confidence score that jointly determine which entry in the table to visit. The states again depend on the user's input as do the confidence scores. For all other parts of the DM, the same internal processes are followed as for the other tasks so far implemented in VICO. Naturally, there are important differences across tasks, for instance concerning the particular task-dependent actions which must be taken at a given point in the dialogue. However, the TM's handling of different actions follow the same procedures (cf. Figure 4.2).

4.12 The user modelling module

4.12.1 Deciding what to model for the 1st VICO prototype

Observation-based user modelling represents a new challenge to spoken language dialogue systems research. In observation-based user modelling, the system builds models on-line of its individual users and uses the model information on-line to support user-adaptive performance. Thus, observation-based user modelling is very different from *design-time user modelling* which produces a hard-coded version of the model information arrived at either in the form of (a) a user-friendly dialogue structure which includes graceful degradation of the dialogue in the face of repeated error; or (b) enabled user customisation of the interface prior to interaction. In *observation-based user modelling,* the model of the individual user is being created on-line rather than at design time. Arguably, observation-based user modelling is also the harder to do successfully among the options just described. Evidence of this claim is the fact that the story of the software and systems world during the last 20 years or so is replete with failed attempts at creating useful on-line adaptive user modelling.

In order to address this challenging aspect of the VICO system, we first analysed the general problem of observation-based user modelling for in-car spoken dialogue information systems, cf. [Bernsen, VICO deliverable D10, August 2002]. Starting from the list of potential VICO user modelling functionalities in the VICO TA, D10 analyses approx. 25-30 user modelling options categorised using the following proposed typology for the field:

1. information on the *driver's task objectives,* including preferences for, or habits with respect to, e.g., hotels, restaurants, tourist and point of interest information, news topics, address locations;
2. information on the *driver's communication with VICO,* including, e.g., the driver's native language, mastery of foreign (VICO) languages, voice properties, strong accent or dialect, speech disorder, attitudes such as talkativeness, scepticism, indecisiveness, politeness, speaking style, toleration of external disturbance;
3. information on the *driver's experience with VICO itself,* including the distinction between being expert, casual user, or novice in using the system.

D10 also proposes a series of evaluation criteria which make explicit the properties that a certain kind of driver information must have in order for that information to be a suitable candidate for observation-based user modelling in VICO. The criteria are that:

1. the chosen user modelling functionality should be top quality in terms of its usefulness to all or most drivers;
2. the user modelling functionality should provide genuine driver adaptivity (or observation-based user assistance) without significant drawbacks;
3. the user modelling functionality should be technically feasible and possible to implement without extreme effort (since we do not have the time for putting extreme effort into them). The technical feasibility and extreme effort conditions not only refer to the analysis, specification, and implementation of the dialogue manager part of the

user modelling functionality but may also involve, e.g., experimentation with new functionality in the speech recogniser and/or the NLU;

4. the user modelling functionality must be based on clearly verifiable information about the driver, for instance information which can be gathered from the dialogue history.

Based on these criteria, the user model information typology, and the large variety of specific kinds of driver information analysed, the conclusion is that the categories (1) and (3) in the typology above jointly include several clear possibilities for on-line collection and use of driver information, whereas category (2) in the typology, although rich in user modelling options, includes fewer obvious candidate types of driver information, if, indeed, any.

We therefore chose to develop a user modelling candidate from category (1) above for the 1st VICO prototype, one which bears a direct relationship to one of the driver tasks enabled by the prototype. It is to model the driver's hotel preferences and use the developed models to support the individual driver's hotel selection dialogue.

4.12.2 VICO UM specification

The next step was to specify hotel reservation preferences user modelling [Bernsen 2002c] in the context of general-purpose user modelling functionality, so that additional user modelling, dealing with other types of information, could be added to VICO without re-implementation of the general user modelling functionality involved. This was done in close contact with the specification of the hotel reservation task done at NISLab at the same time.

During the specification work, it was important to keep in mind that the primary goal of UM specification was not hotel selection support *per se* but the creation of a user modelling module which could undertake several different user modelling functions in a thoroughly modular fashion, clearly separating general user modelling functionality from user modelling task-specific functionality. Given that, the particular goal of the VICO hotel preferences user modelling functionality is to facilitate the driver's hotel selection dialogue by making use of knowledge of the driver's hotel preferences in the past.

Analysis showed that VICO must perform the following tasks in order to support the driver in the hotel selection phase of the hotel reservation phase:

1. identify the driver;
2. create a new hotel preferences UM for unidentified drivers;
3. retrieve that driver's hotel preferences UM from VICO's UM module UMDA (user modelling domain agent);
4. collect information about the driver's hotel preferences from the driver's hotel reservation dialogues with VICO;
5. update the driver's hotel preferences UM with the collected information;
6. store the updated hotel preferences UM;
7. use the hotel preferences UM information when the driver is booking a hotel.

In fact, if one subtracts the specific reference to hotel reservation in the list above, the seven functions listed turn out to be generic ones which will be involved in all or most observation-based user modelling, whatever the nature of the specific driver information dealt with (see [Bernsen and Dybkjær 2002].

The consortium had agreed to postpone (1) above, i.e. driver identification, to the second phase of the VICO project. In the following specification and UM implementation, therefore, we assume that VICO is able to identify the driver the first time the UM runs. This is done by hard-coding the user's identity into the NLU hotel reservation input frame to the DM. Step (2) above is then performed by the UM module, whereupon all subsequent users of the first VICO prototype will be regarded as one and the same driver! Implausible as that may be from

the point of view of future realistic use of the VICO system, this approach will certainly enable us to thoroughly evaluate the UM's update algorithms (step 5).

Based on the hotel reservation task specification, we learned that the following driver hotel preference behaviour feature-value pairs will have to be collected and used by the hotel preferences UM:

1. type [VALUE = HOTEL]
2. hotel name [VALUE = NAME]
3. hotel address [VALUES = ADDRESS ITEMS]
4. number of stars [VALUES = 1,2,3,4, or 5]
5. hotel chain [VALUE = NAME]
6. hotel location [VALUES = TOWN, OUTTOWN, or COUNTRYSIDE]
7. max. prices for single (S) and double (D) rooms [VALUES = S: X Euros and D: Y EUROS]
8. restaurant in hotel [VALUE = TRUE or FALSE]
9. protected parking [VALUE = TRUE or FALSE]

Focusing on the list above, the analysis produced an interesting fact. It is that the attribute/value pairs in the list are actually of two fundamentally different kinds. The first kind are the *generic* hotel characteristics, i.e. 1, 4, 5, 6, 7, 8, and 9. The second kind are the *specific* hotel characteristics, i.e. 2 and 3 which even, at least as a matter or practical fact, imply one another. That is, in the practical world, at least, it rarely happens that two different hotels share the same address. Conversely, even if many hotel names are shared by several hotels, such as the Palace Hotel and many others, few, if any, drivers would insist to stay in hotels bearing a particular name. Thus, when a driver has the Palace Hotel in mind, this means in practice that the driver has a particular hotel in mind, at a particular address, rather than all the hotels called the Palace Hotel wherever they might be located.

Generic hotel characteristics are just that, i.e. characteristics that may belong or not belong to any particular hotel. They are also *partial* in the sense that any actual hotel has a large amount of generic properties which users may be interested in (cf. Section 4.11.1). This means that, independently of where the driver happens to be when the driver wants to book a hotel, the UM can use those characteristics to facilitate the driver's hotel selection task. The *specific* hotel properties, on the other hand, are the opposite of being partial, i.e. they are *general* because they imply all the user-relevant generic properties of a particular hotel. This means that the UM cannot use the specific hotel characteristics to support the driver's hotel selection task independently of where the driver happens to be. Clearly, for instance, if the driver is in Hamburg and wants a hotel right there and then, then the fact that the driver has often preferred a particular hotel in Munich is quite irrelevant to the current hotel selection task. In fact, the VICO hotel preferences UM would appear rather daft if it were to offer the driver the hotel in Munich in which the driver likes to stay.

Even if the above reasoning may seem unusually obscure in the context of software development domain analysis, its implications are not. We concluded that the hotel selection UM could have two quite distinct functionalities. The first, *location-independent* functionality is to offer the driver hotels which have the generic properties of the hotels which the driver has preferred in the past. The second, *location-dependent* functionality is to offer the driver the specific hotels which the driver has preferred in the past *if and only if* the driver happens to be in the particular area in which those hotels are located.

Another interesting phenomenon we found during domain analysis was that the generic hotel properties listed above are not in all cases independent of one another. For instance, if a driver has a persistent record of visiting the Hiltons, then that driver most definitely will not have a

record of staying at low-priced one-star hotels. Now, given that the driver may not state all selection preferences in one input turn but only, say, one or two of these, it is important that the system uses its knowledge about the interdependencies among some of the generic hotel properties to avoid blatantly unhelpful suggestions, such as coupling a driver's input "VICO, please find me a five-star hotel" with that driver's record of staying at hotels costing less than 50 Euros per night or less. The predictable consequence would be empty DB returns. In other words, the hotel preferences UM must be able to reason about generic hotel property interdependencies.

For many UM tasks, such as that of modelling drivers' hotel selection preferences and many others, and apart from the postponed issue of identifying individual drivers, one of the harder design issues will often be that of designing how to *update* the UM with new information. As for UM updating in the case of hotel selection preferences, it would appear to be too simplistic to simply collect all past driver hotel selection constraints and average over these. The problem is that drives may acquire new habits, such as the emerging habit of beginning to stay at more expensive hotels. If, in this case, the update algorithm is one of averaging over the past, and if the driver has a long UM record of staying at inexpensive hotels, then the UM may never fully realise that the driver has changed hotel habits. The practical effect would be a driver's hotel preferences UM which invariably suggested hotels of a standard too low for the driver's current preferences. At the other extreme, the update algorithm might simply take into account the generic properties of the driver's latest booked hotel. However, suppose that the driver were on that occasion forced to book low-standard hotel because none other were available. In this case, an update algorithm which ignores all but the latest driver hotel selection might go seriously wrong.

It seems a plausible guess that the reason why most attempts at designing observation-based user modelling for user adaptation have gone wrong, is that they have adopted a less-than-fortunate set of update algorithms. No matter how shrewd the design analysis, it remains difficult to strike the right balance between time-constant and time-varying user behaviour in the adopted update algorithms. Fortunately, it is quite easy to modify the VICO UM's update algorithms should the testing of the 1st prototype make this desirable. The difficult part is to find out in which direction to modify them in order to optimise user satisfaction. As they are in the 1st prototype, the update algorithms which deal with generic hotel properties essentially take the two latest user hotel reservations into account. The update algorithms, one per generic hotel property, do not average over the properties of the two last hotels but, rather, combines them and interpolate between them when required.

As for the driver's previous selections of specific named hotels and their locations, on the other hand, i.e. the location-dependent part of user modelling, the VICO UM preserves and uses all of them, no matter how long ago it was when the driver stayed in a particular hotel.

4.12.3 VICO UM implementation

This section describes the on-line use of the hotel reservation UM.

The context of use of the hotel selection UM is that the driver tells VICO that the driver wants to book a hotel, possibly adding some selection constraints in the first input turn related to this task. Thus, the driver may say, e.g. "VICO, can you find me a three-star hotel around here?". This is the point where the hotel preferences UM should be brought into play. If VICO waits for additional user turns before using the UM on-line, then the usefulness of the UM will rapidly decline. It also seems clear that the UM should never override the driver's own stated selection constraints nor their implications, such as that "one-star hotel" implies cheap and non-Hilton. The UM therefore needs reasoning capabilities which help avoid the pitfalls due to interdependencies among hotel selection constraints. Moreover, it would appear reasonable to adopt the principle that, if the user's own stated hotel selection constraints are sufficient for

uniquely identifying a hotel through querying the DBs, then the UM's model of that driver's hotel preferences should not be brought into play at all this time around. A final point to be mentioned here is that, since the UM has no knowledge of where the driver is at the moment, the UM should forward its updated model of the driver's past hotel/city pair preferences to some other VICO module which actually knows where the driver currently is. From a system development point of view, this seems clearly preferable to providing GPS information to the UM itself.

Based on design considerations such as those discussed above, VICO's hotel preferences UM can be characterised by the architecture and (mainly) external communication shown in Figure 4.8. The (numbered) sequential communication steps in Figure 4.8 are explained in the list following the figure. The user modelling module (UM) is general-purpose. It consists of a UM scheduler (UMS), a hotel preferences UM agent (HRUM), and a database. The database stores (a) a full record on observed driver behaviour per UM task, (b) a record of user ids for the drivers of a particular car, and (c) an individual updatable user models per driver, organised according to driver id and UM task. In this way, new user modelling tasks can be added by: re-using the UMS, creating a new task-dependent UM agent with the necessary reasoning and updating capabilities, adding a new task-dependent record to the database, and re-using the user id record and the models of the individual users, in the latter case adding a new task-dependent user model to them.

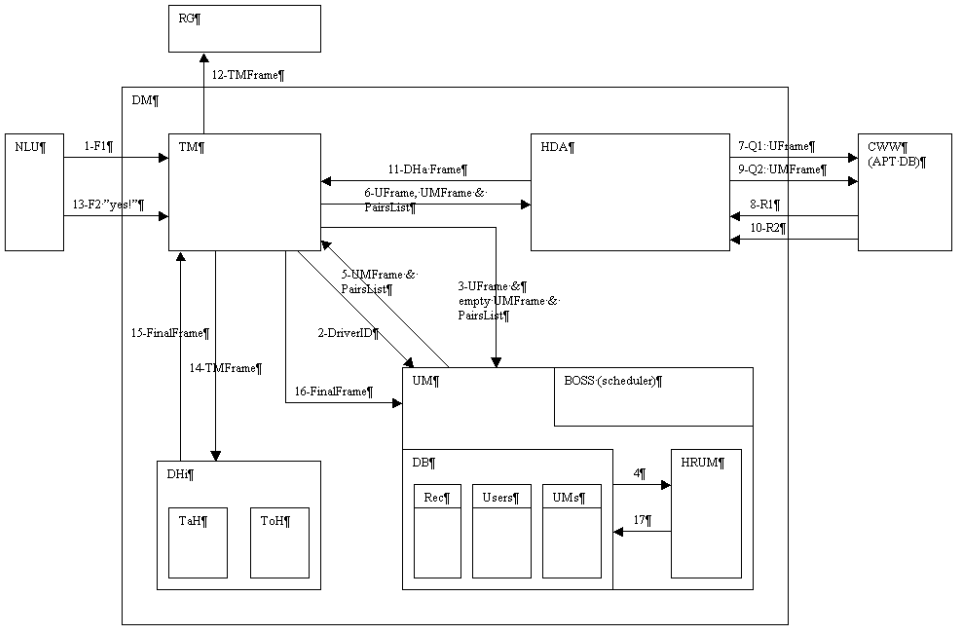


Figure 4.8. Architecture and (mainly) external communication of VICO's user modelling module as implemented for adaptive hotel selection support.

1. The user begins the dialogue.
2. As soon as the driver is identified, the TM sends the driver's id to the UM.
3. The TM sends the UFrame (what the user said), UMFrame (empty frame which UM has to fill), and the empty CityHotelPairs list to the UM.
4. The UM retrieves HRUM from the database (UMs table).

5. The UM sends UMFrame and filled CityHotelPairs to the TM.
6. The TM sends both frames (UFrame and UMFrame) and the CityHotelPairsList to the HDA (DA_hotel).
7. The HDA queries the CWW with UFrame data (so far not using the UM data).
8. CWW replies to the query (R1).
9. If there is any useful information in the UMFrame and if the previous DB return (R1) includes more than three hotels, then the HDA queries the CWW with the additional UMFrame data.
10. The CWW replies to the query (R2).
11. The HDA sends the DHa frame (which contains information about hotels) to the TM.
12. The TM sends TMFrame to the response generator in order to propose one or several hotels to the driver.

In the following dialogue, the driver and VICO eventually agree on a specific hotel. Then:

13. The NLU inputs a frame which expresses the user's agreement.
14. The TM stocks this information in the DHI.
15. The TM retrieves the Final Frame (which contains full information on the selected hotel) from the DHI.
16. The TM sends the Final Frame to the UM.
17. The UM stores the frame in the database (Records table) and then updates the HRUM and stores it as well.

The list above does not describe what happens to the CityHotelPairs list. i.e. the list of previous pairs of cities and hotels visited by the driver. What happens is that the HDA looks for information on the city in which the user wants to book a hotel. If a city is provided by the driver in the first hotel reservation input, the HDA checks if that city is in the city-hotel pairs list. If it is, the corresponding hotels are offered to the driver, overriding any driver hotel selection input. If the city is not in the list, the list is disregarded. If the driver did not provide a city, the HDA must wait until city names are returned from the DB in response to the query made. If any of the returned city names match the cities in the city-hotel pairs list, the corresponding hotels are offered to the driver, again overriding any driver-provided hotel selection input. If there is no match, the city-hotel pairs list is disregarded by the HDA.

4.13 CWW query XML wrapper and unwrapper example

Whenever the XML wrapper receives a frame and information about query type, it first decides whether it is going to make a query about route, POI, hotel, or restaurant. Then a tree structure is built by taking the relevant information from the frame slots as indicated by the query type. Using the Xerces C++ parser this tree structure is then transformed into an XML file. The resulting XML file is sent as a query to the CWW.

The XML unwrapper works the other way around, i.e. it receives an XML file and, using the Xerces parser, it extracts the information contained in the XML file and builds a tree structure. Finally, the node information in the tree structure is extracted and inserted into the relevant slots in the frame.

Figures 4.9 and 4.10 show examples of a hotel reservation query and the corresponding CWW return, respectively. In Figure 4.9, a hotel has already been selected by the user and sufficient information has been provided to enable booking of a room, i.e. the system knows the arrival and departure dates and the number and kinds of rooms (single and/or double) the user wants to book. The selected hotel is identified via an ID number. For the system to confirm a booking, a credit card number is needed. Even without a credit card number, a reservation query can still be made in order to find out about room availabilities but the room(s) will only

be finally reserved once the credit card number is provided. Figure 4.10 shows the response to the query in Figure 4.9. As can be seen, the desired room is available and has been booked, and the total price the user will have to pay is 776 Euros.

```
<get_hotel_reservation>
   <hotel_id>N027</hotel_id>
   <credit_card>1234123412341234</credit_card>
   <arrival_date>2002-09-22</arrival_date>
   <departure_date>2002-09-30</departure_date>
   <persons_number_single>1</persons_number_single>
</get_hotel_reservation>
```

Figure 4.9. A query about hotel reservation wrapped in XML.

```
<hotel_reservation>
   <hotel_id>N027</hotel_id>
   <total_price>776.0</total_price>
   <availability>true</availability>
   <reserved>true</reserved>
</hotel_reservation>
```

Figure 4.10. A response about hotel reservation wrapped in XML.

4.14 Test results and objectives

Since we have not yet run tests with a complete running version of the 1st VICO prototype, the main evaluation result to report so far is the fact that the DM architecture has demonstrated its generality by allowing straightforward implementation of the hotel reservation and restaurant reservation tasks which were specified after the implementation of the present DM. It should be added here that "straightforward" implementation does not imply "simple" or "fast" implementation. The hotel reservation task is a large and complex task which, furthermore, involves incorporation of user modelling functionality, and its implementation is still going on at the time of writing.

4.15 Issues

We expect that thorough testing and evaluation of the DM in the context of a complete running version of the 1st VICO prototype will provide a wealth of information on issues which we have not been able to test and evaluate before. Some of these issues are:

4.15.1 Dialogue structure design

The dialogue structures for the different tasks which have been implemented for the 1st VICO prototype incorporate slightly different dialogue strategies. Thus, whereas the route task dialogue structure is close to being a by-the-book implementation of graceful degradation of the dialogue when faced with user input error, the POI, hotel reservation, and restaurant task implementations are slightly more direct and slightly less patient with user error. Evaluation of the 1st VICO prototype will provide information which will help decide on which general strategy to follow for the final VICO prototype.

4.15.2 Recogniser behaviour

We also need data on possible patterns in the recognisers' output of relevance to dialogue structure design. For instance, how often does it happen that the recognisers produce

persistent noisy input across several user dialogue turns? If this happens rather often, then we may have to modify the dialogue structure to put more emphasis on making the dialogue recover from such problems. Or, how well do the recognisers perform when expecting, and receiving, user answers to yes/no questions? In our current dialogue structure design, it is assumed that system yes/no questions represent a solid fall-back strategy for handling input error problems. If this assumption turns out to be questionable because of recogniser word error rates, then we may have to modify the role of yes/no questions in the dialogue structure.

4.15.3 Database behaviour

At the time of writing, we still have not been able to connect to the APT and TeleAtlas databases due to problems with the CORBA versions used at the different partner sites. Instead, we have had to develop and work with various dummy databases at various stages of module development and testing. This means that we still have to see whether the databases turn out to generate issues of dialogue design. As an example, it was found that the databases sometimes return output consisting of [..., Trento, Trento, Trento, ...], due to the, to us, unexpected fact that there is both a province called Trento (we thought that the province was called Trentino), a city in that province called Trento, and even a part of city in that city called Trento. Somehow, either the DM or the RG must make sure that the system's output is comprehensible to drivers in such cases, given the fact that output of the form "You want to go to Trento, Trento, Trento ..." does not seems sufficiently comprehensible.

4.15.4 Meta-issues

Evaluation of the integrated 1st VICO prototype is also expected to provide crucial information for dealing with input issues which are more general than the individual task or which are common to several tasks. We have already identified a series of such issues, including, among others, users providing an opening greeting to the system, a farewell greeting, asking the system to repeat, asking the system to cancel an already agreed task, or providing persistent low-confidence input or noisy input from the first input onwards. The challenge is to get "first closure" to this set so that we can design a general solution for all such meta-issues. Still, the solution must be extensible, of course, since we may discover additional issues later on. Given "first closure", we plan to implement a meta-issue solution as if it were a separate task to be solved by VICO, rather than laboriously integrating meta-issue solutions into all existing tasks, dialogue structure specifications, etc. This means that we need to be able to draw a clear distinction between meta-issues, on the one hand, and task-related issues on the other.

5. The response generation component

5.1 Introduction

The goal of the response generator is to accept a high-level semantic output description from the DM and convert that description into linguistically, socially, and culturally correct contemporary spoken output language in the language chosen for the current dialogue. The RG's output expression is a text string which will be converted into speech by the speech synthesiser. The RG handles three different languages, English, German, and Italian. Language-setting is being done during initialisation by the SM.

5.2 Architecture and data flow

Figure 5.1 shows the general architecture and main data flow of the VICO RG.

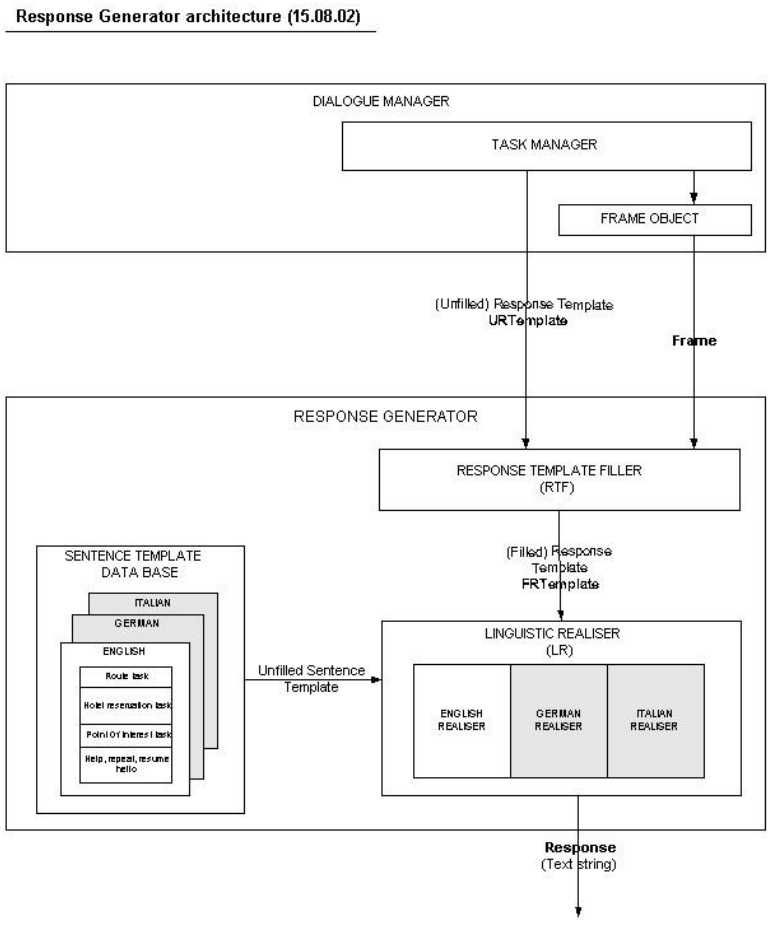


Figure 5.1. The VICO response generator architecture.

In order to produce each system output turn, the RG receives the current semantic frame together with an unfilled response template from the DM's task manager (TM). We call this

response template (URTemplate in Figure 5.1) "unfilled" because it includes, at most, the relevant *attributes* for the output to be produced but does not include the actual *values* in context for those attributes. The actual values are included in the current semantic frame which is also passed to the RG by the TM.

Thus, the task of the first module in the RG, the response template filler (RTF), is to read the current semantic frame in order to retrieve the values corresponding to the empty attributes in the URT. These values are then filled into the URT, turning the URT into an FRT, i.e. a filled response template. Having done that, the RTF discards the semantic frame and passed on the FRT to the linguistic realiser module (LR).

To illustrate, let us continue the example from Section 4.6. In that example, the URT was: "**12:[non-empty-items],13**".

The response template filler fills the slots of the URT with the information in the corresponding frame. In the present example, the RTF receives:

URTemplate -> 12:[non-empty-items],13

The RTF also receives the frame shown in Table 4.4 and repeated in Table 5.1.

Type	**route**
Topic	
PartOfCountry	Fyn
City	Odense
PartOfCity	SE
Street	Cikorievej
Number	20
ConfScore	3
TypeMismatch	0
LexAmbiguity	0
Unique	1
Complete	1
NonUniqueMax3	-1
NonUniqueMore3	-1
NonExistent	-1
Inconsistent	-1
StreetNumbersDB	1

Table 5.1. Example of frame received by the response template filler (RTF).

The filled response template (FRT) then becomes:

[12][non-empty-items{poc:Fyn|city:Odense|street:Cikorievej|number:20|}][13]

This FRT is sent to the LR. The linguistic realiser receives the filled response template and performs two basic tasks. The first task is to retrieve from the LR database the sentence(s) corresponding to the number(s) in the FRT. Secondly, the LR replaces the sentence numbers in the FRT and performs some basic syntactic processing in order to include the response values into the retrieved surface language string parts. In order to do so, the linguistic realiser consults the LR database which is organised in two tables. One table contains the sentence templates (per language), the other contains the language generation patterns (per language as well).

In the present example (language = English), the linguistic realiser retrieves the following sentence templates from the database:

12: → You want to go to [non-empty-items].

13 → The route is being planned.

The linguistic realiser also retrieves from the database the non-empty-items pattern, which in this case has the following attributes:

non-empty-items{poc:|city:|street:|number:|}

The attributes are extracted from the filled response template:

[12][non-empty-items{poc:Fyn|city:Odense|street:Cikorievej|number:20|}]

The corresponding DB entries look as shown in Table 5.2 and Table 5.3.

ENG->12	You want to go to **[non-empty-items]**.
ENG->13	The route is being planned.
GER->12	Sie moechten also **[non-empty-items]**.
GER->13	Einen Augenblick, die Route wird berechnet.

Table 5.2. Sentence templates in the response generator's database.

pattern	EngOutput	GerOutput
non-empty-items{poc:|city:|street:|number:|}	street:|number:|in|city:|poc:|	zur|street:|number:|in|city:|im Bundesland|poc:|

Table 5.3. Patterns in the response generator's database.

Based on the processes described above, the LR produces the following text string:

Response: You want to go to Cikorievej 20 in SE Odense Fyn. The route is being planned.

The text string is sent to the TTS synthesiser which generates the spoken output to the user.

The RG is also responsible for generating output text for presentation on the in-car display. After experimentation at NISLab [Bernsen and Dybkjær 2002], it was concluded that the content of the display text should be brief and to-the-point, so this is how the display text has been implemented.

5.3 Test results

The 1st VICO prototype includes 13 response generation components, three (English, German, Italian) for each of the tasks route, POI, hotel reservation, and restaurant reservation, and one (English) for the VICO information task. In addition, the RG has a general architecture, described in Section 5.2 above, for incorporating any number of new components in any language and for any task. The RG has been, and continues to be, tested from a software point of view. However, this testing is straightforward compared to the second kind of evaluation which VICO response generation needs to undergo in the context of the 1st prototype (see Section 5.4).

5.4 Issues

The main limitation of the VICO response generator is the template-based approach adopted. This approach is known to be likely to run into complexity limitations for very complex tasks. However, we still need to see examples of tasks so complex that the approach begins to falter. We are virtually certain that this will not be the case in the VICO project. Indeed, arguably, as long as we are able to beat the complexity would-be barrier and produce the dialogue

structures for the spoken human-system dialogue used in VICO, we are confident that we can also produce the corresponding template-based response generation. During the first phase of the VICO project, we have continued to elaborate our internal procedures for response generation, so that it is now possible to generate and test response generation for a single language/task pair within a seek or so. Thus, we *might* have used parser-based RG in VICO but the template-based approach adopted does not seem to incur inordinate production time.

With the 1st integrated VICO prototype, we face a very important task which demands consortium-wide discussion. It is to evaluate the responses generated for the already implemented VICO tasks and decide on the *style* of VICO's responses to the drivers. The discussion will help determine the general style in which VICO addresses the driver, so that we can arrive at an "official" VICO output style for the final prototype. As it stands, the current VICO RG uses somewhat different response styles per task. The reason is that we want the consortium to evaluate those styles and comment on them in order to create consensus on the style to adopt. It may be remarked here that, given some form of style guide, it is quite straightforward to revise the system's surface language response style to conform to the guide. All it takes is to replace, per language-task pair, the current response string patterns in the linguistic realiser DB, by the style-conformant ones.

6. The Speech Synthesis Component

The second part of Work Package 5.3 deals with Text-To-Speech (TTS) synthesis. It was decided to integrate commercially available TTS synthesisers in their most advanced versions, in order to provide the highest quality of speech output for the research prototype. The TTS systems were evaluated for three languages: English, German and Italian. In order to find the most suitable TTS system, the most advanced systems in the market were evaluated using the following criteria: intelligibility, naturalness, pronunciation, prosody, transcription, pronunciation dictionary for exceptions, and support of the Microsoft Speech API (SAPI) version 5. The memory requirements of the synthesis systems were estimated and compared, but it must be stated that memory requirements were not an optimisation criterion for this evaluation. The intention was to find a TTS system that can handle the three languages British-English, German, and Italian in high quality. For the first prototype system we plan to integrate only one TTS system, whereas for the in-car demonstrator, which is due in 2003, the best available system for each language should be incorporated. Finally, the component interface for the speech synthesiser was defined in IDL, so that the component can be integrated in the CORBA system framework.

6.1 Evaluation of commercially available TTS systems

As our intention was to find the best speech synthesis product, we first evaluated the quality of the most advanced speech synthesis systems. The evaluation must be considered subjective because only about 10 people (subjects) listened to the sample sentences.

6.1.1 Evaluation criteria

In order to subjectively evaluate the quality of the system, the subjects had to assess the synthesisers, using the following criteria:

1. **intelligibility**: How easy is it to understand the speech output?
2. **naturalness**: How similar is the synthetic speech output to a human voice?
3. **pronunciation**: Is the phoneme inventory complete (for inner- and intra-word units)?
4. **prosody**: Are word pauses and intonation correct?
5. **transcription**: Can the synthesis system handle abbreviations and language-characteristic expressions, like enumerations, time and distance expressions, etc.?

Furthermore, the systems should allow for incorporation of a pronunciation dictionary (in order to handle exceptions and abbreviations) and support the Microsoft Speech API (SAPI) version 5 (so that they can easily be integrated and exchanged). Finally, the memory requirements of the synthesis systems were estimated and compared, because quality corresponds partly to the size of the phoneme of unit inventory.

6.1.2 Set of evaluation sentences

The evaluation criteria influence each other, e.g. the intelligibility suffers from badly pronounced words, etc. We decided to evaluate the criteria separately in order to be able to specify particular mistakes more clearly. Four sets of sentences were compiled for each language in order to evaluate according to the defined criteria.:

Set A: Scenario-typical system responses from the VICO system. This set of responses is used to evaluate intelligibility and naturalness of the speech synthesis for the specific VICO demands.

Set B: This set contains specific sentences that are well suited for controlling the completeness of the phoneme inventory.

Set C: Long sentences with relative clauses to check the correctness of word and sentence prosody.

Set D: A set of sentences that covers time, date, weight, and other measures, and also covers language-typical abbreviations in order to evaluate the correctness of handling the exceptions in transcription.

The collection of sentences for the three languages are listed in Appendix A. We would like to thank our Italian partners from ITC-Irst, particularly Marco Matassoni, for providing us with the four sets of Italian sentences.

6.1.3 Set of TTS synthesis systems to be evaluated

The most promising candidates among commercially available TTS synthesis systems were selected using in-house know-how and publications on the internet. The highest quality TTS systems from the following companies were selected: Elan Speech, Babel-Infovox, Svox, Bell-Labs, Loquendo, Scansoft, and AT&T. For Internet references, see Chapter 8. The following table summarises some of the basic criteria that were considered in order to select the most suitable TTS system for the VICO demonstrator.

TTS System	German voice	English voice	Italian voice	support for SAPI 5.0	memory requirements	exception lexicon available
Elan Tempo	✓	✓	✓	✓	7 MB	✓
Svox TTS v2.1	✓	✓	✓	✓	6 MB	✓
Infovox 330	✓	✓	✓	✓	< 10 MB?	✓
Bell-Labs	✓	✓	✓	?	?	?
Loquendo	✓	✓	✓	✓	> 60 MB?	✓
Scansoft RealSpeak	✓	✓	✓	✓	> 30 MB?	✓
AT&T Natural Voices	✓	✓	-	✓	> 30 MB?	✓

Table 6.1. Basic selection criteria for the TTS candidate systems.

Table 6.1 clearly indicates that, except for *Natural Voices*, all TTS systems support all three languages, British-English, German and Italian. Unfortunately, some information was missing on the web pages, particularly on the *Bell Labs* page, so we did not find out the precise memory requirements, SAPI support, and exception lexicon integration information for the *Bell Labs* TTS. Memory requirement were only specified for a few systems, so we had to estimated the requirements of the other systems using the Windows NT Task Manager. It can be seen in Table 6.1 that the memory requirements of the systems differ greatly. This is due to the fact that some systems are based on diphone units only, while other systems also incorporate much longer units and a unit-selection method. SAPI was supported by all the TTS systems (except that we do not know about *Bell Labs*), but particularly for *RealSpeak* the proprietary inferface is also needed, because not all SAPI functions are implemented yet.

6.1.4 Evaluation results

This section discusses the evaluation results for the TTS synthesis systems for the three languages German, British-English, and Italian. The results describe the quality of the current state of the TTS systems. The quality of most of the systems might improve significantly over the next months due to the fast technological progress. This means that a second short

progress evaluation is needed before integrating the TTS system for the final demonstrator next year.

About 10 subjects, native as well as non-native speakers, were asked to subjectively assess the quality of the TTS systems. Each subject was presented with the same sentences for all the systems, but in different orders. After listening to all utterances of one sentences, the subject gave an average score from 1 (very poor) to 10 (excellent), according to each evaluation criterion for each TTS system. The scores of all subjects were averaged and rounded to 0.5 precision. The following tables summarise the evaluation results.

6.1.4.1 Evaluation results for German TTS systems

The evaluation results are summarised in Table 6.2.

TTS System	Voice	intelligi-bility	natural-ness	pronun-ciation	prosody	trans-cription	overall score	overall rank
Elan Tempo	**Dagmar**	8.5	6.0	8.0	7.0	6.0	7.1	2
Svox TTS v2.1	**Nicole**	7.0	5.5	6.0	4.0	3.5	5.1	5
Infovox 330	**Helga**	5.5	5.0	6.5	4.0	2.5	4.7	6
Bell-Labs	**Male**	6.5	4.5	7.0	6.0	5.0	5.8	4
Loquendo	**Ulrike**	4.5	7.0	3.5	2.5	4.0	4.3	7
RealSpeak	**Vera**	8.0	9.0	8.0	6.5	6.0	7.5	1
Natural Voices	**Klara**	6.0	8.0	7.0	5.5	6.0	6.5	3

Table 6.2. Evaluation results for German TTS systems.

RealSpeak. The *RealSpeak PCMM* system turned out to produce the best-performing synthesis. This was due to its excellent voice, which sounds very natural. *RealSpeak* was among the two best systems in all five categories evaluated. The few pronunciation mistakes it committed can be avoided by using an exception lexicon. Transcription errors can be avoided by intelligent pre-processing of the system's responses. Only the prosodic problems, which are still typical for the unit-selection TTS systems, will remain in the demonstrator. However, none of the other candidates is significantly better in prosody modelling, so *RealSpeak* is the best candidate for the first VICO demonstrator.

Elan Tempo. This system ranked second in our evaluation. It only suffered from its lack of naturalness as the sampling frequency is lower than that of *RealSpeak* and *Tempo* is based only on diphones whereas RealSpeak also uses larger units. *Tempo* performed very well in all other categories, but the machine-like voice prevents it from being the best-performing system. *Elan Speech* is currently working on a new TTS system, called *Sayso*, which relies on unit-selection like *RealSpeak*. Once on the market for our three languages, *Sayso* will be an interesting candidate for next year's evaluation.

AT&T's Natural Voices. This system ranked third in the evaluation. Although having a quite natural voice, the system suffered significantly from erroneous pause modelling and too high speed in some of the sentences. Furthermore, the intonation of questions was not clear enough.

Bell Labs. The fourth rank was taken by the *Bell-Labs* synthesis. This system ranked quite well in most categories, except for naturalness. The metal-like voice turned out to be the most annoying one among the systems tested. In some sentences, the system seemed to be stuttering, which degraded the intelligibility.

Svox. This system ranked fifth, as it received average scores in all categories. The main problem was that it was not able to stress questions at all. It also used some strange phonemes that belong to the Swiss-German dialect and not to the common German language. It sometimes stuttered and made pronunciation mistakes.

Infovox. The *Infovox* system ranked sixth due to its lack of naturalness. The modelling of special transcriptions seems to be no issue for *Infovox*. The system uttered most of our test sentences in set D erroneously.

Loquendo. Finally, *Loquendo* lost the evaluation. Although providing a quite natural voice, the prosody and pronunciation mistakes were so significant that some of the sentences could not be understood. The speed was inconsistent and the voice was distorted at some points.

6.1.4.2 Evaluation results for British-English TTS systems

It was decided to reduce the candidate set to the five most promising systems as there was no evaluation version of the Svox TTS system available. The Infovox TTS engine was not evaluated, as a first quick evaluation confirmed that the voice was very unnatural just as in the German version. Table 6.3 summarizes the results of the British-English TTS evaluation.

TTS System	Voice	intelligi-bility	natural-ness	pronun-ciation	prosody	trans-cription	overall score	overall rank
Elan Tempo	Vicky	5.5	4.0	3.5	3.5	5.0	4.1	5
Bell-Labs	Male	7.0	5.5	8.0	7.0	2.5	6.0	3
Loquendo	Elizab.	6.0	5.5	5.5	3.5	3.0	4.7	4
RealSpeak	Jane	7.5	7.0	7.0	7.5	5.0	6.8	1
Natural Voices	Audrey	7.5	7.0	6.0	5.5	6.5	6.5	2

Table 6.3. Evaluation results for British-English TTS systems.

RealSpeak. The system also ranked first in the British-English evaluation. The system output is easy to understand, sounds quite natural, and makes very few pronunciation and prosody errors. It incorrectly modelled the intonation for some questions. *RealSpeak*'s weakest point are transcription errors. It could not handle the special characters for room and weight measures and also failed to model the different pronunciations of homographs, like the words "lead" and "appropriate" according to the context. The pre-processing of the transcription needs to be improved. Nevertheless, the overall quality of *RealSpeak* was superior when compared to the other four candidates.

Natural Voices. The TTS system *Natural Voices* achieved almost the same overall score as *RealSpeak*. The voice sounded very natural and the modelling of special cases in the transcription was the best of all systems. The prosody of the *Natural Voices* system needs to be improved, particularly for longer sentences which contain relative clauses.

Bell Labs. The system ranked third due to its excellent pronunciation and good prosody modelling. It proved to be the best of the synthesis systems that are based on diphones only, as *RealSpeak* and *Natural Voices* also incorporate larger units and hence require much more memory. *Bell Labs* only provide an American-English version of the synthesis system. The pre-processing of the transcriptions must be improved as the system has problems handling some important abbreviations. In many cases, the system lacked context modelling as it hardly ever could distinguish between homographs which are not homophones.

Loquendo. Although having a potentially quite natural voice, the actual naturalness of the system output is heavily degraded due to distortions. Some sentences were not understandable

due to bad prosody modelling. Pauses appear at wrong places in the sentence. Every dot in the sentence is handled as a full-stop (sentence end), and that is wrong in many cases, like after ordinal numbers and abbreviations.

Elan Tempo. The Elan system had the lowest overall quality as its diphones provided insufficient pronunciation and prosody modelling. The diphone inventory seems to be incomplete and the pronunciation of some words suffers from heavy distortions. Flaps were not modelled correctly. Homophones that span more than one word, like "Why choose white shoes?" cannot be distinguished. However, homographs with their different pronunciations were modelled quite well.

6.1.4.3 Evaluation results for Italian TTS systems

The evaluation results are summarised in Table 6.4.

The set of evaluation candidates had to be further reduced for the evaluation of Italian TTS systems as *AT&T Natural Voices* does not offer an Italian voice. The Italian voice of the *Bell Labs* TTS was considered unacceptable following a short pre-evaluation cycle. It was also found that *Loquendo* offers several very promising voices for Italian synthesis. That is why it was decided to evaluate the two most promising voices of the *Loquendo* system, Roberto and Valentina. These voices were selected in the short pre-evaluation cycle.

TTS System	Voice	intelligi-bility	natural-ness	pronun-ciation	prosody	trans-cription	overall score	overall rank
Elan Tempo	**Sofia**	5.0	3.5	4.5	4.0	6.0	4.6	4
Loquendo	**Roberto**	7.5	6.5	7.5	6.0	7.0	6.9	2
Loquendo	**Valentina**	7.5	7.0	7.5	6.5	7.0	7.1	1
RealSpeak	**Bianca**	7.5	6.5	6.5	5.5	8.0	6.8	3

Table 6.4. Evaluation results for Italian TTS systems.

Loquendo Valentina. The voice Valentina of the *Loquendo* TTS system ranked first in the Italian evaluation. That is no surprise as the home market of *Loquendo* is Italy. The speech output sounds almost natural due to the great number of units Loquendo employs. All utterances can be understood. When the system falls back to diphone units, which happens for infrequent words, it becomes obvious that some rare diphones are missing or transitions are modelling inaccurately. The prosodic pause modelling is almost adequate and much better than the one for *Loquendo's* English and German voices. Some inner-word prosodic distortions still appear. All the special cases of the transcriptions were handled correctly, except for the room measures.

Loquendo Roberto. The second voice of *Loquendo* ranked second in the evaluation. The male voice of Roberto appeared to be a little less natural than the female voice of Valentina, maybe due to the fact that Valentina's voice is more dynamic, hence the listeners reported the impression that Roberto's voice is not as emotional as Valentina's. Furthermore, it can be objectively stated that the synthesis of the voice Roberto causes more distortions than the one of Valentina.

RealSpeak. The voice Bianca of *RealSpeak* TTS ranked third. Its quality proved to be almost as good as the one of *Loquendo*. While the pronunciation of *RealSpeak* is a little inferior due to a little more distortions, the handling of special cases in the transcription is superior. RealSpeak was the only system that handled all special cases correctly.

Elan Tempo. The Elan TTS system came in last with a significant distance to the other candidates. It is the only system that relies only on diphones. All the other systems also use

larger units. The speech output was not natural at all, some sibilants resembled Spanish more than Italian speech. There were sentences that could not be understood due to heavy distortions in pronunciation and prosody modelling. Some abbreviations, like Spa., were not handled correctly by the transcription pre-processing.

6.2 API definition for the TTS component

A tool that employs the interfaces of the TTS systems was written, so that it is possible to enter a text sentence and listen to the result of the speech synthesis. The knowledge acquired by writing the tool will help a lot when implementing the IDL API for the VICO demonstrator. The IDL API was defined in cooperation with Robert Bosch GmbH. Special thanks are due to Frank Steffens. The IDL API contains functions to:

- initialise/deinitialise the synthesis component;
- get the status of the component;
- generate and output the synthesised speech from a string or a file;
- cancel, interrupt or resume the speech output;
- set the language (German, British-English or Italian);
- set system parameters, like pitch, volume, or speed of the synthesized speech.

The implementation of the IDL interface is already prepared, so that a first implemented version will be ready on the 2nd of August 2002, approximately.

6.3 Summary of the TTs assessment

The evaluation of TTS synthesisers for the three languages German, British-English and Italian, examined in detail intelligibility, naturalness, pronunciation, prosody, and transcription. The evaluation proved that the *RealSpeak* system by ScanSoft can handle all three languages in good quality. *RealSpeak* always ranked among the best systems. Either it was the best system or a very close second. All the other TTS systems commit serious errors in at least one of the languages. Although *Loquendo* won the Italian evaluation, the distance to RealSpeak was so close that it was decided that it is not worth the effort to integrate two different TTS system into the 1st VICO prototype. Only RealSpeak will be integrated and it will handle all the three languages. Another short evaluation will be needed for the final VICO demonstrator in 2003, as the technology advances very quickly, so that new versions of the assessed TTS systems will be available next year.

7. Conclusions and future work

This document has reported on the progress made on natural language understanding, dialogue management, and response generation in the first phase of the VICO project. Five tasks have been analysed, designed, implemented, and, at least, software tested, i.e. the route task, the point of interest task, the VICO information task, the hotel reservation task, and the restaurant reservation task. As the VICO information task has only been developed for UK English, development has comprised 13 input language/task pairs and 13 output language/task pairs. In addition, a general-purpose user modelling module has been developed and demonstrated for the hotel reservation task. Natural language understanding processing has been developed by significantly enhancing the capabilities of an in-house parsing module. Dialogue management has been developed from scratch, and response generation has been developed from scratch. The dialogue manager has demonstrated its general-purpose nature by enabling straightforward inclusion of tasks which were not analysed at the time of its development. Significant progress has been made in developing in-house work flow which enables rapid inclusion of new language-task pairs throughout the development life-cycle. In addition, this document has described the comparative evaluation performed of a number of TTS candidate systems for the 1st VICO prototype.

Future work will have the following aspects, at least:

First, we need to thoroughly test the 1st integrated VICO prototype in order to assess the NLU for all task/language pairs, assess the RG for all task/language pairs, assess the DM in the context of working not only with the NLU and the RG but, more importantly because it will be done for the first time, also with the CWW and the TeleAtlas and APT databases, with the SRs, and with the TTS systems adopted. Some of the most important evaluation objectives have been mentioned above, such as dialogue output style and the definition of a comprehensive set of meta-task issues, including meta-issues of error correction, but many other targets are of course important as well, such as NLU grammar and lexical coverage, or the completeness of the dialogue structures for the individual tasks.

Secondly, we will continue to develop our internal development work flow procedures for NLU task/language pair development and testing, specification and incorporation of new tasks into the DM, and RG task/language pair development and testing. The resulting guidelines will help to significantly cut down development time for new VICO tasks and task/language pairs both for existing staff and new staff members. As most parts of the NLU-DM-RG complex developed are entirely new, it is natural to develop and test the novel components first and then proceed to firmly define the procedures to be followed when extending the components to new tasks and new task/language pairs.

8. References

Bernsen, N. O.: Report on user clusters and characteristics. *VICO deliverable D10,* NISLab, August 2002.

Bernsen, N. O. 2002a: VICO point of interest task specification v.3. *NISLab internal document,* September 2002.

Bernsen, N. O. 2002b: VICO point of interest task specification, short version v.4. *NISLab internal document,* October 2002.

Bernsen, N. O. 2002c: VICO 1st user modelling. Analysis and specification of hotel reservation support. *NISLab internal document,* June 2002.

Bernsen, N. O., Charfuelan, M., Dybkjær, L., and Kolodnytsky, M.: Report on modelling of domain-specific knowledge. *VICO deliverable D9,* NISLab, March 2002.

Bernsen, N. O. and Dybkjær, L.: Exploring natural interaction in the Car. In Bernsen, N. O. and Stock, O. (Eds.): *Proceedings of the International Workshop on Information Presentation and Natural Multimodal Dialogue* (IPNMD-2001), Verona, December 2001. ITC-Irst, Trento, 2001, 75-79.

Bernsen, N. O. and Dybkjær, L.: A multimodal virtual co-driver's problems with the driver. In Dybkjær, L., André, E., Minker, W., and Heisterkamp, P. (Eds.): *CD-ROM Proceedings of the ISCA Tutorial and Research Workshop (ITRW) on Spoken Dialogue in Mobile Environments, Kloster Irsee, Germany*. Bonn, Germany: International Speech Communication Association 2002. An augmented version of the paper will appear in: Minker, W., Bühler, D. and Dybkjær, L. (Eds.): *Spoken Multimodal Human-Computer Dialogue in Mobile Environments.* Kluwer Academic Publishers, 2003.

Berton, A., Harald, H., Omologo, M., Svaizer, P.: Progress Report on the Speech Recognizer and Language Models for German, English and Italian. *VICO deliverable D12,* DaimlerChrysler AG (DCAG), October 2002.

Dybkjær, L.: Hotel reservation and restaurant tasks specification, series of *NISLab internal documents,* July-September 2002.

Kurdi, M.-Z.: Combining pattern matching and shallow parsing techniques for detecting and correcting spoken language extragrammaticalities. In *Proceedings of the 2nd Workshop on Robust Methods in Analysis of Natural language Data* (ROMAND 2002), Frascati, Italy, July 2002, 17.

Kurdi, M.-Z.: A spoken language understanding approach which combines parsing robustness with interpretation depth. International Conference on Artificial Intelligence IC-AI01, Las Vegas, USA, June 2001.

Kurdi, M.-Z.: A chunk based partial parsing strategy for re-ranking and normalizing Nbest lists of a speech recognizer. ESSLLI'99, Utrecht, The Netherlands 1999.

Manstetten, D., Berton, A., Krautter, W., Grothkopp, B., Steffens, F., and Geutner, P.: Evaluation report from simulated environment experiments. *VICO deliverable D7,* DaimlerChrysler AG (DCAG), January 2002.

Speech synthesis websites

Elan Speech TTS: http://www.elantts.com/accueil.html

Babel-Infovox 330: http://www.infovox.se/tproducts.htm

Svox TTS Version 2.1: http://www.svox.ch

Lucent / Bell Labs: http://www1.bell-labs.com/project/tts/

Loquendo TTS: http://www.loquendo.com/en/products/TTS.htm

ScanSoft RealSpeak PCMM: http://www.scansoft.com/realspeak/pcmm/
AT&T Natural Voices: http://www.naturalvoices.com/

9. Acknowledgements

The authors wish to thank to following NISLab colleagues for their contributions to the work reported in the present deliverable: Michel Genereux, Christian Stærmose Hvitved, Anja Johansen, Aziz Joumady, Valeria Lacorte, Marnie Lai, Thomas Heine Madsen, Nikolaj Hald Nielsen, Michael Sørensen.

Appendices

Appendix A – Collection of synthesis test sentences

A1 Evaluation sentences for German TTS systems

A1.1 Set A. Sentences from the VICO Scenario

1. Die Route wird geplant.
2. Welcher Zielort möchten Sie?
3. Leider kann ich Sie nicht verstehen. Vielleicht sollen Sie das System neu starten.
4. Buchstabieren Sie bitte Ihren Zielort!
5. Das VICO System kann Ihnen bei den folgenden Aufgaben helfen:
   1. die Adresse eines Zieles, wie z.B. eines Hotels, aus ungefähren Angaben ermitteln
   2. Telefonnummern von Hotels ermitteln und
   3. das VICO-System erklären.

Das VICO-System ist ein hoch modernes prototypisches Sprachdialogsystem. Obwohl die interne Verarbeitung recht complex ist, is die grundlegende Struktur relativ einfach. Die Hauptkomponenten, die Ihre Eingabe verarbeiten und die sprachliche und visuelle Ausgabe erzeugen, sind die folgenden:

ein Spracherkenner

ein natürlichsprachliches Verstehungsmodul

ein Dialogmanager

ein Antwortgenerator

ein Sprachsynthetisator sowie

ein Bildschirm.

Darüber hinaus benutzt das VICO-System GPS, das globale Positionierungssystem, um Ihre momentane Position festzustellen, und besitzt eine zentrale Wissensbasis mit geographischen Informationen über Straßen, Hotels und andere Sehenswürdigkeiten sowie Informationen über Preise, Öffnungszeiten, Angebote usw. Wird der "Speak"-Knopf gedruckt, wird das System aktiv. Wenn the benutzer spricht, setzt der Spracherkenner die akustische Eingabe in Worte um. Diese werden dann vom Sprachverstehungsmodul nach ihrer Bedeutung analysiert. Diese Information wird an den Dialogmanager weitergereicht, der dann entscheidet, was zu tun ist. Dies kann darin bestehen, das GPS nach der derzeitigen Ort des Benutzers zu fragen oder eine Anfrage an die Wissensbasis zu schicken. Sobald der Dialogmanager alle zur Verfügung stehenden Informationen gesammelt hat, formuliert er eine Antwort, die auch eine weitere Frage an den Benutzer sein kann. Diese Antwort wird vom Dialogmanager nur inhaltlich erstellt. Die Ausformulierung in Worten geschieht im Antwortgenerator. Zum Schluß spricht der Synthetisator diese Worte aus. Zusätzlich zur Sprachausgabe wird eine Zusammenfassung der wichtigsten Informationen am Bildschirm dargestellt.

Es gibt 35 Hotels in dieser Preiskategorie. Könnten Sie den Ort etwas genauer

beschreiben? Oder vielleicht weitere Leistungen, die Sie möchten?

Warnung! Der linke Scheinwerfer is defekt.

Nur noch ein halber Liter Benzin!

Straßenname unbekannt. Bitte buchstabieren.

Wie weit wollen Sie heute noch fahren?

Bei der zweiten Ampel links abbiegen! Dies ist die Franz-Huber-Str. Dann zwei Straßen weiter fahren. Das "Goldener Adler Hotel" steht auf der rechten Seite.

A1.2 Set B: Sentences and phrases to check the pronunciation

Die Erinnerung an die vergangenen Schwierigkeiten.

Von jeher habe ich das befürchtet.

Die zwei jeweils quergelegen Balken.

Er kämpft verbissen.

Die Journalisten führen Regie.

Deutschland und Italien in einem Studium über Mechatronik

Der Ingeniuer auf der Montage.

Montage sind grundsätzlich belegt.

Der Ministerialrat.

Das Milieu.

In der Hölle gibt es eine Höhle.

Man trägt keine Hüte in der Hütte.

Er stahl es aus dem Stall.

Ich biete mich an mit einer Bitte.

Es ist eine Wonne, hier zu wohnen.

Das Lamm ist lahm.

Ich gäbe ihm das, was ich anderen gebe.

Das Doppeljubiläum des Werbejahres

A1.3 Set C: Sentences to check word and sentence prosody

1. Die Ergebnisse sehen im Detail wie folgt aus:

Für den Monat Mai wird eine Pauschalzahlung in Höhe von 120 Euro gezahlt; für Auszubildende wird der Betrag entsprechend angepaßt.

Die tariflichen Leistungen (Löhne, Gehälter und Ausbildungsvergütungen) werden mit Wirkung zum 01.06.2002 um 4,0% erhöht. Zum 01.06.2003 werden sie um weitere 3,1 % erhöht.

Darin sind auch die sogenannten ERA-Strukturkomponenten in Höhe von 0,9% bzw. von 0,5% enthalten, die anteilig zum Juli/April 2003 und September 2003 als Einmalbeträge ausbezahlt werden. Nach Einführung von ERA bei DaimlerChrysler fließt das Volumen der ERA-Strukturkomponenten in das neue Vergütungssystem ein.

Das Verhandlungsergebnis sieht eine Laufzeit von 22 Monaten vor.

Erfreulich war die Absatzsteigerung bei der A-Klasse um 11% auf 47.200 Fahrzeuge, die vor allem auf die hohe Nachfrage nach der Variante mit dem verlängerten Radstand zurückzuführen ist.

Vor der 15. Fußball-Weltmeisterschaft sah sich DFB-Teamchef Rudi Völler mit äußerst schwierigen Bedingungen bei der Vorbereitung konfrontiert.

Eine neue Wagenfarbe ist ebenfalls im Angebot: die Metallic-Lackierung Cubanitsilber.

Eventuell profitieren er und seine Spieler von der günstigen Auslosung, nach der ein Duell gegen einen der drei Top-Favoriten erst im WM-Finale am 30. Juni in Yokohama möglich ist.

Muß der Zucker nicht dort drüben stehen?

Wer kann mir sagen, wo der Zucker ist?

Eine andere Ausdrucksweise dafür ist, daß das freie elektromagnetische Feld bei seiner Zerlegung nach ebenen Wellen nur aus transversalen Wellen besteht, d.h. aus Wellen, für die der Polarisationsvektor raumartig und orthogonal zum Wellenvektor ist.

A1.4 Set D: Sentences and Phrases to check language-specific transcriptions

Mit DM 37,95 um 14.25 Uhr am 21. April 1997.

Er ist 1,78 m groß und wiegt 87,5 kg.

Um 0.15 Uhr fährt der Zug.

Der 1. Mai ist ein Feiertag.

Ein 2ter Versuch schlug fehl.

Mit 9,7%iger Steigerung.

Die Modelle C 240 und C 270 CDI sind mit ABS standardmäßig ausgestattet.

Die H7-Lampen.

Mit $1,7 Mrd. haben wir den höchsten Umsatz.

Der weltgrößte Hersteller von V12-Motoren für Pkw produziert im Werk Berlin den Turboantrieb der neuen Edelmarke Maybach.

Das Brett ist 11,65 m lang und 11,65 cm breit.

Er schwingt mit 19,07 kHz.

E/E-Konzepte als Basis für zukünftige Fahrzeuginnovationen

Für unsere Kunden äußert sich Prof. Dr. Günter Hertel.

Planen mit Pioniergeist: Das Dach im Rohbau ist dicht und die Stahlkonstruktion für Lackierung und Montage sind bereits zur Hälfte fertig. Die Erweiterung der Werkes Tuscaloosa im US-Bundesstaat Alabama liegt voll im Plan.

Um 4.00 Uhr betrug die Temperatur 17,3 Grad Celsius.

Er wohnt in der Rothstr. neben dem Bäcker.

A.2 Evaluation sentences for English TTS systems

A2.1 Set A: Sentences from the VICO Scenario

The route is being planned.

Which destination do you want?

I'm sorry. I can't understand you. You might want to restart the system.

Please spell your destination.

VICO can help you with the following tasks:

find the address of a destination, such as a hotel, from general information you provide

find telephone numbers of hotels and

explain the capabilities of the VIVO system.

VICO is an advanced prototype spoken dialog system. Although internal quite complex,

the basic structure is fairly straight forward. The main components which process your input and produce audio and visional output are:

the speech recogniser

the natural language understanding module

the dialog manager

the response generator

the speech synthesizer

and finally the screen.

In addition to these component, VICO also uses GPS, the global positioning system, to locate your current position and possesses a central database which contains both geographical information on streets, the location of hotels and other items of interest as well as other information such as prices, opening times, other attractions, etc. Once you press the speak button, VICO goes active. When you speak, the speech recogniser transforms your speech into words, which are then analysed for their meaning by the language understanding module. This information is then fed to the dialog manager which decides what action to take. This may include consulting GPS for the current location and sending an inquiry to the database. Once it has all the information it can get, it formulates a response, which may be a further question to the user. This response, which the dialog manager specifies only in generic terms, is sent to the response generator, which actually generates the words of the response. Finally, the speech synthesizer speaks these words to the user. In addition to this audio output, VICO presents a summary of the most important information visually on the screen.

There are 35 hotels in this price category. Could you specify the desired location any closer? Or specify any other services you would like?

Warning! Your left front headlight is out.

Only an eight of a gallon of gas left.

This street name is unknown. Could you spell it?

How far do you want to go today?

Turn left at the second stop light. This is Winchester Ave. Then go two blocks. The "Golden Nugget Hotel" is on the right.

A2.2 Set B: Sentences and Phrases to Evaluate Pronunciation

Did he whine about the wine?

He usually prepares the appropriate questions very carefully.

Why choose white shoes?

He has a great eye for grey ties.

A trite solution at right angles.

The average tube is practically a new piece of equipment.

The simplest way is curiously the least intuitive.

The temporary dimensions are specified under the assumption that the length and width have been accurately measured.

The hew and cry of the people.

The large meals in the tower were shared with the help.

The three warrior oiled their tools with a rag.

The various power tools are valuable for keeping deviations low.

He wants to go to a party.

He's looking at a picture.

What picture is he looking at?

A2.3 Set C: Sentences and Phrases to Evaluate Prosody

Lee is accused of having received $13,800 from sports-lottery operator Tiger Pools International to secure a license in 2001.

Just one week after Los Angeles residents learned they will vote this November on whether the San Fernando Valley should secede from the city, word came that the entertainment capital known as Hollywood could also survive on its own.

If voters decide to break off both areas, it would reduce L.A.'s population by approximately 1.5 million people, or roughly 40%.

One idea that researchers have begun to test is temporarily suppressing the body's natural oestrogen and thus providing birth control along with protection from breast cancer.

In 1978, when he became the first non-Italian Pope in more than four centuries, John Paul II made sure to bring along from Cracow his trusted personal secretary, Monsignor Stanislaw Dziwisz, who started working for him in 1966.

It's an epic voyage, drifting down all 4,880 km of Southeast Asia's longest river. Gargan begins at the Mekong's source in the thin air of the Tibetan plateau and goes with the flow until it reaches the South China Sea.

Last month Shama's husband, a Brooklyn native of Lebanese descent, failed to pick up on a soldier's signals as he crossed the checkpoint and suddenly found the red laser dot of the Israeli's rifle sight dancing on his face.

The new hybrid Civic uses a smaller electric motor and a more powerful gas engine that the Prius, so it's always burning gas, except when it's braking or standing still.

A2.4 Set D: Sentences and Phrases to Evaluate Transcription

The 3rd biggest house on Summerset Dr. belongs to Dr. Smith.

Mr. and Mrs. Jones have just bought the Meyers & Son Publishing Co.

On Tuesday, Sept. 11, 1984, shoes were sold for $39.95 a pair.

The room measures 8'6" by 11'9".

It has a resonant frequency of 245.45 Hz.

They lead in the competition.

It has a lead coating.

He read the newspaper.

You can read in the newspaper.

They took appropriate measures.

They will appropriate the money.

He didn't separate it into separate piles.

He bowed to the audience.

The board was bowed out of place.

The winds came from the south.

The road winds up the mountain.

At 10:24 p.m. we'll arrive in Munich.

It weighs 3.76 lbs. when dried.

The ABC company sells widget at $.35 a piece.

He used to come early.

A hammer is used to pound nails with.

They have to leave tomorrow.

He's supposed to come today.

It's at the crossing of Washington Blvd. and Central Ave.

There's a drugstore on 42nd St. not far from my home.

He goes to St. Ann's church.

He has a lot to do. What do you have to do?

A.3 Evaluation sentences for Italian TTS systems

A3.1 Set A: Sentences from the VICO Scenario

Sto calcolando il percorso.

Dove vuole andare?
Mi spiace, non riesco a capire. Forse è meglio riavviare il sistema.
Faccia lo spelling della destinazione.
VICO può collaborare nei seguenti modi:

1. trovare l'indirizzo di una destinazione, tipo un hotel, dalle informazioni generali fornitegli
2. trovare il numero di telefono di hotel e
3. spiegare le capacità del sistema VICO

VICO è un prototipo avanzato di sistema di dialogo vocale. Anche se è sistema complesso, la sua struttura di base è piuttosto semplice. Le componenti principali che elaborano la voce dell'utente e producono risposte vocali e visuali sono:
il riconoscitore vocale
il modulo di comprensione del linguaggio naturale
il gestore del dialogo
il generatore di risposte
il sintetizzatore vocale
ed infine lo schermo video.
Oltre a queste componenti, VICO usa anche il GPS, il sistema di posizionamento globale, per individuare la posizione e possiede una base dati centrale che contiene informazioni geografiche sulle vie, le posizioni degli alberghi e di altri punti di interesse ed anche altre informazioni come prezzi, orari di apertura, altre attrazioni, eccetera. Se si preme il pulsante voce, VICO si attiva.
Quando si parla il riconoscitore converte la voce in testo, il cui significato viene poi analizzato dal modulo di comprensione del linguaggio.
Questa informazione viene poi inoltrata al gestore del dialogo che decide quale azione intraprendere. Ciò può comportare la consultazione del GPS per determinare la posizione attuale o l'invio di una richiesta alla base dati.
Una volta raccolte tutte le informazioni, viene generata una risposta o una ulteriore domanda all'utente. La risposta, specificata dal gestore del dialogo solo in termini generici, viene mandata al generatore di risposte che la traduce in una frase vera e propria. Infine il sintetizzatore vocale si rivolge all'utente pronunciando la frase. Oltre alla risposta vocale, VICO riassume le informazioni principali sullo schermo video.
Ci sono 35 hotel in questa fascia di prezzo. Può specificare meglio la località? Oppure specifichi degli ulteriori servizi che desidererebbe.
Attenzione! Il faro di sinitra è bruciato.
Resta un solo litro di carburante.
Giri a sinistra al secondo semaforo. Questa è Via Mazzini. Poi vada avanti per due isolati. L'hotel Buonconsiglio è sulla destra.

A3.2 Set B: Sentences and Phrases to Evaluate Pronunciation

Capitano tutte al capitano.
Il bagnino non vuole che bagnino i fiori.
Siamo consapevoli che si tratta di un problema di dimensioni molto ampie che provoca non pochi disagi.

Dopo aver lasciato i bagagli nella camera d'albergo, sono usciti.

E quei tre quarti d'ora o quell'ora di accesso agli sportelli dopo pranzo si rivelano molto comodi per gli utenti.

Nella vita non c'è niente di sicuro e di preciso.

Il peggio sembra passato anche se uno scoraggiamento generale tuttora sussiste.

Sul mare ci sono nove navi nuove una delle nove non vuole navigare.

Sopra la panca la capra campa, sotto la panca la capra crepa.

No, non ho un nonno.

Trentatre trentini entrarono in Trento, tutti trentatre trottando.

C'è il questore in questura a quest'ora?

Sotto quattro grossi sassi, quattro gatti grossi e grassi.

Remo rema sul Reno con remi di rame.

A3.3 Set C: Sentences and Phrases to Evaluate Prosody

Non c'è dubbio, infatti, che l'Occidente e i suoi abitanti facciano la parte del leone in quest'orgia di esaurimento delle risorse naturali, mentre i paesi in via di sviluppo, nei cui territori spesso si trova gran parte di queste risorse, subiscono quasi esclusivamente le conseguenze del saccheggio degli ecosistemi.

Fondamentale, per capire i fili e le trame che legano quelle intercettazioni agli sviluppi di queste ore, è la lettura attenta di una conversazione registrata il 24 gennaio 2001 alle ore 20 e 07.

A Piazza Affari l'indice Mibtel segna un aumento del 1,76% a quota 19.760 punti, mentre il Mib30 sale del 2,07% a quota 26.997 punti.

La massa finanziaria che deriva dalle contribuzioni dovrebbe servire all'assistenza degli anziani non autosufficienti, dei malati cronici, disabili e lungodegenti.

Una situazione che, a fine 2002, dovrebbe causare una diminuzione della produzione pari a 3,5 miliardi di euro, più del doppio della già pesante flessione dell'anno scorso che si era fermata ad un -1,7 miliardi.

A3.4 Set D: Sentences and Phrases to Evaluate Transcription

La terza casa più grande su Viale Bellavista appartiene al dr. Rossi.

Il sig. e la sig.ra Berlusconi hanno appena acquistato la Arnoldo Mondadori Editore Spa.

Martedì 11 settembre 1984 le scarpe si vendevano a 40 euro il paio.

La stanza misura 4,3 x 5 m.

Ha una frequenza di risonanza di 245,45 Hz.

Alle 22.24 arriveremo a Monaco.